UNIVERSIDADE BEIRA INTERIOR
Engenharia

Android em Homebanking
Manutenção e Implementação de Funcionalidades em Aplicação Android de Homebanking

Mariana Magalhães Dantas

Relatório do projeto de estágio
Engenharia Informática
(2º ciclo de estudos)

Orientador da UBI: Prof. Doutor Pedro Inácio
Orientador da *ITSector*: Eng. António Faneca

Covilhã, Setembro de 2021

Agradecimentos

A elaboração deste estágio contou com vários apoios, incentivos e orientação por parte de várias pessoas.

Assim, gostaria de deixar especial agradecimento:

- Ao Professor Doutor Pedro Ricardo Morais Inácio pela orientação, apoio e incentivos dados, pelas ideias e sabedoria transmitida e a cima de tudo pela sua presença e total disponibilidade para esclarecimento de dúvidas.
- Ao António Daniel Neves Faneca pela predisposição e orientação na empresa, pelas ideias e dicas, pelo seu acompanhamento e ajuda.
- A toda a equipa da ITSector que tive o prazer de trabalhar e ter contacto, pelo apoio e pela ajuda e pelo bom acolhimento.
- À Professora Doutora Maria Manuela Areias da Costa Pereira de Sousa, regente da cadeira *Projeto de Dissertação ou de Estágio em Engenharia Informática*, por disponibilizar a sua ajuda e dicas para a elaboração do projeto e da sua planificação como também proporcionar o contacto inicial com a empresa.
- Ao Professor Doutor Hugo Pedro Martins Carriço Proença, diretor do curso de mestrado em engenharia informática, pelo apoio dado a todos os alunos e a sua presença indispensável para resolução de problemas.
- A todos os docentes e regentes das cadeiras pertencentes ao mestrado em engenharia informática, pelas boas práticas e pelos conhecimentos passados aos alunos, sem eles a elaboração e idealização tanto prática como teórica deste estágio não seria possível.
- Aos meus amigos, amigas e colegas em especial, Valeriya Popyuk, Rita Correia e Maria Beatriz Pereira, pelo apoio e força.

Por último, agradeço à minha família, em especial a minha mãe Ivone ao meu pai Avelino e ao meu irmão Mauro, por estarem presentes nesta jornada e por apoiarem as minhas decisões, pelos ensinamentos de vida e por me criarem e acompanharem no crescimento não só na vida como na minha vida académica, por serem modelos de coragem e incentivo.

Prefácio

Atualmente o Ser humano está a ultrapassar o que pode ser uma das maiores pandemias de uma geração e com esta veio a necessidade de distanciamento social e de executar muitas das tarefas diárias através de casa. Com o avanço das tecnologias e dos *smartphones* surgiu também a possibilidade de executar várias tarefas a partir de um computador ou telemóvel, desde que este tenha acesso à *Internet*. Esse crescimento deu aos bancos portugueses e internacionais a possibilidade de executar certas tarefas, que só eram possíveis executar num banco ou multibanco, através da *Internet*. A essa forma de executar operações bancárias à distância deu-se o nome de *Homebanking*.

Resumo

Partindo do princípio do *Homebanking* e da sua exploração no sistema operativo *Android*, este relatório descreve um estágio na empresa *ITSector* no projeto M24, sendo este um projeto cujo cliente é o Banco Montepio e que conta com aplicações *Android*, *iPhone Operating System (iOS)* e *Web*.

Como tal, inicialmente, foi feita uma exploração e caracterização da empresa e dos objetivos do estágio. Com estes em mente foi estudado o estado da arte, expondo o que é *Homebanking* e algumas das suas vertentes no ramo *Android*.

Tendo em conta que este estágio teve um grande foco na plataforma *Android* do projeto, apenas foram expostas as ferramentas usadas neste ramo e pela empresa *ITSector* em específico.

Numa primeira fase foi criada uma planificação do estágio e identificados os riscos e os respetivos planos de mitigação. Numa segunda fase foi posto em prática os conhecimentos adquiridos tanto nas cadeiras universitárias como também na *Academia Android* administrada pela empresa.

No que toca ao desenvolvimento, foram abordados vários pontos, desde testes automatizados, criação de *layouts*, criação de novas funcionalidades como um sistema de *logs*, alteração e atualização do *Software Development Kit* - Kit de Desenvolvimento de *Software* (SDK), segurança da informação, o programa de Arredondamentos e por fim como gerar uma nova versão de uma *Application* - Aplicação (APP).

Como pesquisa adicional e anexos foi incluído neste relatório um mapa de *Gantt*, um capítulo sobre a ética e essa vertente no mundo financeiro e bancário, e um relatório criado com o intuito de expandir o conhecimento sobre testes automatizados e as diferentes técnicas utilizadas.

Palavras-chave

Android, Banco Montepio, *Homebanking*, *ITSector*, M24, Planificação, Testes Automatizados

Abstract

Based on the Homebanking principle and its exploitation in the Android operating system, this report focuses on the internship in the company ITSector in specific the M24 project, which is a project whose client is Montepio Bank and which has applications in Android, iOS and Web.

As such, initially, an exploration and characterization of the company and the objectives of the internship was carried out. With these in mind, the state of the art was studied, exposing what Homebanking is and some of its aspects in the Android branch.

Bearing in mind that this internship was carried out in the Android component of the project, only the tools used in this branch and by the ITSector company in particular were exposed.

In a first phase, an innernship plan was created and the risks and the respective mitigation plans were identified. In a second phase, the knowledge acquired both in university and in the Android academy administered by the company was put into practice.

Regarding development, several points were raised from automated tests, creation of layouts, creation of new features such as a log system, alteration and updating of the SDK, information security, the Rounding program and finally how to generate a new version of an APP.

As additional research and attachments, a Gantt map was created, a chapter on ethics and this aspect in the financial and banking world and a report created with the aim of expanding knowledge about automated tests and the different techniques used.

Keywords

Android, Automated Tests, *Homebanking*, *ITSector*, M24, Montepio Bank, Planning

Índice

Lista de Figuras

Listings

Acrónimos

AES *Advanced Encryption Standard*
API *Application Programming Interface*
APK *Android Package*
APP *Application* - Aplicação
BCP Banco Comercial Português
BPI Banco Português de Investimento
CMMI *Capability Maturity Model Integration* – Modelo Integrado de Maturidade e de Capacidade
COVID *Corona Virus Disease* - Doença Corona Vírus
DES *Data Encryption Standard*
DEV Desenvolvimento
ENEI Encontro Nacional de Estudantes de Informática
ESN *Electronic Serial Number*
GUI *Graphical User Interface* - Interface Gráfica do Utilizador
ID *Identifier*
IDE *Integrated Development Environment* - Ambiente de Desenvolvimento Integrado
IDEA *Integrated Development Environment Aplication* - Ambiente de Desenvolvimento Integrado de Aplicações
IMEI *International Mobile Equipment Identity*
iOS *iPhone Operating System*
ISO *International Organization for Standardization*
IVA Imposto sobre o Valor Acrescentado
JSON *JavaScript Object Notation*
MDW *Middleware*
MEID *Mobile Equipment Identifier*
RC4 *Rivest Cipher 4*
RSA *Rivest-Shamir-Adleman*
SDK *Software Development Kit* - Kit de Desenvolvimento de *Software*
SMS *Short Message Service* - Serviço de Mensagens Curtas
PDF *Portable Document Format* - Formato de Documento Portátil
PIN *Personal Identification Number* - Número de Identificação Pessoal
PMC *Plataforma MultiCanal*
UBI Universidade da Beira Interior
UI *User Interface*
URL *Uniform Resource Locator*
VPN *Virtual Private Network*
XML *Extensible Markup Language*

Capítulo 1

Introdução

O presente relatório de estágio tem como objetivo principal a introdução da empresa onde o estágio foi realizado, a apresentação desse estágio e dos seus objetivos, e também a planificação das tarefas que nele foram realizadas e do seu plano de execução. O relatório serve também como demonstração, explicação e exposição do que foi desenvolvido e realizado ao longo do estágio.

Anteriormente ao planeamento do estágio ou de qualquer execução, é necessário ter em vista quais os objetivos a alcançar e como será possível desenvolver os mesmos, sendo esse o foco da secção 1.3. Dado que este estágio curricular se realizou na empresa *ITSector*, é também imperativo a sua caracterização como empresa, aqui incluída na secção 1.1. Ainda neste capítulo podemos encontrar a secção 1.4 – *Organização do Documento*, que descreve com mais detalhe os capítulos deste documento.

Tendo como base os objetivos gerais de um estágio curricular, este consiste numa preparação para o mundo real do trabalho em empresas no ramo da informática. Desse modo, o projeto em que este estágio incidiu, foi um projeto real e de contacto com o cliente mais antigo da *ITSector*: o Banco Montepio.

1.1 Caraterização da Empresa

O estágio desenvolveu-se na empresa *ITSector*, que é uma empresa 100% portuguesa, com mais de 15 anos de experiência em transformação digital para o setor financeiro. A *ITSector* é uma empresa de desenvolvimento de *software* especializada em transformação digital para instituições financeiras. Desde 2005 que trabalha na forma de interação de utilizadores de todo o mundo com aplicações empresariais.

Inovando e apresentando conhecimento especializado, esta empresa trabalha em conjunto com o cliente para criar as melhores soluções para o negócio [ITS20a]. Desenvolve projetos para mais de 20 países a partir dos centros portugueses localizados em Aveiro, Braga, Bragança, Castelo Branco, Lisboa e Porto e fora de Portugal, nomeadamente em Berlim, Luanda e Varsóvia. A empresa conta, à data de escrita deste documento, com mais de 550 especialistas que trabalham em conjunto para criar e melhorar soluções personalizadas ao negócio.

A *ITSector* desenvolve soluções de *software* personalizadas, como plataformas bancárias omnicanal, soluções de crédito e aplicações móveis avançadas, para *iPhone Operating System (iOS)* e *Android*. dãom especial foco à integração do *design* de *interface* do utilizador com o *design* de experiência do utilizador para com o objetivo de criar uma experi-

ência agradável.

Esta empresa tem vindo a expandir a sua presença internacional através de um crescimento sustentável e sólido, com foco na criação de soluções inovadoras e disruptivas que trarão clientes, parceiros e instituições financeiras para o futuro da transformação digital. Recentemente, foi adquirida por uma empresa francesa denominada *Alten* para criar postos de trabalho e aumentar as suas ligações com o exterior, tendo também como intuito o aumento de receitas e de especialistas, mas mantendo a sua estrutura interna. Tem como presidente o Sr. Renato Oliveira e como vice-presidentes o Sr. Jorge Ferreira, Sr. Alexandre Viana e o Sr. Jorge Bravo.

Os projetos de desenvolvimento de *software* da *ITSector* foram avaliados no nível de maturidade dois do modelo *Capability Maturity Model Integration* – Modelo Integrado de Maturidade e de Capacidade (CMMI) — Desenvolvimento (DEV) v1.3, bem como certificados com Sistema de Gestão de Segurança da Informação em conformidade com os requisitos da *International Organization for Standardization* (ISO) 27 001 [ITS20b].

1.2 Caraterização da Estagiária

A estagiária Mariana Magalhães Dantas, Magalhães da mãe e Dantas do pai, é uma estudante que frequenta o 2.º ciclo em Engenharia Informática na Universidade da Beira Interior (UBI) à data de escrita deste relatório. É natural de Barcelos, nasceu em Abrantes e reside em Espinho.

A estudante tem uma Licenciatura em Informática Web obtida na UBI e finalizada no ano de 2019. Enveredou por este estágio para ganhar mais conhecimentos do mundo do trabalho e da área *Android*. Esse interesse surgiu desde a unidade curricular de *Programação de Dispositivos Móveis*, lecionada no primeiro semestre do terceiro ano do curso Informática Web na UBI e aumentou quando realizou o seu projeto final de Licenciatura subordinado ao tema *Gamificação da atividade física para melhorar a qualidade de vida*. Este projeto final consistiu na criação de uma aplicação móvel, neste caso *Android*, para promover a atividade física. Esta aplicação tinha conexão a uma pulseira *Xiaomi* para contabilizar os passos dados pelo utilizador e, dessa forma, o recompensar.

Antes da entrada na universidade, a aluna frequentou o curso *Ciências e Tecnologia* no Agrupamento de Escolas Coelho e Castro e no Agrupamento de Escolas Doutor Manuel Gomes de Almeida. Ao longo do seu percurso de vida participou também em atividades de voluntariado na área das tecnologias, mais concretamente na *Measuring Team* do *Web Summit* de 2019 e como mentora no evento *Ignite Your Future* de 2019.

Demonstrou interesse também na troca de ideias com outras universidades no mesmo meio das tecnologias, frequentando o Encontro Nacional de Estudantes de Informática (ENEI) de 2018 e 2020, e participando também na organização das primeiras Jornadas de Informática Web, enquanto membro do núcleo de *InfoWeb* no ano letivo 2015/2016, como Secretária, e no ano letivo de 2016/2017, como Tesoureira.

1.3 Apresentação e Objetivos do Estágio

Este projeto de estágio teve, como objetivo principal, a implementação de funcionalidades na aplicação M24, e como objetivos secundários a execução de testes automatizados ou de correção de *bugs*. De modo a complementar os conhecimentos prévios nas tecnologias circundantes, a empresa forneceu, como treino inicial, uma academia *Android* para conseguir entregar as competências numa equipa adequada. Para além do treino inicial, foi necessária investigação e estabelecimento do estado da arte, como também a ambientação às diferentes tecnologias utilizadas.

Na generalidade, este estágio teve o propósito da preparação de um estudante universitário para o mundo do trabalho, sendo este especialmente focado nas áreas da programação *Android* e testes automatizados.

1.4 Organização do Documento

Este relatório encontra-se organizado da seguinte forma:

- O primeiro capítulo – **Introdução** – apresenta o enquadramento, a caracterização da empresa, a estagiário e os objetivos deste estágio. Contém ainda a organização do documento;
- O segundo capítulo – **Estado da Arte e Tecnologias e Ferramentas** – descreve o estudo da área de trabalho e das aplicações já existentes neste ramo. Aborda também os conceitos de ataques informáticos possíveis na área *Homebanking*. Descreve também o cliente final da aplicação M24 e dos seus outros serviços. Ainda neste capítulo, são apresentadas as ferramentas utilizadas e estudadas para as tarefas e ainda a metodologia *Scrum* utilizada pela equipa de desenvolvimento na *ITSector*;
- O terceiro capítulo – **Planificação** – enumera e discute as tarefas, o plano de execução, a planificação do trabalho, os requisitos do estágio e os riscos encontrados na altura da planificação;
- O quarto capítulo – **Implementação e Testes** – é composto pela exposição do que foi efetuado na Academia Android e como foi feita a ambientação ao projeto M24. Expõe ainda a planificação e testes automatizados realizados, nomeadamente dos menus dinâmicos e do menu de três níveis. Ainda neste capítulo, é exposto como alterar o *Software Development Kit* - Kit de Desenvolvimento de *Software* (SDK), criar um sistema de *logs* e tornar o mesmo seguro. Por fim, é discutido o programa de arredondamentos e como gerar uma nova versão da aplicação;
- O quinto capítulo – **Conclusões** – contém as conclusões principais retiradas ao longo da elaboração deste estágio e também algumas indicações acerca do trabalho futuro.

Capítulo 2

Estado da Arte e Tecnologias e Ferramentas

2.1 Introdução

Este capítulo está destinado à apresentação dos estudos feitos e pesquisas dos temas considerados relevantes nas áreas em que se inseriu o estágio. Para isto, foi expandido o tema de *homebanking* (secção 2.2) e aplicações já existentes neste ramo (secção 2.3), como também a aplicação em questão (subsecção 2.3.4).

A descrição das ferramentas e tecnologias utilizadas e estudadas é também abordada neste capítulo na secção 2.6. Estas tecnologias vão de encontro ao que já é utilizado pela empresa e pela equipa de trabalho, i.e., as tecnologias e ferramentas a usar são iguais para os membros da equipa a trabalhar na mesma área tal como também a mesma nomenclatura utilizada para homogeneizar o código e o projeto, e ser mais fácil a perceção e utilização do código por outros membros.

Antes de trabalhar com dados dos clientes ou com aplicações que trabalham e gerem dados de extrema importância, é importante estudar os ataques que são possíveis nessa área, para prevenção de ataques e melhoria da segurança de qualquer aplicação e consequentemente a segurança dos dados pessoais do utilizador. Desse modo, foram estudados alguns ataques informáticos no *homebanking* (secção 2.4).

Ainda neste capítulo reflete-se o estudo com mais pormenor do cliente Montepio e as suas soluções *homebanking* (secção 2.5), onde este se encontra no mundo financeiro e tecnológico, os serviços fornecidos (subsecção 2.5.1) e também a sua política de privacidadem ba secção 2.5.2.

No que conta às tecnologias e ferramentas, sendo este um projeto de uma aplicação *homebanking*, há três principais variantes para a sua implementação, a primeira sendo *web*, a segunda *Android* e a terceira *iOS*. Neste caso as funções e tarefas implementadas serão na versão *Android* da aplicação. Como tal o *Integrated Development Environment* - Ambiente de Desenvolvimento Integrado (IDE) utilizado foi o *Android Studio* 4.1.1, sendo esta a versão mais atual deste IDE (secção 2.6.1).

Para ir de encontro com o trabalho em equipa, foram estudadas (e são discutidas na subsecção 2.6.2), as ferramentas *Git* e *Gitkraken* e o *Gitflow*. Já para os testes automatizados as subsecções 2.6.3 e 2.6.3.1 referentes ao *Appium Descktop* e *Intellij Integrated Development Environment Aplication* - Ambiente de Desenvolvimento Integrado de Aplicações (IDEA) respetivamente, descrevem estes *softwares*.

Foi ainda estudada a metodologia *Scrum* e exposta a forma que a equipa a usa para agilizar

e realizar o projeto. Sendo uma equipa formada por pessoas de diferentes sectores e vários entendidos nos assuntos, é necessário utilizar um sistema ágil para avançar com facilidade (secção 2.7).

2.2 *Homebanking*

Homebanking refere-se a serviços bancários móveis, pela *web*, por telefone ou por *e-mail*. As primeiras experiências com bancos *online* foram em 1980. Contudo, o *homebanking* só se tornou popular com o aparecimento da *Internet*, em meados da década de 1990 [Kag20].

Hoje em dia, para obter qualquer informação sobre uma conta bancária ou efetuar operações bancárias, não é necessário efetuar uma deslocação a um banco físico. Para melhorar a experiência do utilizador de um banco, as instituições bancárias começaram a utilizar a *Internet* e as novas tecnologias. Através do *homebanking* é possível fazer uma variedade de operações bancárias, tais como:

- Observar o saldo disponível e o contabilístico;
- Pagar contas;
- Consultar movimentos da conta;
- Fazer transferências bancárias (nacionais e internacionais);
- Obter informações sobre os cartões de crédito;
- Efetuar pagamentos de serviços ou ativar/desativar o débito direto;
- Aderir a novos cartões de crédito;
- Alterar limites de crédito ou opções de pagamento dos cartões de crédito;
- Aceder a informação financeira que permite acompanhar os investimentos;
- Entrar em contacto com um gestor ou obter informações dos contactos da instituição.

Esta forma de comunicação com uma instituição bancária traz vantagens, como, por exemplo, não haver horários definidos para efetuar quaisquer operações referidas anteriormente, permitir ao utilizador ser alertado caso a sua conta seja alvo de ataques e tomar conhecimento imediato do que está a acontecer à sua conta [PT19].

2.3 Aplicações *Homebanking*

Nos dias que correm, quase todas as instituições fornecem *homebanking* e têm uma *Application* - Aplicação (APP) móvel que acompanha essa tecnologia conhecida como *Mobile Banking*.
As aplicações mencionadas a seguir são APPs de diferentes bancos nas quais é possível efetuar as diferentes operações numeradas na secção 2.2.

2.3.1 *Moey!*

A *Moey!* (representada na figura 2.1) é uma aplicação móvel criada pelo Crédito Agrícola. Como todas as aplicações *homebanking*, é possível utilizar esta aplicação para efetuar qualquer operação de *homebanking*.
Uma das vantagens da utilização desta aplicação é a não existência de taxas que o cliente teria de pagar ao levantar dinheiro na zona Euro. Ou seja, não são cobradas quaisquer comissões de levantamento na Zona Euro. Já o resto das aplicações aplica uma taxa de 1.7% sobre o valor do levantamento [Eco20].

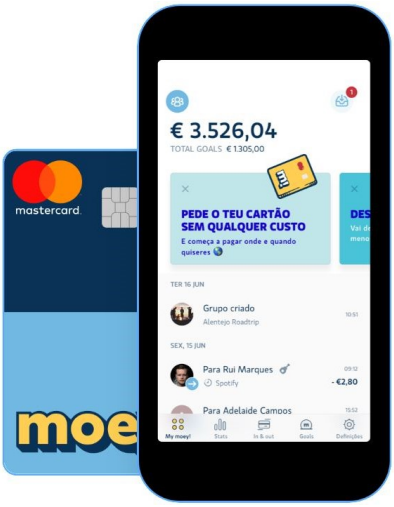


Figura 2.1: Captura de ecrã da Aplicação *Moey!*.

Esta aplicação tem uma funcionalidade interessante que consistem permitir dividir uma despesa com os amigos. Para isso o utilizador cria uma lista de contactos com os quais dividirá a despesa. Quem tiver um *smartphone* da marca *Apple* pode ainda fazer pagamentos com o *smartwatch*.

O *MB Way* está incluído na aplicação e permite fazer até 40 transações por mês sem qualquer encargo associado. Ultrapassando este limite de transferências, são cobrados 50 cêntimos por transferência.

Aos utilizadores da *Moey!* não são cobrados quaisquer custos de abertura ou gestão de conta e podem ter acesso a um cartão físico [Eco20]. Esta aplicação conta com a avaliação de mais de 1200 pessoas na *Play Store*, tendo a pontuação de 4,1 nesta loja [Agr20].

2.3.2 BPI APP

Tal como o Crédito Agrícola, o banco Banco Português de Investimento (BPI) oferece ao cliente a possibilidade de verificar saldos e movimentos, fazer transferências e de consultar as contas de outros bancos. Inclui também as funcionalidades do *MB Way*. Com esta aplicação (representada na figura 2.2), o cliente pode não só organizar de forma automática os seus movimentos, como também configurar alertas e notificações para não se esquecer das suas obrigações financeiras.

Como melhor forma de observar os gastos, a aplicação apresenta também um resumo anual ou mensal das receitas e despesas [Eco20]. Na *Play Store* esta aplicação conta com mais de cinco mil avaliações, tendo uma classificação de 4,1 [BB20].

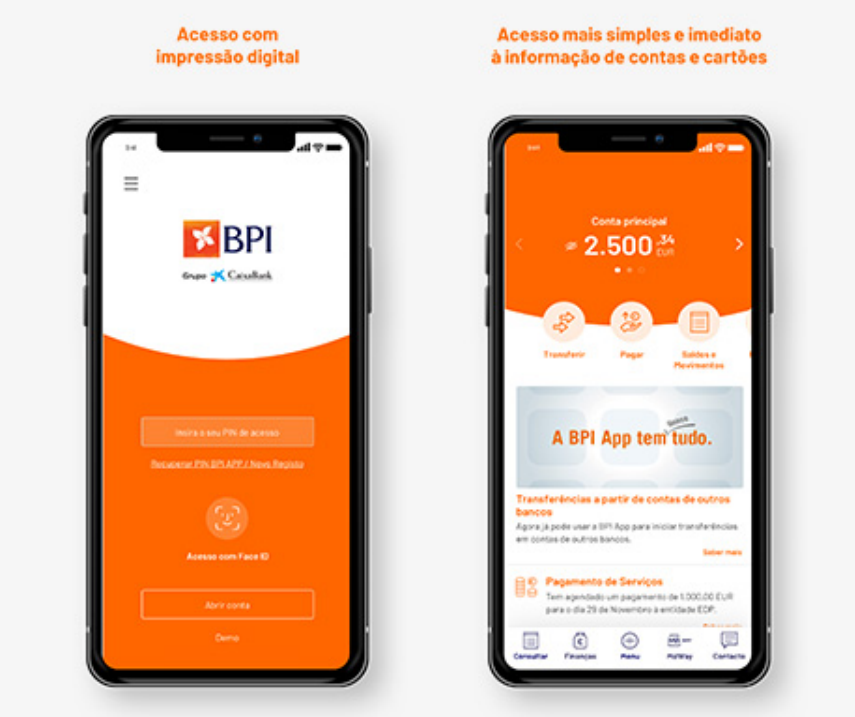


Figura 2.2: Capturas de ecrã da Aplicação BPI.

2.3.3 *Millennium* BCP APP

O banco *Millennium* Banco Comercial Português (BCP) também disponibiliza *homebanking* através da sua aplicação móvel (representada na figura 2.3). Através desta, os clientes podem pedir cartões, ativar cartões de crédito, simular o pedido de diferentes créditos e oferece ainda a possibilidade de poupança com produtos exclusivos da aplicação *Millennium*. A aplicação fornece também uma forma segura de contactar com o banco sem ser preciso uma deslocação ao mesmo [Eco20].

Atualmente esta aplicação apresenta uma classificação de 4,9 na *Play Store* contando com mais de 18,5 mil avaliações [Mil20]. Segundo o banco *Millennium* BCP, esta aplicação é das mais completas, pois fornece ao utilizador as seguintes *features*:

- Cartões;
- Seguro Viagem *ON/OFF*;
- Fundos de investimento;
- Calculadora "Quanto posso pedir?";
- Associação de contas;
- Crédito pessoal com assistente.

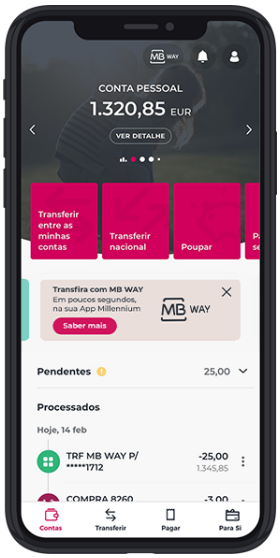


Figura 2.3: Captura de ecrã da Aplicação *Millenium* BCP.

2.3.4 M24 APP

M24 é a aplicação fornecida pelo banco Montepio, que permite ao cliente aceder a esta aplicação após aderir ao serviço Montepio24 (estudado na subsecção 2.5.1). Esta aplicação, tal como todas as anterioremente referidas, fornece a possibilidade de ter "o banco no bolso". Atualmente, a aplicação tem uma classificação de 3,9 na *Play Store* e fornece as seguintes funcionalidades:

- Abertura de Conta;
- *Login* com *Personal Identification Number* - Número de Identificação Pessoal (PIN) Montepio24 ou impressão digital (em equipamentos compatíveis);
- Consulta de saldos e movimentos de contas, cartões de débito e crédito;
- Pagamentos de compras e serviços;
- Pagamentos ao Estado;
- Carregamento de telemóveis;
- Realização de transferências (nacionais/internacionais);
- Agendamento de transferências e pagamento de serviços;
- Ativação de Cartões de Débito e Crédito;
- Pagamento Cartão de Crédito;
- *Cash-advance* através do Cartão de Crédito;
- Carregamento do Cartão Pré-Pago;
- Extravio de cartões;
- Gestão de Utilização de Cartões;
- Constituição, mobilização e reforço de poupanças;

- Mealheiro Digital;
- *Short Message Service* - Serviço de Mensagens Curtas (SMS) *Code* — Adesão e manutenção;
- Beneficiários Frequentes;
- Gestão de Funcionalidades do Serviço Montepio24;
- Gestão de Limites do Serviço Montepio24;
- Documentação Digital;
- Serviço de Alertas;
- Mensagens M24;
- Localizador de Balcões;
- Personalização do *menu* e ecrã de pré-*login* com imagens à escolha [Ger20].

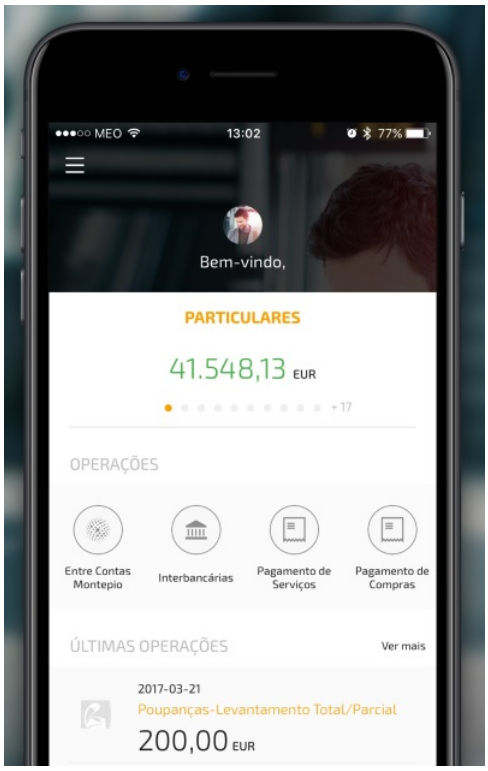


Figura 2.4: Captura de ecrã da Aplicação M24.

Na figura 2.4 está representada a captura de ecrã da página principal da aplicação M24. Este ecrã é apenas visível após o cliente fazer *login* na sua conta. Neste ecrã é possível observar o nome, o saldo e as últimas operações realizadas desta conta. Há a possibilidade de aceder às funcionalidades através dos botões listados no *menu* operações ou ainda abrir o *menu* total clicando no ícone de *menu* no lado esquerdo superior do ecrã. Esta aplicação fornece também a funcionalidade de arrastar a página para o lado para abrir e fechar o *menu* geral.

2.4 Ataques Informáticos

A segurança informática e dos dados do cliente não parte só do banco como também do seu utilizador, sendo que ataques informáticos sobre os clientes são cada vez mais sofisticados e difíceis de detetar. Por isso, o investimento na sensibilização dos utilizadores e na sua proteção é um dos pontos mais importantes no que conta a *homebanking*, pois acima da reputação do banco está a segurança dos bens dos seus clientes.

Estes ataques podem ser de *phishing*, e.g., utilizando *e-mail*, mensagens de texto ou chamadas como também através da fraude de investimentos e *spoof* de *websites* bancários. Esta categoria de ataques encontra-se melhor descrita nas subsecções seguintes, que contêm uma breve descrição destes e de como o cliente só pode resguardar a eventualidade de sofrer um destes ataques.

2.4.1 Fraude de Investimento

Estas fraudes são, por exemplo, propostas de oportunidades lucrativas de investimento em ações, criptomoedas, metais raros, terrenos ou energias alternativas.

Podem ser encontradas se o cliente receber chamadas telefónicas de números desconhecidos repetidamente, se estas ofertas estiverem disponíveis por um tempo curto ou exclusivas a este cliente, ou ainda se forem prometidos lucros rápidos e o investimento for assegurado.

Nestes casos o cliente deve sempre obter aconselhamento imparcial, rejeitar chamadas desconhecidas sobre investimentos, desconfiar sobre ofertas com lucros elevados e rápidos, e reportar estas fraudes à polícia para prevenir novas vítimas [dB21c].

2.4.2 *Phishing*

O *Phishing* usa *e-mails* fraudulentos para burlar o cliente e motivá-lo a forneça os seus dados pessoais, financeiros ou códigos de segurança. Estas mensagens podem parecer idênticas aos enviados por norma por bancos e copiar a aparência com logótipos e estilo visual da mensagem de *e-mails* reais. Podem conter anexos e *links* fraudulentos e até usar linguagem para transmitir urgência.

De forma a prevenir que isto aconteça, é aconselhado ao cliente manter o seu *software* atualizado, examinar o *e-mail* com cuidado comparando o endereço do remetente aos anteriores, ficar suspeito deste pedir dados sensíveis como palavras-passe, encaminhando para o banco *e-mails* suspeitos, não descarregar anexos nem clicar em *links*. É importante referir a necessidade de estar ainda mais atento ao usar o telemóvel, pois pode ser mais difícil detetar um ataque neste tipo de dispositivos móveis [dB21b].

2.4.3 *Smishing*

A palavra *Smishing* vem da junção do acrónimo SMS e a palavra *Phishing*, sendo então este ataque realizado através de SMSs com o intuito de obter dados pessoais, financeiros ou segurança por mensagem de texto. Estas mensagens pedem para clicar num *link* ou ligar para um número de forma a "verificar", "atualizar", ou "reativar" a conta bancária, mas este *link* leva a uma página falsa e o número é do atacante (ou de um intermediário). Deste modo, é imperativo o cliente não clicar em *links*, anexos ou imagens recebidas por SMS não solicitadas e sem conferir o remetente.

O cliente não deve ficar intimidado pelo discurso de urgência, deve conferir tudo antes de responder. Nunca deve fornecer os seus PINs ou palavras-passe e, no caso de responder a uma mensagem suspeita, deve contactar o banco imediatamente através dos meios seguros [dB21e].

2.4.4 *Vishing*

A palavra *Vishing* vem da junção da palavra voz e *phishing* e, tal como os anteriores, este ataque fará com que a vítima forneça as suas informações, contudo neste caso por chamada de voz. Como nos outros casos, o cliente nunca deve fornecer PINs dos cartões de crédito ou débito nem palavras-passe dos serviços *homebanking*, não transferir dinheiro a pedido do atacante, não assumir que a chamada é genuína, procurar sempre o número real e ligar diretamente para lá após dizer que ligará de volta a chamadas suspeitas. Ter cuidado com chamadas não solicitadas e anotar sempre o número que liga para o caso de ser necessário reportá-lo ao banco [dB21a] ou às autoridades.

2.4.5 *Websites* Bancários Falsos

Por norma, os *e-mails* ou mensagens de texto criadas para o *Phishing* ou o *Smishing* contêm *links* que conectam a *websites* falsos, os quais podem ter a mesma aparência que o real e procuram obter os dados do cliente. Estes *websites* falsos podem conter janelas *pop-up* de forma a pedir ao utilizador os seus dados, podem ter alertas de urgência ou ainda um *design* descuidado como erros na escrita e falhas no *design*. Nestes casos (de acesso ao *homebanking*), o utilizador deve utilizar um navegador que não permita janelas *pop-up*, escrever manualmente o *link* real do banco (em vez de clicar em *links*) e se algo importante realmente precisar da atenção do cliente o banco alertará o mesmo após a entrada na conta [dB21d].

2.5 Banco Montepio

O Banco Montepio não é só o cliente mais antigo da empresa *ITSector*, como também a instituição financeira mais antiga de Portugal e única pela sua vocação de instituição de poupança e de disponibilização de serviços financeiros universais a clientes particulares, clientes no setor empresarial e para as instituições da economia social e empreendedores sociais [Mon21]. Também conhecido por Caixa Económica Montepio Geral, este banco tem mais de 300 balcões de norte a sul do país e ilhas, um *website*, e a sua aplicação e os serviços M24 [Mon21].

Atualmente, o banco encontra-se numa reestruturação que passa com o pedido da saída de 900 trabalhadores. Contudo, só foi aceite pelo governo a saída de 400 trabalhadores até 2023. Uma parte destas saídas será para reformas antecipadas, e a outra parte tem de ser aceite pelo governo, tendo o banco o estatuto de empresa em reestruturação. Este estatuto foi pedido a 23 de agosto de 2020, numa carta endereçada à ministra Ana Mendes Godinho pelos administradores do banco Nuno Mota Pinto e José Carlos Mateus [Tei21].

2.5.1 Serviço Montepio24

Para uma gestão diária e de *homebanking*, o banco Montepio fornece os serviços Montepio24. A aplicação móvel M24 só pode ser utilizada se for concretizada a adesão a estes

serviços por parte do cliente. Estes serviços fornecem acesso à aplicação ***Net24*** sendo esta uma aplicação *homebanking* que pode ser acedida através de um computador, *tablet* ou telemóvel através do *website* do banco Montepio. É possível também utilizar o serviço **Netmóvel24**, que permite aceder ao banco através de um telemóvel e realizar as operações e consultas desde que o telemóvel tenha acesso à *Internet* através do *link https://m.bancomontepio.pt*.

O serviço ***Phone24*** é o canal de atendimento telefónico que o banco dispõe, em que o cliente liga para o banco através de um número de aderente. Ao contrário dos outros, este serviço tem um custo acrescido por cada chamada de 0,10 Euros/minuto (+Imposto sobre o Valor Acrescentado (IVA)) para chamadas originadas nas redes fixas e 0,25 Euros/minuto (+ IVA) para chamadas originadas nas redes móveis.

O SMS24 é um serviço bancário que permite aceder ao conjunto de operações e consultas através do envio de uma SMS. Já o SMS *Code* permite realizar transferências e pagamentos no *Net24* e Netmóvel24. De forma a minimizar as fraudes, o banco Montepio não envia nenhum *e-mail* ou mensagem a solicitar a ativação do SMS *Code*. O banco Montepio fornece também um cartão matriz para que o cliente possa validar as operações nos canais à distância. Em janeiro de 2021, o Banco fornecia aos clientes uma nova ferramenta denominada por "APProva" de forma a substituir o cartão matriz o SMS *code* e o SMS 3D *Secure*, e caracteriza este sistema como um sistma de de Autenticação Forte, que permite verificar de forma segura e fiável a identidade do utilizador.

Estes serviços pretendem criar uma maior comodidade ao cliente sem deixar de ter a devida segurança e privacidade de dados e procurando defender o cliente contra os ataques informáticos.

2.5.2 Política de Privacidade

Todos os bancos e empresas que trabalham com dados pessoais têm de ter uma política de privacidade de dados. Esta política tem de ser aceite e cumprida tanto pelo cliente como pelo banco. Neste caso o banco Montepio conta com 12 pontos base para esta política:

1. Compromisso Banco Montepio – onde se descreve quais os pontos a que o banco se compromete;
2. Responsável pelo tratamento – denominação da entidade que trata os dados do cliente;
3. Encarregado de proteção de dados – apresentação do responsável pela proteção dos dados do cliente;
4. Dados pessoais – onde é descrito o que são os dados pessoais, quais são, como são tratados e, porque são necessários;
5. Recolha dos dados pessoais – de forma a prestar serviços o banco requer a recolha dos dados pessoais, neste ponto são descritas as utilizações dos dados e como estes são recolhidos;

6. Finalidades e fundamentos do tratamento – como são tratados os dados e com que finalidade o é feito. Aqui o banco acrescenta também os artigos referentes aos fundamentos legais;

7. Perfis de decisões automatizadas – este destaca as obrigações no âmbito de prevenção do branqueamento de capitais, financiamento ao terrorismo e fraude, no decurso da avaliação da sua capacidade creditícia utilizando o sistema de *scoring*;

8. Transferência de dados pessoais – os dados pessoais podem ser acedidos pelos colaboradores do banco, contudo só no contexto necessário e de comprimento a diligências e obrigações. Este ponto refere ainda quais os elementos têm acesso a estes dados;

9. Prazos de conservação dos dados pessoais – este ponto refere o período durante o qual o banco pode tratar e armazenar os dados pessoais;

10. Direitos dos titulares dos dados pessoais – quais os direitos dos titulares;

11. Medidas de segurança – descrição das medidas de segurança tomadas pelo banco Montepio;

12. Atualização da política de privacidade – quando é que a política foi atualizada e porquê [Mon19].

2.6 Tecnologias e Ferramentas

Esta secção destina-se a explorar e explicar as diferentes tecnologias e ferramentas estudadas e usadas ao longo do estágio. Começando pelo *Android Studio*, passando pelo *Git* e acabando nas ferramentas para testes automatizados.

2.6.1 *Android Studio*

O *Android Studio* é uma ferramenta gratuita e a sua primeira compilação estável foi lançada em dezembro de 2014 [Goo20]. No projeto levado a cabo neste estágio faz-se uso de implementações em código *Java*, recorrendo-se também a *Extensible Markup Language* (XML), sendo que o *Android Studio* fornece uma visualização em tempo real de qualquer alteração visual feita no XML. O XML não é usado apenas para o estilo e o *layout* mas também para o armazenamento de *strings* de forma a facilitar a tradução da aplicação. A aplicação incorpora também a leitura e envio de estruturas de dado no formato *JavaScript Object Notation* (JSON) no que conta aos serviços, em que este serve como formato de comunicação normalizado entre a APP e os serviços.

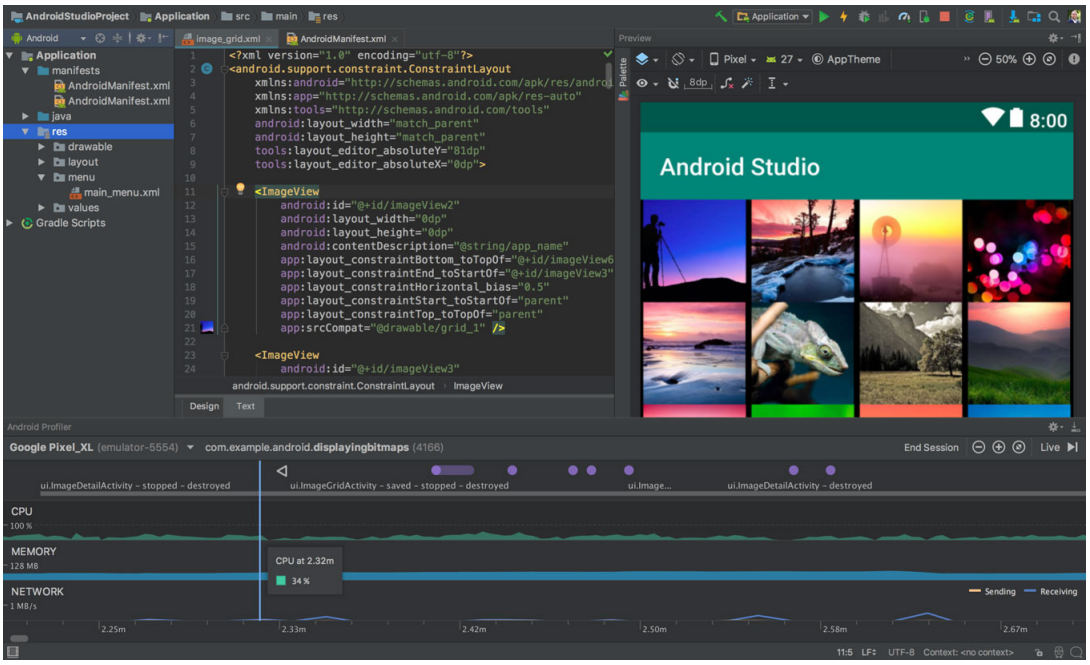


Figura 2.5: Captura de ecrã com a janela do IDE *Android Studio*.

Na figura 2.5 podemos observar a partição que uma janela do *Android Studio* pode tomar, tendo o código e o *design* pontos fulcrais nesta representação.

2.6.2 *Git, GitKraken* e *GitFlow*

De forma a haver um controlo de versões de software e de criar um *workflow* coerente e adequado, este projeto utiliza o *Git* e o *GitKraken*, sendo o primeiro o sistema de controlo de versões mais utilizado no mundo à data de escrita do relatório, e o segundo uma *Graphical User Interface* - Interface Gráfica do Utilizador (GUI) para interagir com o *Git*. O *Git* é um exemplo de sistema de controlo de versões distribuídas, ou seja, em vez de ter apenas um único lugar para o histórico de versões, tem um sistema de subversões.

No sentido de utilizar as tecnologias anteriores, foi necessário estudar o *GitFlow*. O *GitFlow* é o *workflow* que ajuda no desenvolvimento contínuo e na implementação de práticas *DevOps*. Esta foi publicada pela primeira vez por *Vicent Driessen*, definindo um modelo de ramificação projetado em torno do lançamento do projeto [atl20].

Segundo o *workflow* criado por *Vicent Driessen*, o qual podemos observar a figura 2.6, cada *developer* puxa e empurra (representados pelas setas na figura) o software em que está a trabalhar. Estas ações são efetuadas para a origem, contudo cada um destes profissionais pode ter o próprio *branch* de desenvolvimento.

Este modelo tem um repositório central com dois ramos principais: *master* e *develop*, que se encontram destacados a negrito do eixo x da figura 2.6. O ramo mestre, na origem, deve ser o mesmo para todos os utilizadores. Sendo o ramo *mestre* o ramo de produção, é posteriormente criado um ramo paralelo para o desenvolvimento.

No contexto do projeto desenvolvido, existem mais três categorias de ramos:

- *feature branches* – usados para novos componentes para um lançamento futuro;
- *releases branches* – onde são criadas as novas aplicações para produção;
- *hotfix branches* – que surgem quando um *bug* crítico é encontrado na versão de produção e deve ser resolvido imediatamente (e posto de volta na versão de produção) [Dri10].

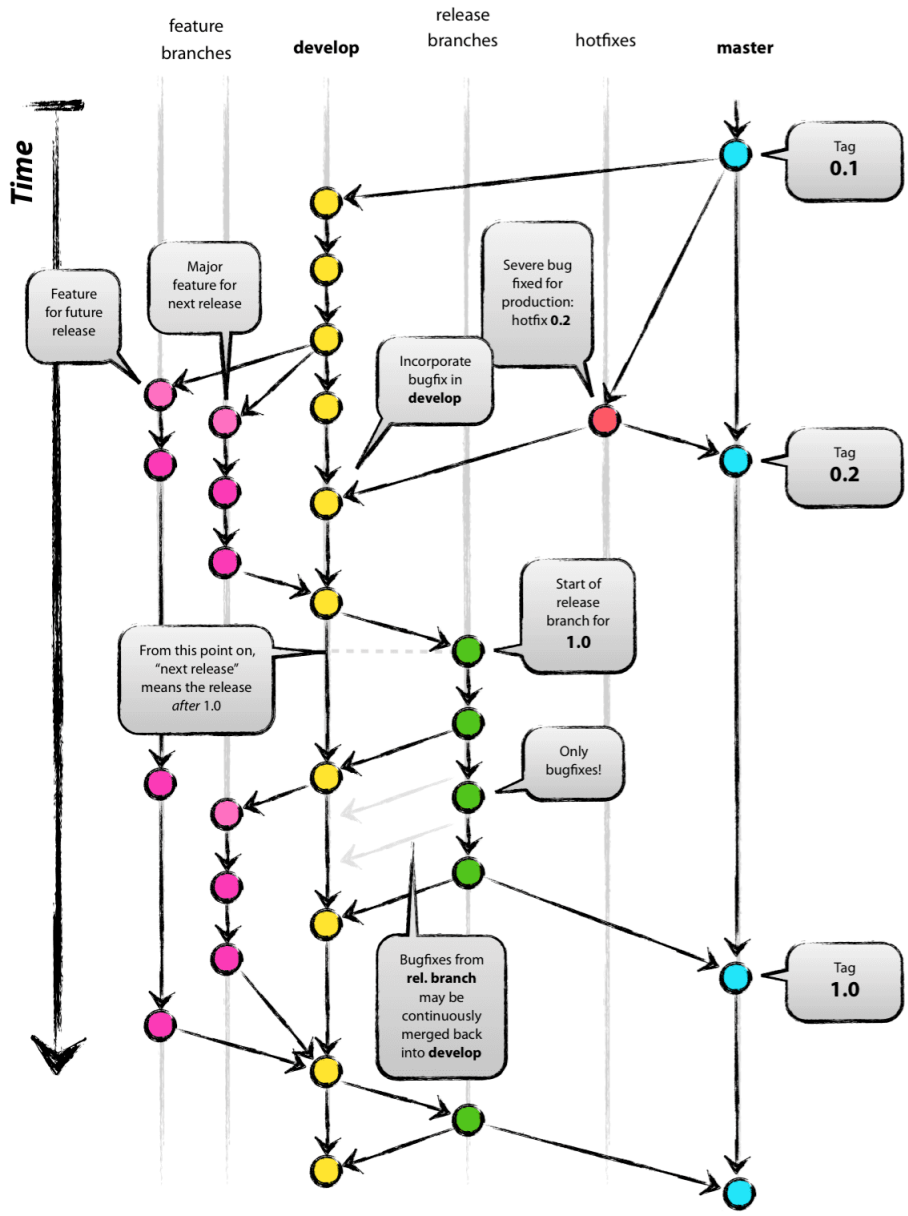


Figura 2.6: Um modelo de ramificação *Git* de sucesso por *Vincent Driessen* [Dri10].

2.6.3 *Appium Desktop*

Após a criação de qualquer componente para a aplicação, esta teve de ser testada, e para isso foi usado o *Appium Desktop*. Esta ferramenta é composta por um conjunto de aplicações para *Mac*, *Windows*, e *Linux* que oferece ao utilizador a possibilidade de automatizar os testes num servidor *Appium* com uma *interface* flexível e bonita. Este *software* é uma

combinação de algumas ferramentas relacionadas ao *Appium*:

- Interface gráfica para o servidor *Appium* – permite definir opções, iniciar ou parar o servidor, ver *logs*, entre outras funcionalidades;

- Inspetor – este pode ser usado para observar os elementos da aplicação, obter informações básicas sobre estes elementos e realizar interações básicas com estes também [Gra20].

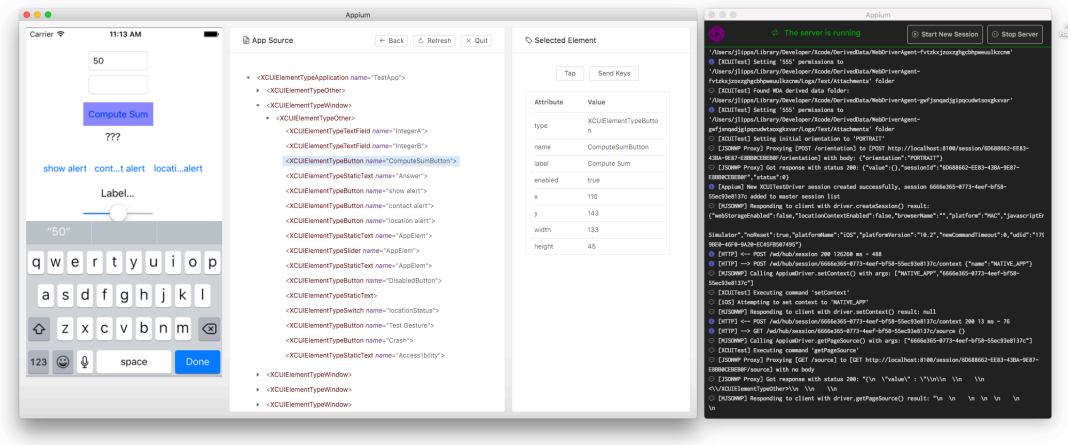


Figura 2.7: Captura de ecrã com a janela do IDE *Appium Desktop*.

Destacam-se aqui quatro partes fulcrais na figura anterior (figura 2.7). Do lado esquerdo, podemos encontrar a *interface* da aplicação visível ao utilizador; uma listagem de todos os elementos encontrados na *interface* que, após seleção de um desses elementos da *interface*, se podem inspecionar com mais detalhe. Do lado direito pode-se observar a janela do servidor *Appium* onde observamos os *logs* das interações, tais como `POST`s/`GET`s feitos.

2.6.3.1 Intellij IDEA

Os testes realizados pelo *Appium* são criados através da execução do código gerado a partir da ferramenta *Intellij IDEA*. Este *software* é apenas um exemplo dos vários editores de código *Java* existentes no mercado. A *interface* deste IDE é bastante parecida à do *Android Studio*, como podemos observar na figura 2.8, sendo que neste podem ser criados e compilados *scripts* para os testes automáticos. Também é possível ver os *logs* produzidos pelos *scripts*.

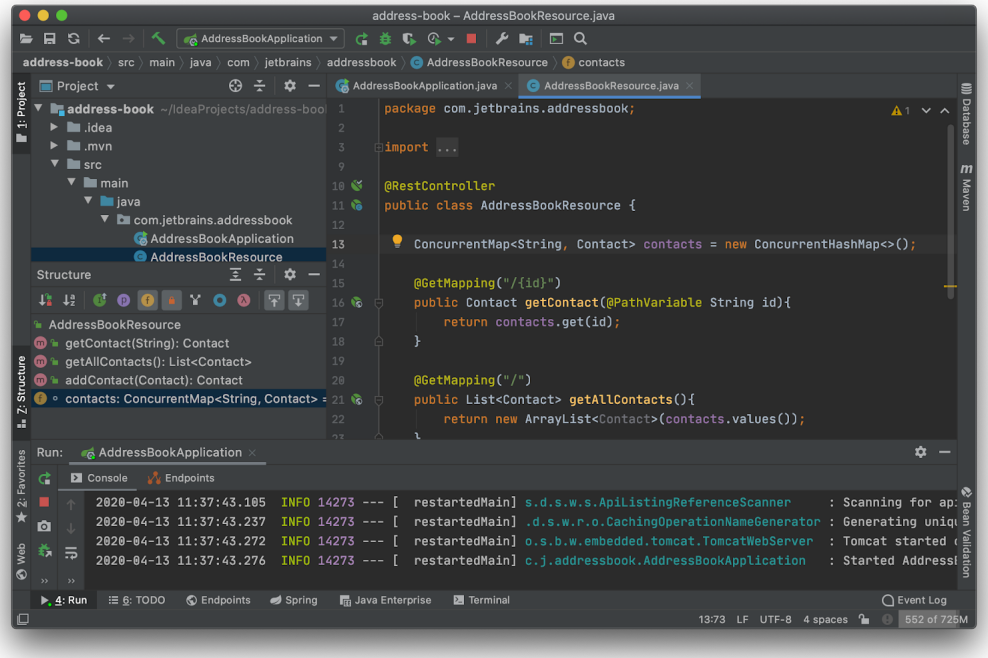


Figura 2.8: Captura de ecrã com a janela do IDE *Intellij IDEA*.

2.7 Metodologia *Scrum*

A metodologia *Scrum* foi utilizada pela equipa para agilizar o progresso e a criação do programa de arredondamentos (Secção 4.10). Esta metodologia é composta por uma série de *sprints* sendo estes ciclos em que o projeto foi dividido. Cada *sprint* tem um tempo fixo e determinadas tarefas a serem executadas antes do final do tempo de *sprint*. No caso do programa de arredondamentos, o tempo de cada *sprint* é de duas semanas. Na figura 2.9 podemos observar figurativamente o funcionamento da metodologia *Scrum* e dos pontos essenciais de uma *Sprint*.

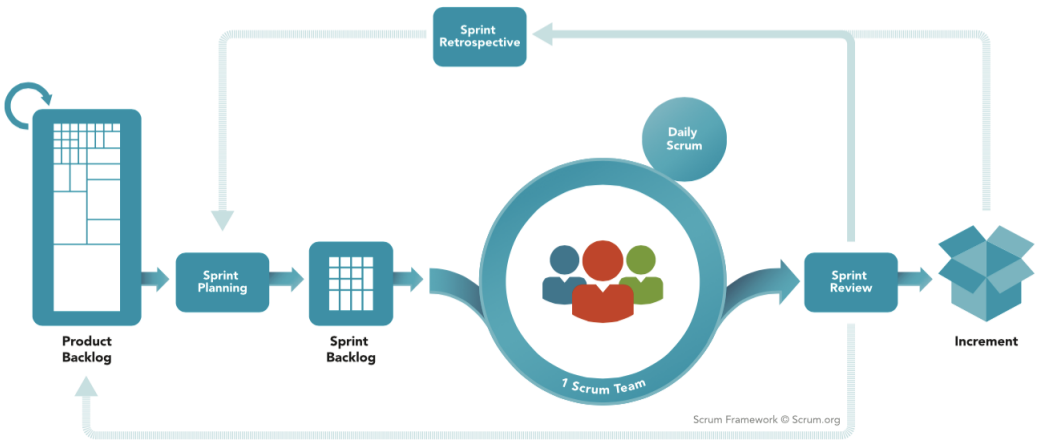


Figura 2.9: Funcionamento da metodologia *Scrum*.

Para alcançar o produto final (*Increment*), é inicialmente criado um *Product backlog*, sendo este uma lista de tarefas a realizar para alcançar o estado final do produto. No início de cada *sprint* é realizada uma reunião de planeamento em que o *product owner* dá a prioridade aos itens de *backlog* criando o *Sprint backlog*. Isto é, o *product owner* escolhe quais as tarefas a realizar nesta *sprint*, dando também a definição de "*pronto*" a qual as tarefas devem alcançar para serem aceites como incremento ao produto final.

Ainda no planeamento da *sprint*, a equipa de desenvolvimento define as diferentes atividades que tem de realizar dentro da *sprint* em tarefas (*tasks*) e definem os *story points* (tempo) necessários para finalizar cada uma das tarefas. No caso da tarefa não ser finalizada, o elemento volta ao *product backlog* para mais tarde ser adicionado a uma nova *sprint* e planear o que falta para alcançar o estado de *pronto*.

Diariamente é realizada uma reunião (*daily scrum*) em que é dado o estado das tarefas por cada elemento da equipa para acompanhar o desenvolvimento da *sprint*, expondo o que foi elaborado no dia anterior, identificar contratempos e priorizar o trabalho para o dia.

Por fim, é feita uma reunião de revisão de *sprint*, onde a equipa apresenta as funcionalidades implementadas sendo dado o estado de "pronto" às tarefas da *sprint*. No fim da *sprint* é feita também uma reunião de retrospetiva, levantando os pontos positivos, os pontos que podem ser melhorados e os pontos já melhorados, para corrigir contratempos e melhorar a produtividade da equipa para a próxima *sprint*, dando assim início a um novo ciclo [Agi14].

2.8 Conclusão

Os serviços de *Homebanking* são, sem dúvida, serviços bastante importantes nos dias que correm. Dando ao utilizador uma abundância de escolhas e possibilidades para estar em contacto com os seus bens, permitindo transferir dinheiro, controlar e consultar as suas contas a partir de casa ou onde estiver. Várias são as aplicações bancárias existentes, cada uma delas para diferentes clientes de diferentes bancos. Estas aplicações não só têm clientes diferentes como também estão projetadas para ir ao encontro do *design* e preferências dos bancos. Algumas dessas aplicações foram revistas neste capítulo.

Com a análise das aplicações já existentes neste ramo, foi possível detetar padrões utilizados por todas elas. Em primeiro lugar as aplicações apenas apresentam a conta do cliente após o *login*, sendo que esta tipicamente a única a qual o utilizador tem acesso; em segundo lugar, após o *login*, a página principal da aplicação apresenta um valor monetário que se encontra ligado à conta do cliente, uma listagem de transações e operações efetuadas nesta conta e um *menu* para fácil acesso às operações mais utilizadas. É ainda de destacar que todas têm um *menu* geral com as possíveis funcionalidades *homebanking* disponibilizadas por cada aplicação em específico. As aplicações estudadas têm um *design* que se mantém ao longo da utilização e dos fragmentos *Android*. Deste modo, muitos dos

Layouts utilizados futuramente nas tarefas de estágio já foram implementados anteriormente.

Muitas das falhas que ocorrem no mundo informático parte de falhas humanas e de problemas criados por utilizadores, sendo muitas vezes o utilizador destas aplicações o elo mais fraco na segurança destes projetos. Desse modo, é importante ter em mente ataques informáticos que envolvem o mesmo de forma a prevenir que aconteça.

Sendo o Banco Montepio o cliente ao qual o projeto M24 pertence, foi fundamental um estudo deste e dos seus serviços. Para entrar e trabalhar num novo projeto é necessário estudar as tecnologias e ferramentas que vão ser utilizadas neste. Começando pelo trabalho em equipa, foi importante haver uma ambientação relativa ao funcionamento do *Git*, do *Git flow* e do *GitKraken*, de forma a obter o código e a perceber a estruturação em *branches* do projeto. Foi também necessária a ambientação com o funcionamento de entrega das diferentes tarefas realizadas, com o intuito de criar uma funcionalidade na aplicação conduzido através da metodologia *Scrum*.

Seguidamente, o projeto M24 concretiza-se para diversos sistemas operativos (e.g., iOS). Contudo, este estágio e respetivo relatório apenas se centram na variante *Android*. Por isso, o IDE *Android Studio* foi estudada também estudada. Por último, para os testes automatizados (expandido no anexo C), para além do estudo destes foi também necessário o estudo da ferramenta *Appium Desktop*. Já o *software Intellij IDEA* foi estudado para a criação dos *scripts* dos testes automatizados.

Capítulo 3

Planificação

3.1 Introdução

Para uma boa execução e um bom percurso de estágio foi necessária a enumeração de tarefas a desenvolver e também o seu plano de execução. Sendo este um relatório de um estágio curricular, desenvolvido no contexto de um projeto real, muitas das tarefas a ser realizadas não eram possíveis de serem apresentadas antemão por não serem do conhecimento da empresa, ou serem de implementação confidencial. Nesse âmbito, as tarefas apresentadas são uma estimativa criada inicialmente de como o estágio iria decorrer e onde seriam utilizadas as horas de trabalho neste projeto.

3.2 Enumeração das Tarefas

De uma forma preliminar, identificaram-se as seguintes tarefas para este estágio:

- Tarefa 1 – Academia *Android*, estudo da plataforma *Android Studio* como também criação de aplicações *Android*;
- Tarefa 2 – Ambientação à empresa e integração na equipa de trabalho. Investigação e ambientação ao projeto M24, com a implementação de funcionalidades *mockup*;
- Tarefa 3 – Revisão do estado da arte e tecnologias importantes. Estudo dos testes automatizados e elaboração de um relatório sobre o mesmo tema;
- Tarefa 4 – Planificação e implementação da especificação de testes para funcionalidade "*Open Banking*";
- Tarefa 5 – Implementação de *Layouts*;
- Tarefa 6 – Planificação, implementação e testes de funcionalidades;
- Tarefa 7 – Escrita do relatório de estágio.

3.3 Plano de Execução

Como referido na introdução a este capítulo, as tarefas a realizar ao longo do estágio não eram do conhecimento inicial. Contudo a execução de qualquer funcionalidade cumpre sempre o mesmo plano de execução, pelo que se pode generalizar:

1. O pedido é recebido, seja este por parte do cliente ou por parte da empresa;

2. A tarefa é planificada, dividida por partes sendo criado um diagnóstico a apresentar ao cliente com estimativa de custos;
3. Sendo a estimativa aceite pelo cliente, é então posta em execução o plano de tarefas, por norma começa pala criação dos *fragments* e *views* da funcionalidade;
4. Estes fragmentos e *view* são interligados com os serviços da aplicação;
5. São criados *layouts* seguindo um modelo criado pela equipa de *design* e aceite pelo cliente;
6. São planificados testes automatizados para cada uma das funções da nova funcionalidade;
7. Os testes são executados de forma a confirmar que não existe nenhum problema;
8. Por fim a funcionalidade é passada para a certificação interna onde são geradas evidências de teste e estas seguem para a certificação do cliente.

Devido ao desconhecimento de quais as funcionalidades a realizar antemão, não é aqui apresentada uma estimativa de tempo.

3.4 Planificação do Trabalho

O Mapa de *Gantt* criado pode ser visto com detalhe no anexo A. Este mapa foi criado a partir das tarefas referidas na secção anterior 3.3 e tem mais precisão nas tarefas realizadas na parte inicial do projeto.

A Tarefa 1 – "Academia *Android*" teve a duração de quatro semanas sendo composta por três *milestones*:

- M1.1 – Primeiro módulo da Academia *Android* concluído com sucesso;
- M1.2 – Segundo módulo da Academia *Android* concluído com sucesso;
- M1.3 – Conclusão da Academia *Android* com sucesso.

O primeiro Entregável (*deliverable*) do projeto é assim o D1 – Projeto final da Academia *Android*.

A Tarefa 2 – "Ambientação à empresa e integração na equipa de trabalho" teve a duração de quatro semanas, as quais serviram como ambientação à empresa, equipa e projeto, e correspondendo às seguintes *milestones*:

- M2.1 – Projeto *Android* e estruturação no *Git* revisto com sucesso;
- M2.2 – Criação do novo menu para a aplicação com sucesso;
- M2.3 – Implementar funcionalidade com sucesso.

Esta tarefa produziu dois *deliverables*, o D2.1 – Aplicação com o novo *item* de *menu*; e o D2.2 – Aplicação com a nova funcionalidade pronta para testes.

No decorrer das quatro semanas de ambientação com a junção das duas semanas seguintes, ou seja, a um total de seis semanas, foram feitos os estudos sobre *homebancking* e aplicações neste ramo como também a testes automatizados, sendo esta a Tarefa 3 – "Revisão do estado da arte e tecnologias importantes". Esta tarefa teve quatro grandes *milestones*:

- M3.1 – Estado da arte referente ao *Git* revisto e descrito com sucesso;
- M3.2 – Estado da arte referente ao tema *Homebanking* e aplicações *Homebanking* revisto e descrito com sucesso;
- M3.3 – Estudo dos testes automatizados revisto com sucesso;
- M3.4 – Ferramenta *Appium* estudada e descrita com sucesso.

A Tarefa 3 produziu o *deliverable* D3 – Estado da arte, sendo este apenas parte do estado da arte final.

No que conta a implementações, a Tarefa 4 – "Planificação e implementação especificação de testes para funcionalidade *Open Banking*" remete para a criação de testes automatizados tendo a duração de quatro semanas e para dois *milestones*:

- M4.1 – Estudo da funcionalidade a testar com sucesso;
- M4.2 – Implementação dos testes com sucesso.

Esta tarefa deu também origem a dois *deliverables*, o D4.1 – Planificação dos testes automatizados; e o D4.2 – Testes automatizados prontos a executar.

Houve também a criação de *layouts* para as funcionalidades da APProva na Tarefa 5 – "Implementação de *Layouts*". Esta tarefa teve apenas o *milestone* M5 – Implementação de *Layouts* e devolveu o *deliverable* D5 – *Layouts* criados e prontos a usar na aplicação.

A Tarefa 6 – "Planificação, implementação e testes de funcionalidades" serve apenas para marcar a posição temporal das tarefas desconhecidas inicialmente.

Por fim, a Tarefa 7 – "Escrita do relatório de estágio" remete para a elaboração dos diferentes relatórios. É dominada por três *milestones*:

- M7.1 – Criação do relatório de Testes automatizados;
- M7.2 – Planificação de Estágio criada e apresentada;
- M7.3 – Estágio terminado e defendido;

Os três marcos anteriores têm *deliverables* correspondentes:

- D7.1 – Relatório Testes automatizados;
- D7.2 – Relatório de Planificação de Estágio e apresentação do mesmo;
- D7.3 – Relatório de Estágio e apresentação do mesmo.

3.5 Requisitos do Estágio

Tendo este projeto como objetivo a integração no mundo do trabalho, foram inicialmente levantadas as aptidões necessárias para ingressar neste. Posto isto, a Academia *Android* fornecida pela própria empresa *ITSector* e os estudos anteriores na cadeira de "Programação de Dispositivos Móveis", lecionada no primeiro semestre do terceiro ano dos cursos de informática na UBI, foi a base para os **conhecimentos *Android*** necessários.

Contudo, este projeto não é constituído apenas pela parte *Android* e a programação de uma aplicação. Também era necessário testar a aplicação. Como tal foram estudados os testes automatizados e realizado um relatório sobre as técnicas e o *software* usado para executar testes automatizados. É ainda relevante referir que a unidade curricular de *Qualidade de Software*, lecionada no primeiro ano do mestrado em Engenharia Informática serviu como base de conhecimentos referentes a testes e técnicas de teste.

Para os testes foi estudada a ferramenta *Appium* e utilizada a linguagem *Java*. Esta linguagem foi estudada ao longo do percurso académico, mais a fundo na unidade curricular de *Programação* do primeiro semestre do primeiro ano do curso de Informática *Web* e na unidade curricular de *Programação Orientada a Objetos*, do primeiro semestre do segundo ano do mesmo curso.

Para trabalhar em equipa é necessária uma rigorosa organização e estruturação, sendo assim necessário **estudar o *Git* e o *Git Flow*** de forma a compreender como o projeto está organizado no *Git* e que as alterações realizadas fossem integradas na *branch* certa do projeto.

A empresa *ITSector* trabalha no setor financeiro e, consequentemente, em soluções ***Homebanking*** de modo que foi necessária também a familiarização com este setor e com estas soluções, acrescentando ainda que todo o código trabalhado será **confidencial**. Foi assinada uma declaração de confidencialidade de forma a resguardar propriedade da empresa e do cliente, como também soluções que ainda não foram dadas a conhecer ao público.

Por fim, como o mundo está a ultrapassar uma pandemia, o trabalho que seria realizado através dos escritórios da empresa foi feito a partir de casa em **teletrabalho**.

3.6 Análise de Riscos e Plano de Mitigação

3.6.1 Risco 1 – COVID-19

Atualmente vivemos em dias de incerteza e risco, com a pandemia do vírus *Corona Virus Disease* - Doença Corona Vírus (COVID)-19. A implementação do teletrabalho foi algo necessário. Contudo, as medidas de confinamento implicaram o distanciamento e consequentemente não há integração física e contacto pessoal com a equipa sem ser a partir das redes de comunicação *online*.

Este risco esteve sempre a um nível elevado e concretizou-se durante este estágio, isto é, já decorriam e continuaram a decorrer as medidas de mitigação de forma a diminuir as consequências da pandemia, o que acarretou o teletrabalho e o confinamento.

Deste modo, as medidas de mitigação passaram por um acompanhamento diário e reuniões *online* com a equipa, de forma a promover a interação e comunicação do estado do projeto e de equipa.

3.6.2 Risco 2 – Cliente

Sendo um projeto para um cliente externo, a cessão de contrato ou projetos com o banco Montepio poderiam ser alterados. Tal implicaria corte de custos e recursos ou até mesmo fim de contrato. Este risco era tido como sendo de nível baixo, porque não havia indícios de que aconteceria e que se tivesse de executar o plano de mitigação (o que se veio a verificar).

O plano de miticação envolveria a passagem do estágio para outro projeto, outra equipa e podendo implicar o estudo de novos tópicos, tecnologias e ferramentas. Para que tal não acontecesse foi importante que a comunicação entre a *ITSector*, o banco Montepio e a equipa de desenvolvimento fosse harmoniosa.

3.6.3 Risco 3 – Equipamentos

Para o desenvolvimento do projeto era necessário ter os devidos dispositivos para observar os resultados e criar os devidos *scripts* e funcionalidades. Neste caso era necessário um computador com acesso à *Internet*, com as ferramentas estudadas e listadas na secção 2.6 devidamente instaladas e um dispositivo móvel *Android*. Ocorrendo o não funcionamento de algum destes equipamentos, poderia causar a paragem de desenvolvimento. A probabilidade deste risco se tornar real é foi considerada baixa (e não se concretizou), sendo que a *ITSector* disponibilizou o computador e um departamento de correção de problemas de equipamentos.

Contudo, se algo errado tivesse ocorrido, o computador seria enviado para a empresa *ITSector* para ser efetuada a sua restauração ou reposição. O dispositivo móvel foi disponibilizado pela própria estagiária, e seria requisitado um ou a sua reposição no caso de avaria.

3.6.4 Risco 4 – Ferramentas

As ferramentas utilizadas para a programação e execução de tarefas não são da autoria ou pertença da empresa *ITSector*. Qualquer espécie de indisponibilidade e problema criado por estas constituiria um risco para o projeto, mas este risco tinha uma baixa probabilidade de acontecer.

Para minimizar o risco, delineou-se ser necessário fazer um *backup* de toda a informação que está a ser utilizada em qualquer das ferramentas. Na eventualidade de se materializar,

as ferramentas deveriam ser substituídas atempadamente e incorporar o *backup* feito com esta nova ferramenta.

3.6.5 Risco 5 – Tarefas

Por vezes uma tarefa é planeada e por algum motivo não é concretizada, seja por falta de tempo, por razões relacionadas com o cliente, com a empresa ou até mesmo com a a equipa. Este risco tinha um nível médio de probabilidade de acontecer. Como principal medida de mitigação, havia-se estabelecido que deviam ser redirecionados todos os esforços para a concretização da tarefa, sejam horas ou trabalho.

No caso da tarefa ser interrompida, seria necessário a passagem da listagem de recursos disponibilizados até ao cliente de forma a serem remunerados, e expor também a causa de interrupção. Após esta passagem, o cliente ou a empresa poderia pedir a reestruturação da tarefa, ou repensar nas formas de chegar ao mesmo objetivo. Este risco não se materializou.

3.6.6 Risco 6 – Sistemas Operativos

A aplicação a ser desenvolvida no contexto deste projeto estava implementada para *Android* e *iOS*, e havia o risco da não disponibilidade eventual desses sistemas operativos. Contudo, este risco era de nível muito baixo e não se materializou. Era expectável que o cliente informaria da intenção da exportação e exploração da aplicação para um novo sistema operativo sendo que apenas só aconteceria se o alcance deste novo sistema operativo justificasse a despesa do suporte.

3.6.7 Risco 7 – *Google*

Dispositivos *Android* têm uma grande incorporação dos serviços *Google*, sendo o *Android* alegadamente independente da *Google*. Na realidade, um sistema *Android* sem estes serviços é, à data de escrita do relatório, algo improficiente. Por exemplo, para a instalação da aplicação não poderia ser possível o *download* a partir da *play store*, ou até o visualizar um mapa não poderia ser feito no *Google Maps*, e qualquer pesquisa na *Internet* teria de ser feita a partir de um navegador externo. Algo relacionado com a falta destas aplicações e integrações foi tido como um rico com uma probabilidade média de acontecer. Se acontecesse, o mercado ficaria mais fragmentado, sendo necessário o suporte a outro tipo de ficheiros e leitores destes.

Na eventualidade do surgimento da implementação da aplicação com outros serviços senão os da *Google* teria de ser feita a exploração de serviços para substituir os disponibilizados pela *Google* ou até na criação de novos serviços. Este risco não se veio a cristalizar.

3.7 Conclusão

A enumeração de tarefas e o plano de execução delas é relevante numa planificação de estágio, contudo sendo este um estágio mais próximo da realidade, muitas das tarefas ainda não eram do conhecimento do supervisor da *ITSector* no início. Algumas das tarefas foram assim inicialmente definidas de uma forma mais abstrata que outras. Contudo, foi possível planificar e demonstrar as tarefas já realizadas através de uma Mapa de *Gantt* 3.4, apresentando também a duração de cada tarefa. Por fim, foram enumerados os requisitos do estágio de forma a fazer um agrupamento das aptidões e conhecimentos necessários para a realização deste estágio e para além das *soft skills*.

Capítulo 4

Implementação e Testes

4.1 Introdução

Este capítulo aborda as diferentes etapas e tarefas realizadas ao longo do estágio. Cada uma destas têm objetivos diferentes, abordam diversas aplicações e finalidades diferentes. Partindo da planificação criada e do mapa de *Gantt* (que pode ser visto mais pormenores no apêndice A) o estágio teve início com uma Academia Android (discutida na secção 4.2), que teve como objetivo relembrar conceitos e enraizar novos métodos de programação.

Após a Academia, foi atribuído um projeto e equipa a cada elemento/estudante estagiário com base nos resultados e progresso. Neste caso, o projeto ao qual a estagiária foi integrada foi o M24. Como tal, houve uma ambientação ao projeto (foco da secção 4.3) de forma a ter conhecimento do fluxo, classes e organização da aplicação Android. Como forma de completar o progresso e implementação completa de uma aplicação real, foram também planificados e implementados diferentes fluxos de testes automatizados (aqui discutidos na secção 4.4).

De forma a refletir neste capítulo de forma fiel a que foram as necessidades do projeto, da equipa e do cliente, discutem-se a seguir os seguintes tópicos e tarefas, que determinam tambéma subdivisão em secções:

- Implementação de menus dinâmicos (Secção 4.5);
- Criação de menus com três níveis (Secção 4.6);
- Alteração de *SDK* (Secção 4.7);
- Criação de um sistema de *Logs* e segurança para o mesmo (Secção 4.8);
- Programa de arredondamentos (Secção 4.10);
- Gerar novas versões para testes (Secção 4.11).

4.2 Academia Android

A ITSector fornece instrução e ambientação à tecnologia *Android* como ponto de partida e análise de conhecimentos aos estagiários que vão trabalhar com essa plataforma. Esta academia teve a duração de um mês e foi repartida em três módulos:

1. Curso *Udacity Um* – denominado por *"Android Basics: User Input"* tem como intuito introduzir a tecnologia e modo de programação *Android*;

2. Curso *Udacity Dois* – denominado *"Developing Android Apps"* serviu como desenvolvimento e introdução a novos conceitos;

3. Projeto final da Academia – desenvolvimento de uma aplicação *Android* em que foram utilizados os novos conhecimentos e posto em prática o que foi aprendido.

4.2.1 *Android Basics: User Input*

Sendo este o primeiro curso da academia, foram introduzidas as partes básicas de programação e a interligação entre os diferentes elementos que compõem uma aplicação. Relembrando componentes da linguagem Java e como esta é uma linguagem orientada a objetos. Utiliza como exemplo uma aplicação que tem o intuito de criar um pedido (por exemplo, numa pastelaria) em que o utilizador introduz informação como: nome, item de pedido, quantidade e extras. A aplicação cria um sumário do pedido e possibilita ainda enviar essa listagem por *e-mail*.

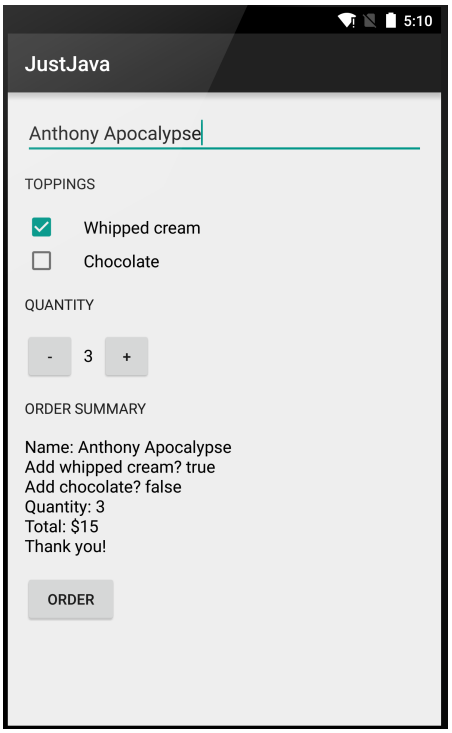


Figura 4.1: Primeira aplicação do curso da Academia *Android*.

Podemos aferir na figura 4.1 que a primeira aplicação pôs em prática os elementos base da criação de uma aplicação Android, desde o *layout* à programação e introdução de lógica. Foi apresentada a possibilidade de utilização e ligação de outras aplicações disponíveis no dispositivo utilizando a capacidade de enviar *Intents* quando enviada informação para outra aplicação. Neste curso foi dado o exemplo de envio de informação para a aplicação de *e-mail*, criando um *e-mail* a partir desta.

Quando o utilizador clica no botão `Order`, que está exemplificado na figura 4.1, o seguinte código é executado (trecho de código 4.1):

```
Intent intent = new Intent(Intent.ACTON_SENDTO);
intent.setData(Uri.parse("mailto:"));
intent.putExtra(Intent.EXTRA_SUBJECT, "Just a order for" + name);
if(intent.resolveActivity(getPackageManager()) != null){
    startActivity(intent);
```

```
}
```

Trecho de código 4.1: Abertura de APP para envio de *e-mail*.

Explicado de forma direta, o código apresentado abrirá uma aplicação que manipula e trabalha com *e-mails* no dispositivo e criará um *email* com a informação enviada como extra no *Intent*.

O mesmo exemplo explorou a forma como se desligam as *strings* da lógica da programação. Tendo em conta que uma aplicação pode ser disponibilizada para vários países, a apropriação da linguagem utilizada deve ser tida em conta. Por exemplo, uma aplicação é disponibilizada para utilizadores ingleses e portugueses. De forma a não ter de disponibilizar duas aplicações diferentes para diferentes utilizadores é apenas apresentada uma aplicação com a possibilidade de seleção de várias linguagens recorrendo à mesma estrutura, programação e lógica, mas a diferentes ficheiros de *strings*. Conforme a localização, linguagem do utilizador e do dispositivo a aplicação utiliza um ficheiro diferente de *strings*. Este ficheiro é criado na pasta *resources* e apresenta as mesmas *labels* para cada *string* contida na aplicação, contudo com diferentes valores.

Um projeto *Android* com estes recursos deverá ter a seguinte estrutura de pastas:

```
MyProject/
      res/
          values/
                 strings.xml
          values-b+pt/
                 strings.xml
```

As pastas *resources* são utilizadas para armazenar os valores da aplicação, tratando-se dos valores das *strings*, apenas o ficheiro *strings.xml* precisa ser alterado. Estando o dispositivo numa linguagem listada nas diferentes pastas de valores, essa pasta será utilizada em vez da pasta predefinida `values`.

Por exemplo, estando o dispositivo em Português o ficheiro *strings* que será utilizado será o da pasta `values-b+pt/`. Caso não esteja em Português, o ficheiro utilizado será o predefinido e contido na pasta `values/`.

Os estilos e temas no *Android* permitem separar as informações de *design* da informação da estrutura e do comportamento da *User Interface* (UI). O tema é composto por uma coleção de atributos aplicados a uma APP, atividade ou hierarquia de visualização. O estilo é uma coleção de atributos que especificam a aparência de uma única *View*. Os estilos são aplicados aos elementos do *layout* através do *style*. Por exemplo, se queremos que o texto de um *TextView* apareça a verde, é então criado um estilo no ficheiro *resource* da seguinte forma (trecho de código 4.2):

```
<style name="GreenText" parent="TextAppearance.AppCompat">
       <item name="android:textColor">#00FF00</item>
```

```
</style>
```

Trecho de código 4.2: Criação do estilo para cor do texto.

Finalmente, acrescenta-se a seguinte linha de código (trecho de código 4.3) ao de *layout* do elemento gráfico *TextView* para que este assuma o estilo pretendido, tendo em atenção que o nome do *style* tem de ser o mesmo que o criado no ficheiro de estilo.

```
<TextView
    style="@style/GreenText"
    ... />
```

Trecho de código 4.3: Aplicação do estilo criado ao elemento pretendido.

Os temas são aplicados no *manifest* da seguinte forma caso seja um estilo geral para toda a aplicação (trecho de código 4.4):

```
<manifest ... >
    <application android:theme="@style/Theme.AppCompat" ... >
    </application>
</manifest>
```

Trecho de código 4.4: Aplicação de temas no *manifest*.

4.2.2 *Developing Android Apps*

Developing Android Apps é o segundo curso da academia *Android*, sendo por isso mais avançado, desenhado para aplicar consecutivamente diferentes conhecimentos. O objetivo deste curso é a criação de uma aplicação denominada *Sunshine*. Esta aplicação tem a ligação a serviços e a listagem da informação recebida num elemento gráfico *recyclerView*. É também neste curso que se aprende um pouco sobre o ciclo de uma aplicação e funcionalidades que correm em *background*.

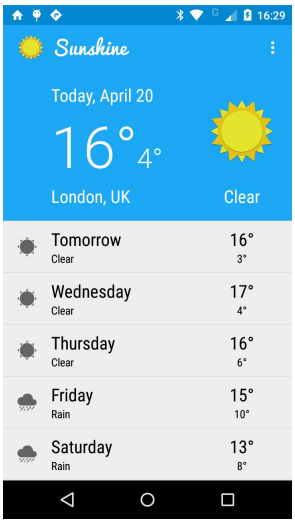


Figura 4.2: *Mockup* da aplicação *Sunshine*.

A figura 4.2 apresenta o aspeto geral da aplicação *Sunshine*, que tem uma ligação aos serviços com o clima e condições meteorológicas. Este serviço envia uma listagem com: o nome do dia da semana, a temperatura mínima e a temperatura máxima e o estado geral do dia. Para cada elemento (dia da semana), é criado um item com as devidas proporções e informações. Posteriormente este elemento é adicionado ao *recyclerView* que os apresenta como uma listagem.

4.2.3 Projeto Final da Academia *Android*

O projeto final teve como objetivo aplicar todos os conhecimentos adquiridos na academia *Android*. O objetivo do projeto era a criação de uma aplicação de listagem de filmes, tendo uma ligação a uma *API*, criando cada elemento e integrando-o num `RecyclerView`.Com o objetivo de dar ao utilizador a possibilidade de descobrir os filmes mais populares em exibição, criar uma lista de favoritos, ver informações sobre o filme e ainda a possibilidade de ver o *trailer* do mesmo. Logo que seja aberta, a aplicação apresenta ao utilizador uma lista de posteres de filmes (figura 4.3). Esta lista pode ser reorganizada conforme popularidade ou classificação através do menu (figura 4.4).

Figura 4.3: Captura de ecrã inicial da aplicação do projeto final da Academia *Android*.

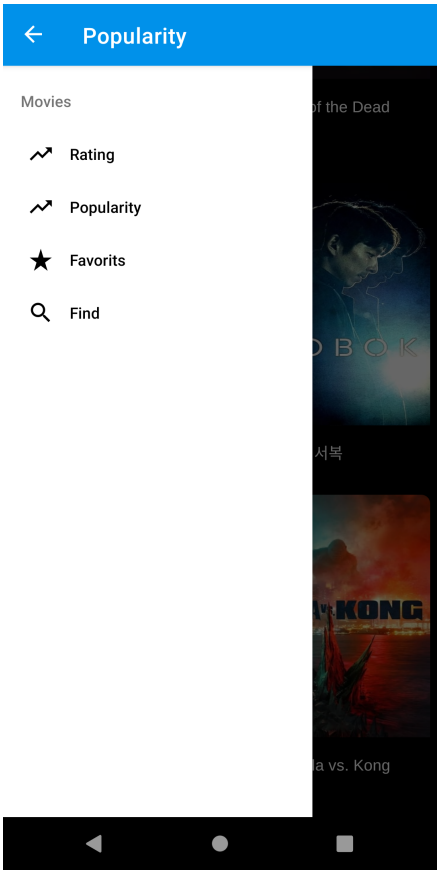


Figura 4.4: Captura de ecrã do menu da aplicação do projeto final da Academia *Android*.

Se o utilizador escolher um poster de entre os exibidos, irá despoletar a abertura da página de informações desse filme (figura 4.5). Esta página contem informações como:

1. Título original;
2. Miniatura da imagem do póster do filme;
3. Uma sinopse;
4. Avaliação/Comentários de outros utilizadores;

5. Data de lançamento.

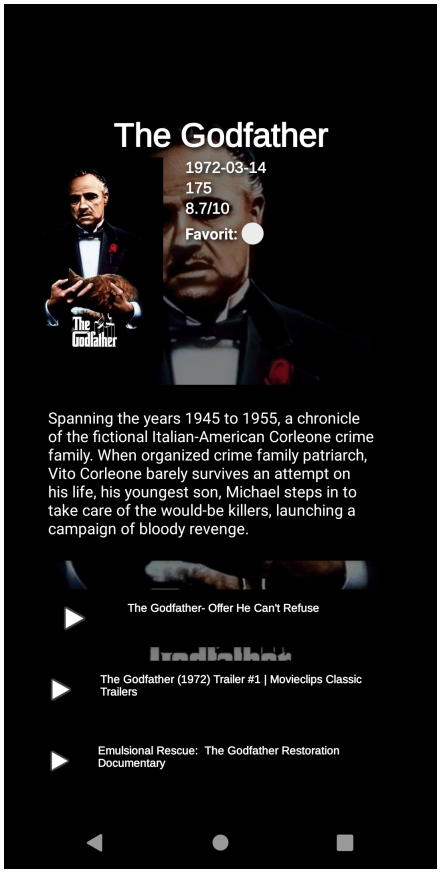


Figura 4.5: Captura de ecrã dos detalhes do filme da aplicação.

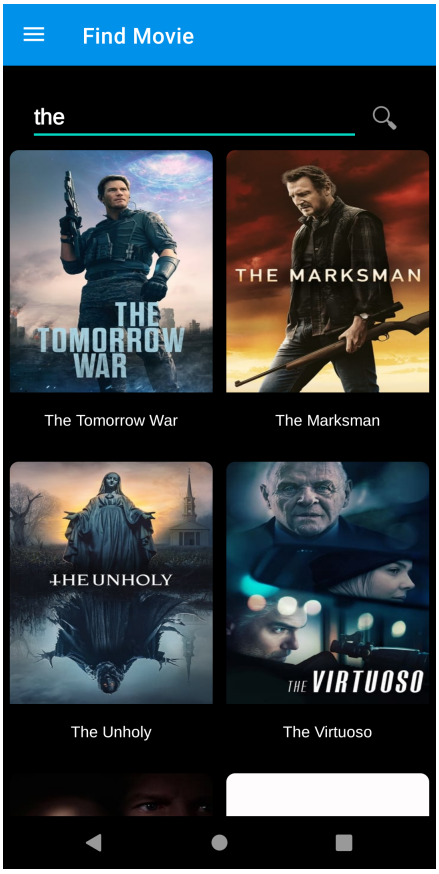


Figura 4.6: Captura de ecrã da pesquisa de filmes da aplicação.

Através da página de detalhes do filme, o utilizador pode adicionar esse filme à lista de favoritos, ver o *trailer* do filme no *YouTube* e ler avaliações e comentários de outros utilizadores sobre o filme. Por fim, o utilizador pode ainda procurar por um filme utilizando a página "*Find Movie*" e escrevendo o nome do filme ou parte desse nome (figura 4.6).

No caso específico, para a lista de posteres dos filmes foi utilizada a biblioteca *Picasso*, que está encarregue do carregamento e armazenamento de imagens. Por ser uma biblioteca externa, é preciso acrescentá-la ao ficheiro de preparação/compilação da plataforma (*Gradle*) através da inserção de uma linha no ficheiro `app/build.gradle` (Trecho de código 4.5):

```
implementation 'com.squareup.picasso:picasso:2.71828'
```

Trecho de código 4.5: Implementação da biblioteca Picasso.

É possível então fazer o *load* da imagem do poster através do seguinte código (Trecho de código 4.6):

```
Picasso.get()
    .load("https://image.tmdb.org/t/p/w342/" + movie.getPosterPath())
    .into(imageMoviePoster);
```

Trecho de código 4.6: *Load* da imagem do póster.

A biblioteca obtém a imagem através de um *Uniform Resource Locator* (URL) e mostra-a numa `ImageView` denominada *imageMoviePoster*.

A `Recycler View` da listagem de filmes é formada pela imagem carregada pela biblioteca Picasso, anteriormente mencionada e o nome do filme recebido através dos serviços da *Application Programming Interface* (API) `themoviedb.org`. Para utilizar esta API, foi necessário criar uma conta no *website* `themoviedb.org` e gerar uma *key* (chave de acesso à API), sendo esta *key* única e pessoal.

Através desta API, é possível então listar e retirar diferentes informações, tais como listagem variadas de filmes, séries e novelas de televisão, listar informações de cada um desses itens ou ainda efetuar a classificação/comentários sobre algum título. Contudo, devido à *key* ser pessoal, é necessário adicionar restrições ao ficheiro *.gitignore* para que esta não seja submetida para o *git*.

4.3 Ambientação ao Projeto M24

No que diz respeito ao *Android*, o projeto M24 está estruturado a partir de fragmentos e tendo em conta a privacidade. Isto é, dependendo do utilizador, diferentes funcionalidades serão expostas em fragmentos (que é uma das unidades lógicas para representar informação gráfica em aplicações *Android*). Existem fragmentos privados para aqueles utilizadores que efetuaram o *login* e fragmentos públicos para utilizadores sem o *login* feito.

De uma forma abstrata, pode-se dizer que o projeto é assim composto por diferentes fragmentos, *layouts*, *adapters*, serviços (*response*, *tasks* e *requests*), ficheiros de estilo e *strings* e por fim pelo *manifest*. Contando também com diferentes *Build variants*, estas representam diferentes resultados possíveis de um processo de *build* e agrupam um conjunto de opções de construção, livremente definidas pelo programador, permitindo assim que as aplicações tenham como base a mesma estrutura. Dentro destas variantes, existem *builds* para clientes particulares e outras para empresas. Existem 12 *Build variants*, seis para particulares e seis para empresas, e cada uma destas têm variantes para desenvolvimento, produção e qualidade (podem ser versões *debug* ou *realese*).

4.3.1 Tarefa Um

Como primeira tarefa de ambientação foi pedida a criação de um item no menu da aplicação, com o nome *"Bolsa Resumo"*. Esse item devia estar imediatamente abaixo do item do menu designado por *"Bolsa"*. O item de menu deve redirecionar para um *fragment* que, assim que lançado, chame o serviço `publicMG/FinancialMarketTransaction` e apresente

no ecrã *"Número de Índices: <número total de índices devolvidos pela resposta desse serviço>"*.

4.3.2 Tarefa Dois

Nesta segunda tarefa foi necessário criar um item de menu denominado de *"Cons. Posição Integrada"*, imediatamente abaixo do item *"Posição Integrada"*. Este item redireciona o utilizador para uma vista com uma estrutura semelhante à da *Consulta de Contas à Ordem* (figura 4.7) com a seguinte legenda:

1. Seletor de contas que foi populado com a lista dos vários tipos de ativos;
2. Mantida apenas a *tab* denominada *"Contas"*;
3. Listagem de contas associadas ao tipo de ativo selecionado no seletor de contas, com os dados *nome da Conta*, *saldo* e *moeda*.

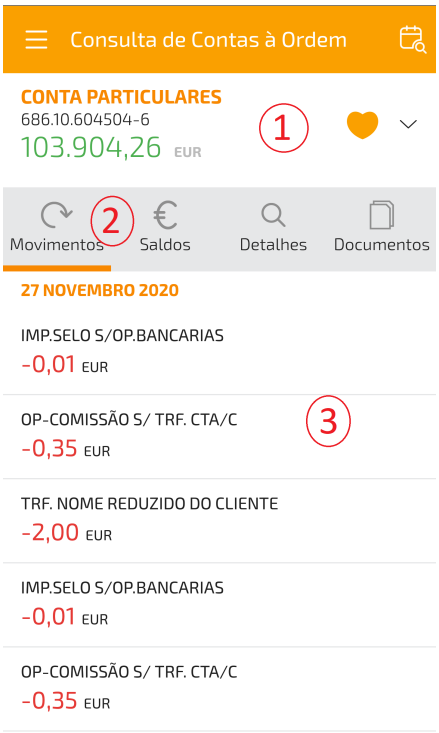


Figura 4.7: Captura de ecrã da Consulta de Contas.

Todos os dados que são apresentados nesta nova vista são fornecidos pelos serviços que atualmente fornecem a funcionalidade de *"Posição Integrada"*.

4.4 Planificação e Implementação de Testes

No seguimento do estudo dos testes automatizados (anexo C) e da ambientação ao projeto e aplicação *Android*, foram criados quatro fluxos de testes automatizados:

- Consulta de Consentimentos;
- Cancelamento de Consentimentos;
- Criação de Consentimentos;
- Pesquisa de Consentimentos.

Foram criados casos de teste para cada um deles, uma vez que cada um dos fluxos tinha diferentes itens de teste e ecrãs. Os casos são descritos nas subsecções seguintes.

4.4.1 Consulta de Consentimentos

O primeiro conjunto de casos de teste visam testar o ecrã apresentado na figura 4.8. Este ecrã é referente à consulta de consentimentos. É nele que podemos obter a listagem de consentimentos que o utilizador tem. A partir deste podem ser observados os detalhes de cada um dos consentimentos, cancelar um consentimento ou ver as execuções dos consentimentos ativos.
É ainda neste ecrã que temos acesso à pesquisa e à filtragem de consentimentos através do ícone de pesquisa canto superior direito do ecrã. Para testar este ecrã e as suas diferentes vertentes, como também o fluxo da consulta de um consentimento, foram criados os seguintes casos de teste (*Happy flow* é a designação dada ao fluxo direto em que tudo deve correr bem):

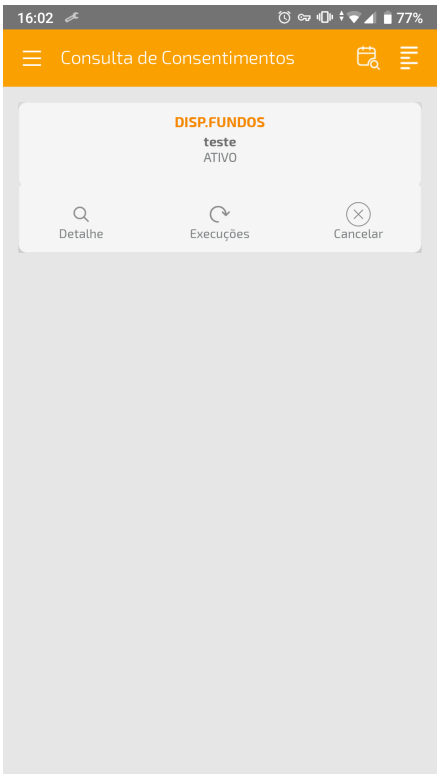


Figura 4.8: Captura de ecrã da aplicação no fragmento de consulta de consentimentos.

***Happy flow* com Detalhes**

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Consentimentos`;
3. Verificar se título da `toolbar = Consulta de Consentimentos`;
4. Verificar que o ecrã apresenta resultados;
5. Fazer *click* no botão `Detalhe` do primeiro elemento;
6. Verificar que ecrã de detalhes aparece, sendo que o ecrã de detalhe está representado na figura 4.9.

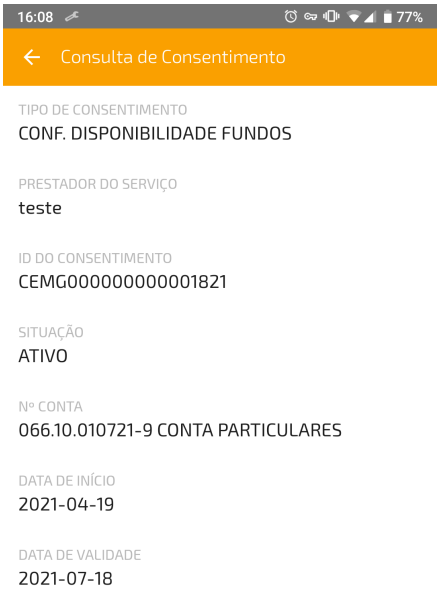


Figura 4.9: Captura de ecrã da consulta dos detalhes de consentimentos.

***Happy flow* com Execução**

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Consentimentos`;
3. Verificar se título da `toolbar = Consulta de Consentimentos`;
4. Verificar que o ecrã apresenta resultados;
5. Simular *click* no botão `Execução` do primeiro elemento;
6. Verificar que ecrã de execução aparece, sendo que o ecrã que deve aparecer está representado na figura 4.10.

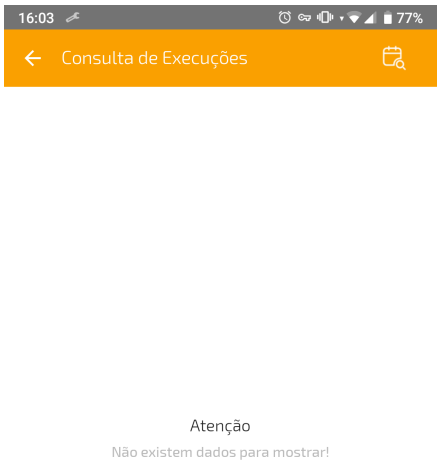


Figura 4.10: Captura de ecrã da aplicação no fragmento de consulta das execuções de consentimentos.

4.4.2 Cancelamento de Consentimentos

O próximo fluxo descrito visa a testar as interações com o ecrã de cancelamento de consentimentos e o respetivo fluxo para cancelar um consentimento apresentado na figura 4.11. Para isso, foram criados os dois casos de teste descritos a seguir.

***Happy flow*:**

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Consentimentos`;
3. Verificar se título da `toolbar = Consulta de Consentimentos`;
4. Verificar que o ecrã apresenta resultados;
5. Simular *click* no botão `Cancelar` do primeiro elemento;
6. Verificar que o ecrã de cancelamento aparece com os dados do elemento clicado (figura 4.11);
7. Simular *click* no botão `Confirmar` no ecrã de cancelamento;
8. Inserir *SMS Code*;
9. Fazer *click* no botão `Confirmar`;

10. Esperar que ecrã de confirmação de cancelamento bem-sucedido apareça e confirmar que apresenta os dados do elemento clicado.

/c *SMS Code* inválido:

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu *`Open Banking`* > `Consentimentos`;
3. Verificar título da *`toolbar`* = `Consulta de Consentimentos`;
4. Verificar que o ecrã apresenta resultados;
5. Fazer *click* no botão `Cancelar` do primeiro elemento;
6. Verificar que ecrã de cancelamento aparece com os dados do elemento clicado;
7. Simular *Click* no botão `Confirmar` no ecrã de cancelamento;
8. Fazer *click* mp botão `Confirmar`;
9. Verificar que aparece nota de `erro = Valor inválido`.

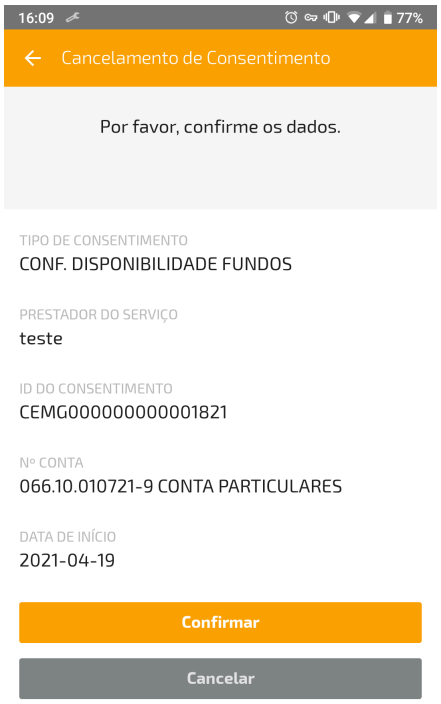


Figura 4.11: Captura de ecrã da aplicação no fragmento de cancelamento de consentimentos.

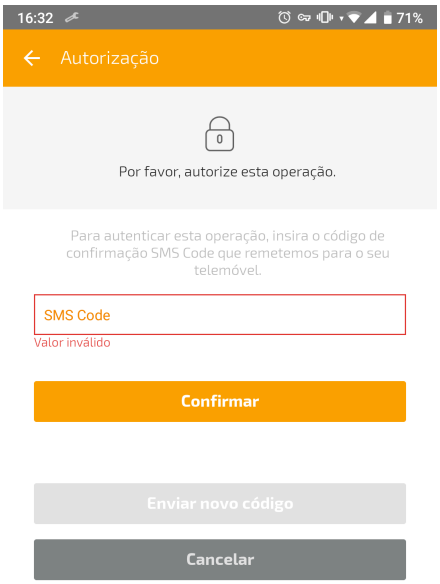


Figura 4.12: Captura de ecrã da aplicação no fragmento do *SMS Code* dos consentimentos.

4.4.3 Criação de Consentimentos

Este fluxo visa testar todas as componentes que levam à criação de um consentimento, este remetendo para o ecrã apresentado na figura 4.13. Para isso foram criados quatro

casos de teste, descritos a seguir.

Happy flow

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Conf. Disponibilidade Fundos`;
3. Verificar se título da `toolbar = Confirmação da Disp. De Fundos`;
4. Preencher campo Prestador do serviço com `<prestador do serviço>`;
5. Simular *click switch* no campo "Nº DO CARTÃO";
6. Confirmar que campo de "Nº DO CARTÃO" apareceu;
7. Preencher campo "Nº DO CARTÃO" com `<nº do cartão>`;
8. Simular **click** no botão `Continuar`;
9. Verificar que ecrã de confirmação aparece com prestador de serviço = `< prestador do serviço>` e Nº DO CARTÃO = `<nº do cartão>`;

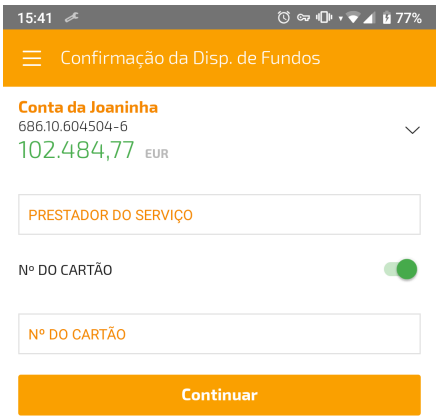


Figura 4.13: Captura de ecrã da aplicação no fragmento de criação de consentimentos.

10. Simular um **click** no botão `Confirmar`;
11. Esperar ecrã de confirmação de operação concluída com sucesso onde prestador de serviço = `< prestador do serviço>` e Nº DO CARTÃO = `<nº do cartão>`;
12. Ir para menu `Open Banking > Consentimentos`;
13. Confirmar que existe elemento com o mesmo nome de `<prestador de serviços>` e nº do cartão com o `nº do cartão` fornecido.

/c Nº do cartão desativado:

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Conf. Disponibilidade Fundos`;
3. Verificar título da `toolbar = Confirmação da Disp. De Fundos`;
4. Preencher campo Prestador do serviço com `<prestador do serviço>`;
5. Fazer *click* no botão `Continuar`;
6. Verificar que ecrã de confirmação aparece com prestador de serviço = `< prestador do serviço>`;

7. Fazer *click* no botão `Confirmar`;
8. Esperar ecrã de confirmação de operação concluída com sucesso onde prestador de serviço = `< prestador do serviço>`;
9. Ir para menu `Open Banking > Consentimentos`;
10. Confirmar que existe elemento com o mesmo nome de `<prestador de serviços` fornecido anteriormente.

/c Prestador de serviços inválido:

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Conf. Disponibilidade Fundos`;
3. Verificar título da `toolbar = Confirmação da Disp. De Fundos`;
4. Simular *click* no botão `Continuar`;
5. Verificar que aparece nota de `erro = "O campo Prestador do Serviço deve ser preenchido"`.

/c Nº do cartão inválido:

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Conf. Disponibilidade Fundos`;
3. Verificar se título da `toolbar = Confirmação da Disp. De Fundos`;
4. Preencher campo Prestador do serviço com `<prestador do serviço>`;
5. Simular um *click switch* "Nº DO CARTÃO";
6. Confirmar que novo campo de "Nº DO CARTÃO" apareceu;
7. Preencher campo "Nº DO CARTÃO" com `<nº do cartão inferior a 6>`;
8. Fazer *click* no botão `Continuar`;
9. Verificar que aparece `aviso = "O tamanho do campo Número do Cartão deverá estar compreendido entre 6 e 19.;`
10. Preencher campo "Nº DO CARTÃO" com `<nº do cartão superior a 19>`;
11. Verificar que aparece um número de cartão com apenas os primeiros 19 dígitos inseridos.

4.4.4 Pesquisa de Consentimentos

Por último, o fluxo de pesquisa de consentimentos. Este fluxo visa testar todos os itens e componentes do ecrã representado pelas figuras 4.14 e 4.15. Para os testar, foram criados os seguintes os caos de teste descritos em baixo.

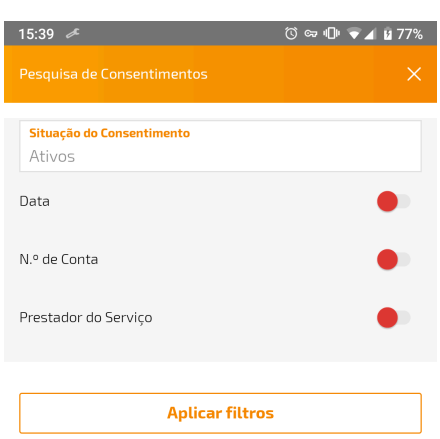


Figura 4.14: Captura de ecrã da aplicação no fragmento de pesquisa de consentimentos.

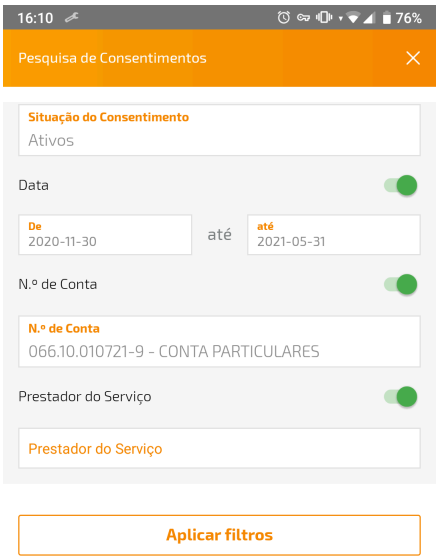


Figura 4.15: Captura de ecrã da aplicação no fragmento de pesquisa de consentimentos com todos os *switch* ligados.

***Happy flow*:**

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Consentimentos`;
3. Verificar se título da *`toolbar`* = `Consulta de Consentimentos`;
4. Simular *click* botão de pesquisa na *toolbar*;
5. Verificar que ecrã `Pesquisa de Consentimentos` apareceu;
6. Selecionar Situação do Consentimento:
    - Fazer um *click* no campo `Situação do Consentimento`;
    - Verificar que ecrã de seleção de situação do consentimento apareceu;
    - Fazer um *click* no elemento da `lista = <situação do consentimento`;
    - Fazer um *click* no botão `Selecione`;
    - Verificar que elemento clicado da lista aparece no campo `Situação do Consentimento`.
7. *Switch "on"* Data;
8. Verificar que Campos de data apareceram;

9. Preencher Campos da data:
    - Simular *click* no campo de seleção de data inicial;
    - Verificar que ecrã de escolha de data apareceu;
    - Selecionar `<data inicial>`;
    - Emitir um *click* no botão `OK`;
    - Fazer um *click* no campo de seleção de data final;
    - Verificar que ecrã de escolha de data apareceu;
    - Selecionar data após a `data inicial = <data final>`;
    - Fazer *click* em `OK`.
10. *Switch "on"* Nº de Conta;
11. Verificar que campo "Nº de conta" apareceu;
12. Selecionar conta:
    - Emitir um *click* no campo `Nº de Conta`;
    - Verificar que ecrã de seleção de conta apareceu;
    - Simular um *click* em `um elemento da lista = <nº de conta>`;
    - Fazer *click* no botão `Selecione`;
    - Verificar que elemento clicado da lista aparece no campo nº de conta;
13. *Switch "on"* Prestador do Serviço;
14. Verificar que o campo `Prestador do Serviço` apareceu;
15. Preencher campo com `<prestador do serviço>`;
16. Simular um *Click* no botão `Aplicar filtros`;
17. Confirmar que aparece nota "Consentimentos `<situação do consentimento>`, para a conta à ordem `<nº de conta>`, de `<prestador do serviço>`, entre `<data inicial>` a `<data final>`.

/c Datas inválidas:

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Consentimentos`;
3. Verificar se título da `toolbar = Consulta de Consentimentos`;
4. Simular *click* no botão de pesquisa na *toolbar*;
5. Verificar que ecrã "Pesquisa de Consentimentos" apareceu;

6. *Switch "on"* Data;
7. Verificar que Campos de data apareceram;
8. Preencher Campos da data:
    - Emitir um *click* no campo de seleção de data inicial;
    - Verificar que ecrã de escolha de data apareceu;
    - Selecionar `<data inicial>`;
    - Fazer um *click* em `OK`;
    - Emitir um *click* no campo de seleção de data final;
    - Verificar que ecrã de escolha de data apareceu;
    - Selecionar data após a `data inicial = <data final>`;
    - Emitir um *Click* em `OK`.
9. Finalmente, simular um *Click* no botão `"Aplicar filtros"`;
10. Verificar que aparece nota de `erro = "A data de início deverá ser inferior à data de fim"`.

/c Outras situações do consentimento:

1. Fluxo de *login* com cliente 3 656 020;
2. Ir para menu `Open Banking > Consentimentos`;
3. Verificar título da `toolbar = Consulta de Consentimentos`;
4. Emitir um *click* no botão de pesquisa na *toolbar*;
5. Emitir um *click* em `Situação do consentimento`;
6. Verificar que janela de escolha apareceu;
7. Simular um *click* no elemento `Todos`;
8. Fazer um *click* no botão `Selecione`;
9. Verificar que *switch* das datas já não está disponível;
10. Fazer *click* em `Situação do consentimento`;
11. Verificar que janela de escolha apareceu;
12. Simular um *click* no elemento `Cancelados`;
13. Fazer *click* no botão `Selecione`;
14. Verificar que *switch* das datas já não está disponível;
15. Fazer *Click* em `exit` para fechar a janela;

16. Confirmar que não aparece nota nenhuma de pesquisa.

4.4.5 Implementação e Execução dos Testes

Para a execução dos *scripts* de teste foi utilizada a ferramenta *Appium*, referida e estudada na secção 2.6.3. Esta ferramenta permite detetar os elementos presentes numa aplicação, atividade ou fragmento apresentado, através da execução da aplicação.

Feita a ligação ao dispositivo, a aplicação é iniciada e, a partir desta, é possível navegar até ao fragmento ou atividade em que identificamos o elemento. Identificando esse elemento, podemos fazer *refresh* no *Appium* apresentando então todos os elementos visíveis na aplicação, como é possível ver como exemplo na figura 4.16.

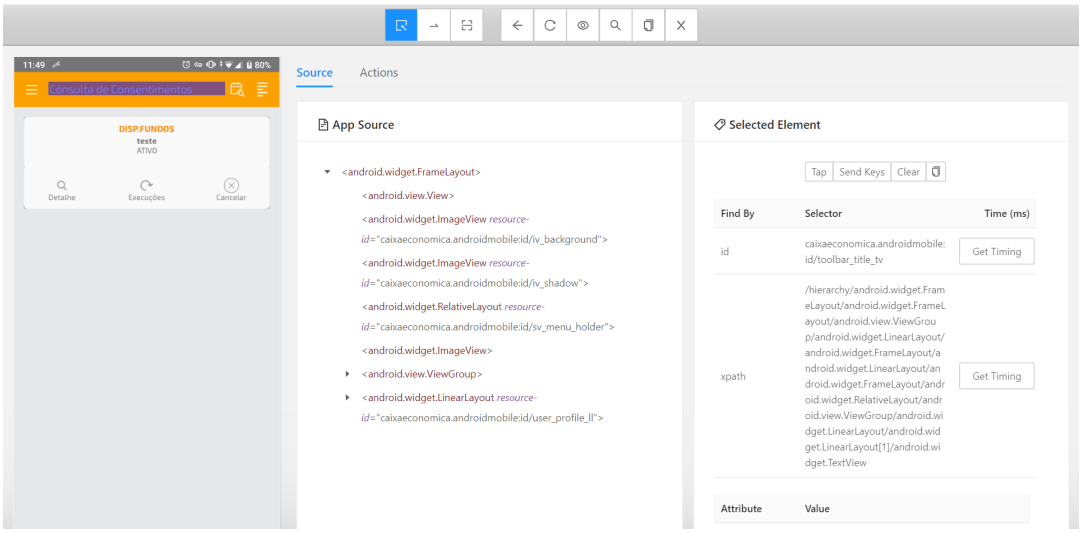


Figura 4.16: Captura de ecrã da aplicação *Appium* com exemplo de deteção de elemento.

Na figura 4.16 podemos ver como é possível detetar o *Identifier* (ID) do elemento de texto da *toolbar* (este elemento é utilizado em todos os casos de teste como forma de confirmar que o devido texto esta correto e que o fragmento e atividade que está a ser testada é a correta). A deteção do ID neste caso foi realizado através da seleção do item que queremos identificar, seguidamente pela informação do lado direito do ecrã apresentado com o *"Selected Element"* e pelo *find by* > *id*.

Em relação à criação dos *scripts* de testes, foi criada uma *suite* de testes que, no final, lista os diferentes fluxos criados, representada pelo código seguinte (Trecho de código 4.7):

```
<!DOCTYPE suite SYSTEM "https://testng.org/testng-1.0.dtd" >
<suite name="Suite">
    <test name="OpenBankingTestsSuite">
        <classes>
            <class name="tests.OpenBanking.OpenBankingQuery"/>
            <class name="tests.OpenBanking.OpenBankingCreation"/>
            <class name="tests.OpenBanking.OpenBankingSearch"/>
            <class name="tests.OpenBanking.OpenBankingCancel"/>
        </classes>
```

```
    </test> <!-- Test -->
</suite> <!-- Suite -->
```

Trecho de código 4.7: *Suite* de testes criada.

Esta *Suite* é utilizada para executar os testes, através da configuração `run` da ferramenta *Intellij* descrita no subcapítuloEngenharia Informática 2.6.3.1. Podemos ver como essa configuração foi efetuada na figura 4.17

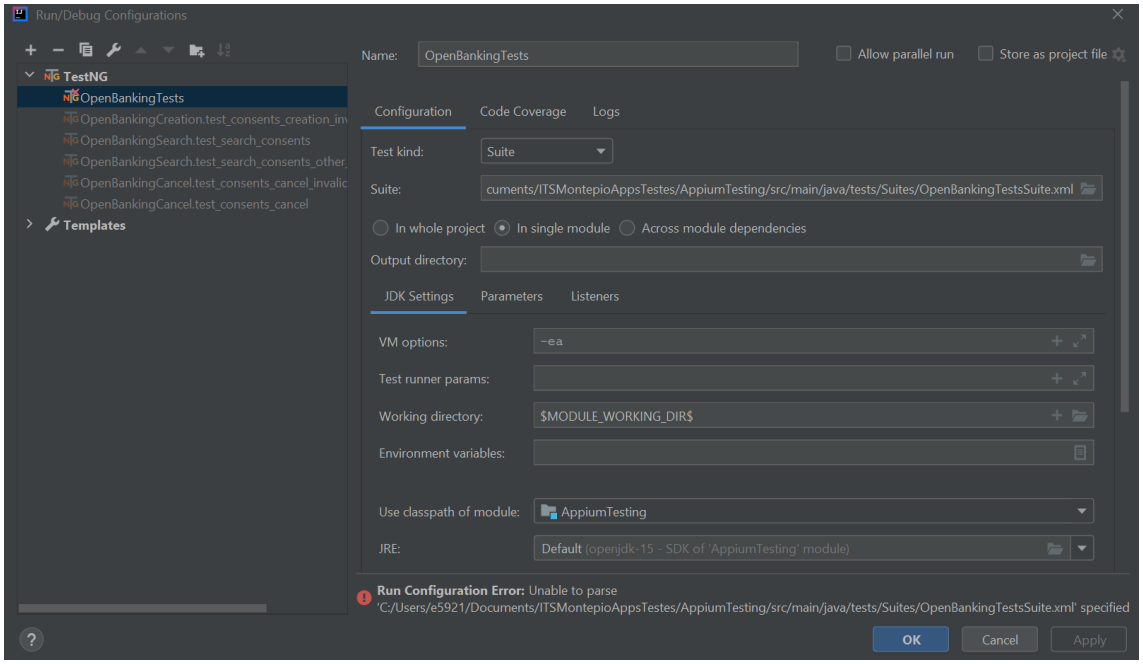


Figura 4.17: Captura de ecrã da janela *"Run/Debug Configurations"* da ferramenta *IntelliJ*.

Dando um nome à configuração, é possível definir qual o tipo de teste (*Test kind*) que, para o caso, é *"Suite"*, e definir a *directory* desta *Suite*. É importante conferir a pasta dos módulos usada. Neste caso foi usado o módulo *"AppiumTesting"* com a SDK predefinida.

Após definir a *Suite*, é necessário criar as classes que vão ser executadas nela. Estas classes vão apresentar os fluxos criados anteriormente para cada um dos casos de teste.

A classe de consulta (trecho de código 4.8) tem dois casos de teste: o *happy flow* c/ detalhes criado a partir da função `test_consents_details()` e o *happy flow c/ execução*, invocado a partir da função `text_consents_execution()`. Cada um dos casos de teste tem um `MGTestIdentifier` de forma a identificar qual o teste que está a correr e uma descrição que detalha o que o teste faz.

```
public class OpenBankingQuery extends BaseTestClass {

    private final String groupName = "OpenBanking";

    @Test(groups = {groupName})
    @MGTestIdentifier(name = "Consentimentos - Happy Flow c/Detalhes",
    description = "Fluxo normal de consulta de Consentimentos c/Detalhes")
    public void test_consents_details() {
```

```
        login_happyFlow();

        openBankingQueryPage.openMenu();
        openBankingQueryPage.goToConsentsFromMenu();

        openBankingQueryPage.validateToolbarTitle(getConsts().TOOLBAR_TITLE_CONSENTS);
        openBankingQueryPage.validateResultsAreVisible();

        openBankingQueryPage.clickDetailsButton();
        openBankingQueryPage.confirmPageDetailsIsVisible();
    }

    @Test(groups = {groupName})
    @MGTestIdentifier(name = "Consentimentos - Happy Flow c/Execução",
    description = "Fluxo normal de consulta de Consentimentos c/Execução")
    public void test_consents_execution() {
        login_happyFlow();

        openBankingQueryPage.openMenu();
        openBankingQueryPage.goToConsentsFromMenu();

        openBankingQueryPage.validateToolbarTitle(getConsts().TOOLBAR_TITLE_CONSENTS);
        openBankingQueryPage.validateResultsAreVisible();

        openBankingQueryPage.clickExecutionsButton();
        openBankingQueryPage.confirmPageExecutionsIsVisible
        (getConsts().TOOLBAR_TITLE_CONSENTS_EXECUTION);
    }
}
```

Trecho de código 4.8: Classe `OpenBankingQuery`.

No código anterior é utilizada uma classe denominada `openBankingQueryPage`. Esta classe é utilizada para listar e criar todas as funções executadas no fluxo de consulta dos consentimentos. A classe deve estar listada também na classe `BaseTestClass` para ser feito um *setup* inicial e seja iniciado o *controller Appium*.

Posto isto, cada um dos casos de teste listados anteriormente tem uma classe deste tipo com os diferentes fluxos pertencentes a esse caso. Dentro desses fluxos são chamadas as diferentes funções listadas na *Page* referente a esse caso. Seguidamente, é apresentada a `OpenBankingQueryPage` (Trecho de código 4.9) como forma de exemplo de como é estruturada.

```
public interface OpenBankingQueryPage {
    void openMenu();
    void goToConsentsFromMenu();
    void validateToolbarTitle(String title);
    void validateResultsAreVisible();
    void clickDetailsButton();
    void confirmPageDetailsIsVisible();
    void clickExecutionsButton();
    void confirmPageExecutionsIsVisible(String executionTitle);
}
```

Trecho de código 4.9: Interface `OpenBankingQueryPage`.

Estes fluxos devem ser iguais, tanto para iOS como para *Android*. Dessa forma apenas a programação de cada função é que difere. Por isso, é criada uma classe denominada `OpenBankingQueryPageAndroid` referente ao *Android*, que implementa a classe anterior (`OpenBankingQueryPage`).

Começando pela função de abrir o menu `openMenu()` (Trecho de código 4.10), esta é genérica a todos os fluxos, sendo deste modo utilizada uma função criada no `AndroidHelper`, denominada por `openMainMenu(driver)` que recebe o *driver AppiumDriver*.

```
public void openMenu() {
    AndroidHelper.openMainMenu(driver);
}
```

Trecho de código 4.10: Função `openMenu`.

Esta função (ver trecho de código 4.11) deteta o ícone menu através do ID (`MobileBy.ByAccessibilityId("Navegar para cima")`) e executa o *click*. É importante referir que existe um tempo máximo para esta ação ser executada de 60 segundos. Tal limitação faz com que, se a ação não for executada, o caso de teste dê erro neste passo. O tempo de espera é definido pelo *driver* recorrendo ao método `WebDriverWait()`.

```
public static void openMainMenu(MobileDriver driver) {
    MobileElement menuIcon = (MobileElement) (new WebDriverWait(driver, 60))
            .until(ExpectedConditions.elementToBeClickable(new
            MobileBy.ByAccessibilityId("Navegar para cima")));

    menuIcon.click();
}
```

Trecho de código 4.11: Função `openMainMenu`.

Depois do aparato de teste estar posicionado no menu, é necessário encontrar o item menu com o devido título (neste caso *Open Banking*) e sub-menu *consentimentos*. O item é detetado e clicado através da função `getElementFromActionInMainMenuSubMenu()`, invocada dentro da função de teste, como mostra o Trecho de código 4.15:

```
public void goToConsentsFromMenu() {
    AndroidHelper.getElementFromActionInMainMenuSubMenu(driver,
            Configs.getConsts().MAIN_MENU_ITEM_OPEN_BANKING_TITLE,
            Configs.getConsts().MAIN_MENU_SUBITEM_CONSENTS).click();
}
```

Trecho de código 4.12: Função `goToConsentsFromMenu`.

A função `getElementFromActionInMainMenuSubMenu()` pertence à classe `AndroidHelper` por ser uma função que é executada em diferentes classes e por ser genérica. No código seguinte podemos ver uma técnica de deteção do elemento diferente da anterior. Desta vez, o elemento é detetado através do *xpath* e do ID (Trecho de código 4.13).

```
public static MobileElement getElementFromActionInMainMenuSubMenu(MobileDriver driver,
            String actionTitle, String subActionTitle) {
    MobileElement menuItem = getElementFromActionInMainMenu(driver, actionTitle);
    menuItem.click();

    String resourceID = "caixaeconomica.androidmobile:
    id/navigation_drawer_child_label_tv";
    MobileElement itemChild = (MobileElement) (new WebDriverWait(driver, 60))
            .until(ExpectedConditions.presenceOfElementLocated(
            By.xpath("//*[contains(@text, \"" + subActionTitle + "\")
            and @resource-id=\"" + resourceID + "\"]")));
    return itemChild;
}
```

Trecho de código 4.13: Deteção de elemento através do *xpath*.

A função anterior devolve o elemento `itemChild`, permitindo que este possa ser clicado e utilizado pela função `goToConsentsFromMenu()` (Trecho de código 4.15). Como forma de validar o título apresentado, é executado o método `validateToolbarTitle(String)`.

```
public void validateToolbarTitle(String title) {
    AndroidHelper.assertToolbarTitle(driver, title, true, 10);
}
```

Trecho de código 4.14: Função `goToConsentsFromMenu`.

A função de validação do título receber o *driver*, uma *string* com o título específico, um *boolean* que determina se o título dado deve ser ou não igual ao apresentado pela aplicação e por fim o tempo de *timeOut*, remetendo finalmente para a função do `AndroidHelper` denominada `assertToolbarTitle()` (Trecho de código 4.15). Esta função começa por detetar o elemento do título com o ID do mesmo, indo de encontro com o ID detetado pelo *Appium* no início da execução. Após encontrar o elemento, é feito o *assert*, isto é, se o título dado é ou não igual ao encontrado no mesmo ID.

```
public static void assertToolbarTitle(MobileDriver driver, String toolbarTitle, boolean s
    MobileElement element = (MobileElement) (new WebDriverWait(driver, timeoutInSeconds))
            .until(ExpectedConditions.elementToBeClickable(new
            By.ById("caixaeconomica.androidmobile:id/toolbar_title_tv")));

    Assert.assertTrue(element.isDisplayed());
    if (shouldMatch)
        Assert.assertTrue(element.getText().equals(toolbarTitle));
    else
        Assert.assertFalse(element.getText().equals(toolbarTitle));
}
```

Trecho de código 4.15: Função `assertToolbarTitle`.

Para validar se os resultados estão visíveis é executada a função `validateResultsAreVisible()` (Trecho de código 4.16) em que a mesma lógica é seguido, i.e., procurando detetar se pelo menos um elemento é visível. Nesta função são

também levantados os elementos que identificam o primeiro elemento, sendo este o nome do elemento e a situação do elemento.

```
public void validateResultsAreVisible() {
    MobileElement element = (MobileElement) (new WebDriverWait(driver, 10))
            .until(ExpectedConditions.presenceOfElementLocated(
            By.id("caixaeconomica.androidmobile:id/layout_account_info_ll")));
    Assert.assertTrue(element.isDisplayed());

    List<MobileElement> amountElementsFoundNames = driver.findElements(
            AndroidHelper.getElementsLocatorByResourceID(
            "caixaeconomica.androidmobile:id/tpp_tv"));

    firstElementName = amountElementsFoundNames.get(0);
    firstElementNameString = firstElementName.getText();

    List<MobileElement> amountElementsFoundSituations = driver.findElements(
            AndroidHelper.getElementsLocatorByResourceID(
            "caixaeconomica.androidmobile:id/consent_tpp_tv"));

    firstElementSituation = amountElementsFoundSituations.get(0);
    firstElementSituationString = firstElementSituation.getText();
}
```

Trecho de código 4.16: Função `validateResultsAreVisible`.

Todas as funções restantes funcionam nas seguindo o mesmo raciocínio e lógica: detetar o elemento com o devido nome ou ID para confirmar que este está visível ou para nele clicar.

Ao serem executados, estes testes irão criar um relatório com os resultados para demonstrar se o teste correu bem ou se não correu bem, permitindo determinar em que ponto ocorreu um impedimento. Estes relatórios têm o aspeto representado pela imagem 4.18.

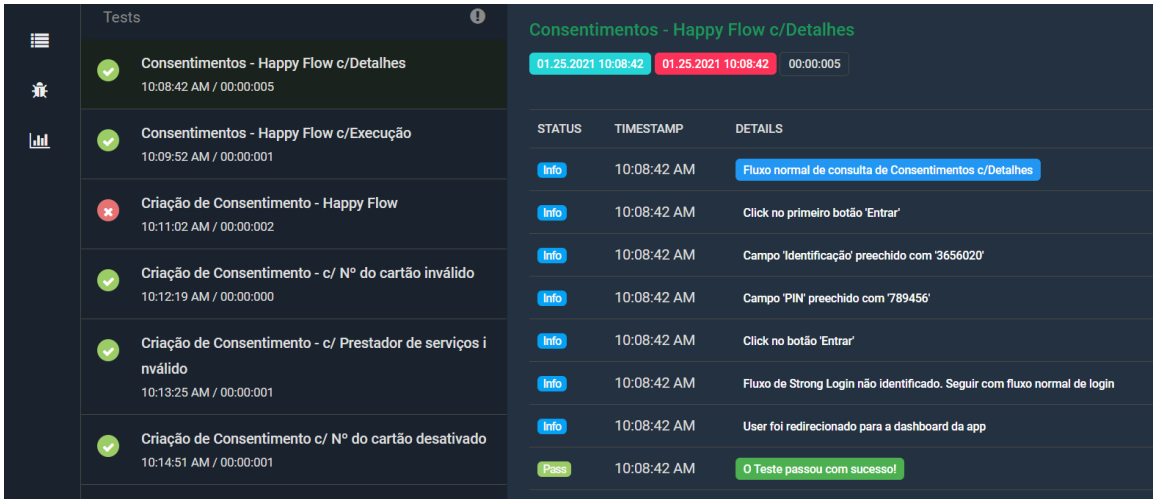


Figura 4.18: Captura de ecrã do reporte exemplo.

Este ficheiro permite ver quanto tempo o teste demorou e todos os passos executados até o mesmo. Do lado esquerdo são listados os testes ocorridos e do lado direito é listado o

estado do teste num determinado tempo do teste. No caso do teste não correr bem, terá o erro descrito no relatório e com o *status* a *Fail*, como pode ser visto na figura 4.19.

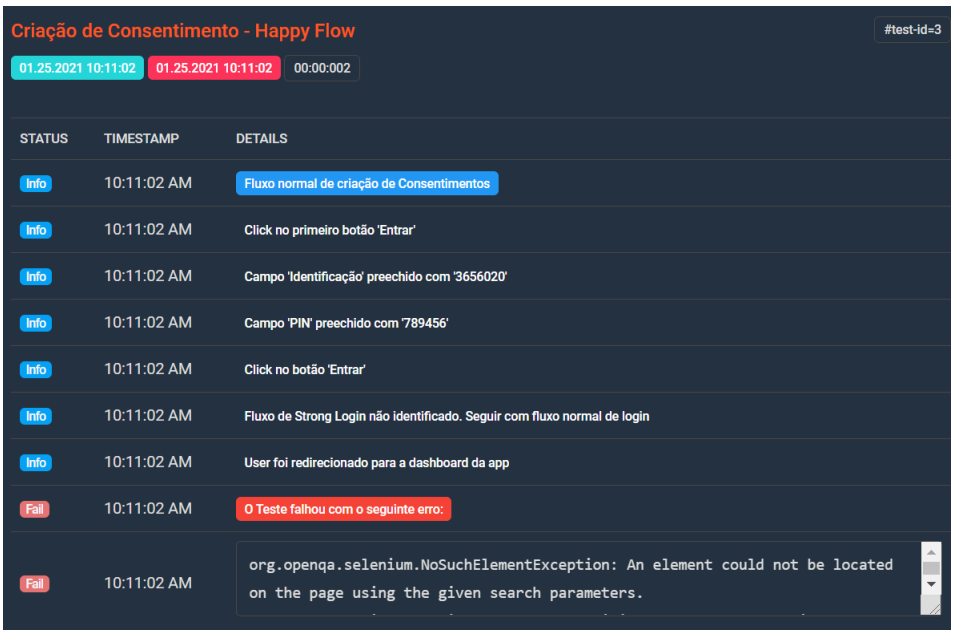


Figura 4.19: Captura de ecrã do realatório exemplo com teste falhado.

No caso de haver um salto num passo do teste, o relatório terá um aspeto semelhante ao da figura 4.20.

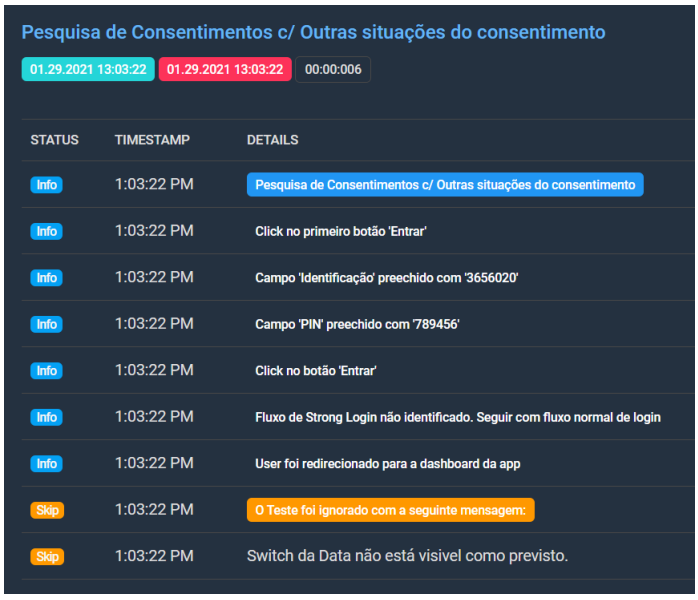


Figura 4.20: Captura de ecrã do relatório exemplo com teste saltado.

A informação do salto é guardada, como também é identificado qual o passo do teste no qual foi dado o *skip* ao teste. No caso do teste da figura 4.20, devido ao *switch* da data não estar visível como previsto, o teste foi saltado. Nestes relatórios, é ainda possível obter os gráficos e informações representados na figura 4.21. É relatada a data e hora do início

do teste, do fim do teste, quantos testes passaram e quantos falharam, como também um gráfico geral de percentagem de falha e passado. Numa tabela adicional, é apresentado o ambiente em que o teste foi corrido (neste caso no ambiente de qualidade) e qual o utilizador utilizado para criar este reporte.

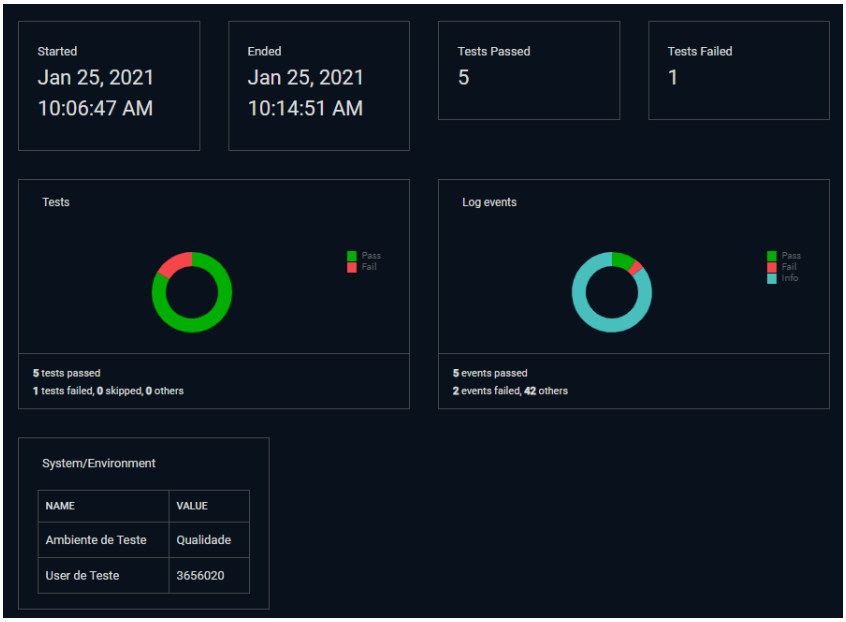


Figura 4.21: Captura de ecrã do relatório com os gráficos e informações gerais.

4.5 Menus Dinâmicos

O menu é uma lista de elementos numa gaveta de navegação (*navigation drawer*). Cada um destes elementos tem um *ID* e um texto que os distingue. Até ao momento, o menu era apresentado de forma estática na aplicação M24, mas com os menus dinâmicos é possível desativar certas funcionalidades, nomeadamente escondendo o item do menu. Todos os atributos e parâmetros são atribuídos no *backend*, o que torna estes menus realmente dinâmicos.

4.5.1 Serviços

A ideia desta parte do estágio era então criar menus dinâmicos que iriam beber de um serviço na *web*. Foi então necessário criar um serviço de armazenamento dos menus e das suas posições na aplicação móvel, conseguido através do seguinte Trecho de código 4.17:

```
public interface MenuServices {
    @POST(CommunicationCenter.ServiceDynamicMenusStep0MG)
    Call<ResponseDynamicMenusStep0> getDynamicMenu();
}
```

Trecho de código 4.17: Serviços de armazenamento dos menus.

Com esta funcionalidade é possível que a instituição financeira tenha uma melhor organização e ordem sobre o que cada utilizador e tipo de conta tem acesso. Isto é, certos utilizadores (tipos de conta) podem ter acesso a itens do menu diferentes de outros utilizadores, dando a liberdade ao banco de disponibilizar diferentes funcionalidades para diferentes utilizadores e necessidades.

4.5.2 Obter Menus Dinâmicos

A função `getDynamicMenus()` faz uma chamada ao serviço anterior e recebe uma lista de itens de menu através de um *Listener*. Se o *Listener* obtiver *response* é executado o *stepo*. A classe no trecho de código 4.18 recebe o resultado da *response* dos serviços, obtém o menu através dos itens de menu dinâmico e configura o menu com uma função denominada `setMenu(...)` que recebe um `ArrayList<DynamicMenuItem>`, que é o que é devolvido pelos serviços.

```
ResponseDynamicMenusStep0(ErrorObj error, List<ApiMessages> messages,
Boolean refreshAccounts, MetaInfoObj metaInfo, String status,
ResponseDynamicMenusStep0Result result)
```

Trecho de código 4.18: Classe `ResponseDynamicMenusStep0`.

No caso do serviço não devolver nenhuma resposta (no caso de erro), é considerado que o *stepo* falhou e é consecutivamente enviada uma mensagem de erro.

4.5.3 Exemplo Prático

Como podemos ver na figura 4.23, o ecrã mostrado tem um menu em que a opção *"Posição Integrada"* se encontra desativada. Neste caso, se o utilizador clicar neste item de menu, o aviso representado na figura 4.22 é mostrado, demonstrando então que a opção de menu está desabilitada.

Figura 4.22: Aviso de opção desabilitada.

No caso dos itens de menu que se encontram a branco, o utilizador pode continuar a utilizá-los normalmente. Por outras palavras, se um item do menu for clicado, abrirá o respetivo fragmento.

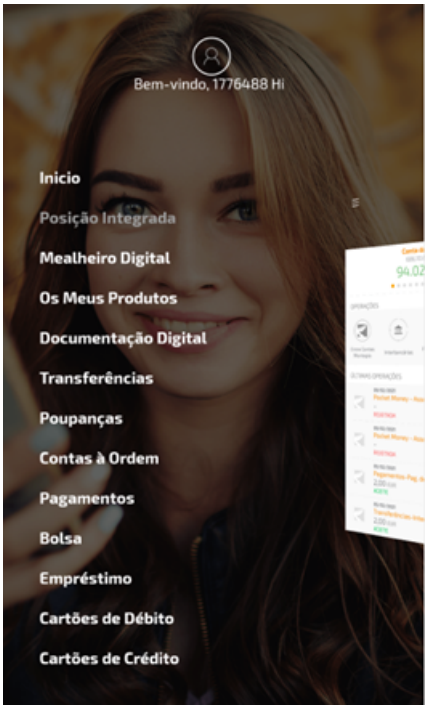


Figura 4.23: Captura de ecrã do menu com opção desabilitada.

4.5.4 Adicionar Elemento ao Menu

Cada elemento do menu é composto pelos seguintes atributos:

- Idr – ID do elemento;
- *Description* – Descrição do elemento / nome do elemento;
- URL – *link* que nos leva até à funcionalidade;
- *isAvailable* – boolean para confirmar se o elemento está ou não disponível para o utilizador;
- *SubMenuResponse* – para elementos de menu que têm sub-menus é dada uma lista de *DynamicMenuItems*, composta por basicamente elementos de menu.

De forma a adicionar uma nova funcionalidade ao menu, é necessário criar um *fragment* para o qual o menu irá direcionar o utilizador assim que o elemento for clicado. Por exemplo, o elemento *'consentimentos'* tem o seguinte fragmento (Trecho de código 4.19):

```
public class ObaConsentsListFragment extends BaseFragment;
```

Trecho de código 4.19: Fragmento `ObaConsentsListFragment`.

Para criar a ligação entre o elemento do menu ao fragmento criado é necessário criar uma `tag` na `class ApplicationWorkFlowController`. Por exemplo, a `tag` da funcionalidade *consentimentos* é a que está representada no Trecho de código 4.20.

```
public static final String GOTO_VERIFY_OBA_TAG = "GOTO_VERIFY_OBA_TAG";
```

Trecho de código 4.20: *Tag* da funcionalidade de Consentimentos.

Ainda na mesma classe, nomeadamente na função `getAvailableApplicationWorkflows()`, é necessário interligar a `tag` criada com o fragmento em questão. O Trecho de código 4.21 exemplifica como tal pode ser conseguido:

```
applicationWorkFlowHashMap.put(GOTO_VERIFY_OBA_TAG, ObaConsentsListFragment.newInstance()
```

Trecho de código 4.21: Interligação da *tag* com o fragmento.

É o *workflow* que alterará o fragmento anterior para o fragmento desejado. A `NavigationDrawer` é composta pelos elementos de menu, como tal a classe `NavigationDrawerObject` serve para determinar o que acontece quando o elemento é selecionado. Nesta classe existe uma função denominada `getGotoDynamicAction(String menuGoTo)` . É esta função que interliga o elemento da *navigation drawer* ao *workflow*. No exemplo do item dos *consentimentos* é acrescentado o seguinte (Trecho de código 4.22):

```
case "consentimentos":
```

```
return ApplicationWorkFlowController.GOTO_VERIFY_OBA_TAG;
```

Trecho de código 4.22: Caso `consentimentos`.

Como tal é imperativo manter o nome do elemento retornado pelo serviço; caso contrário não será efetuada a conexão correta entre o elemento menu e o elemento do serviço.

4.6 Menu Três Níveis

Uma das tarefas estava relacionada com a o desenho e implementação de menus com sub-níveis, em que a primeira opção clicada abra um sub-menu e esse mesmo sub-menu abra um sub-menu. Para isso foram analisadas diferentes possibilidades de implementação:

- Implementação de uma *dialog box* com as opções do sub-menu em que o menu atual continuava igual, mas quando o elemento de sub-menu é pressionado surge uma *dialog box* do estilo da figura 4.24. No topo é colocado o nome do sub-menu, e os seus itens são colocados em baixo do mesmo.

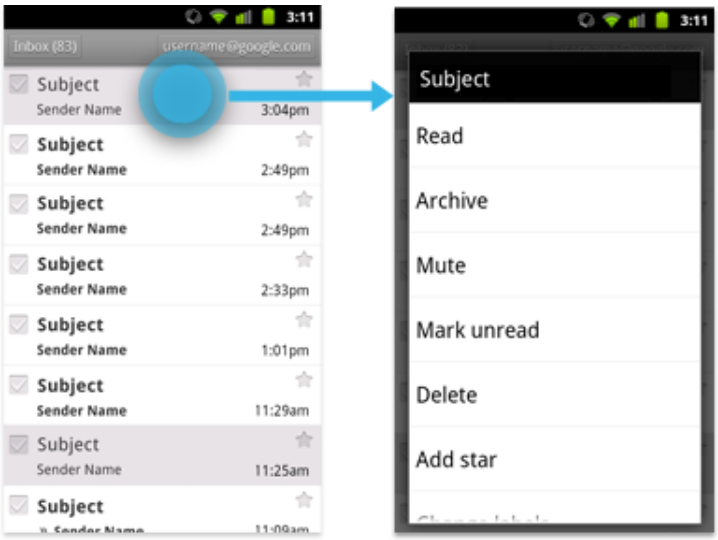


Figura 4.24: Exemplificação da primeira possibilidade de implementação de menu de três níveis.

- Utilização do mesmo menu já implementado, acrescentando um sub-menu a cada sub-menu já existente. Esta opção seria mais confusa, pois o menu utilizado atualmente não utiliza indentação. Desse modo teria de ser utilizada um sistema de cores. Utilizando a captura de ecrã na figura 4.25 como referência, em que *"friends"* já seria o sub-menu e em que os diferentes amigos seriam as opções desse sub-menu, todos eles teriam uma tonalidade diferente dos itens menu. Neste caso *"friends"* seria um elemento de um sub-menu.

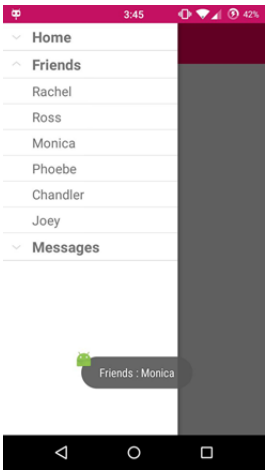


Figura 4.25: Exemplificação da segunda possibilidade de implementação de menu de três níveis.

- Implementação de um *fragment* com as opções do sub-menu. Esta possibilidade pode ser vista em execução, por exemplo, nos menus de definições, em que quando a opção *"definições"* é clicada e surge um *fragment* com o sub-menu. Como podemos ver na figura 4.26, no menu de definições, o clique num item irá despoletar a abertura um sub-menu desse item. Cada um dos elementos desse sub-menu tem também um sub-menu.

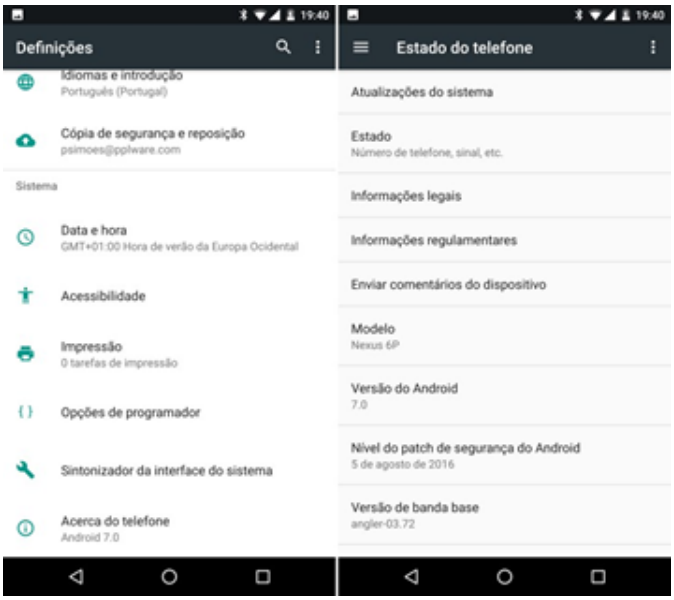


Figura 4.26: Exemplificação da terceira possibilidade de implementação de menu de três níveis.

- Implementação de um menu *dropper*. Isto é, quando o item do sub-menu é clicado, é aberto um sub-menu ao lado do menu existente. Para utilizadores com o ecrã pouco largo, o menu é substituído pelo sub-menu podendo voltar ao menu normal com o premir do botão *back*. Para utilizadores de *tablets*, o sub-menu apareceria disposto ao lado direito do menu já existente, como exemplificado na figura 4.27.

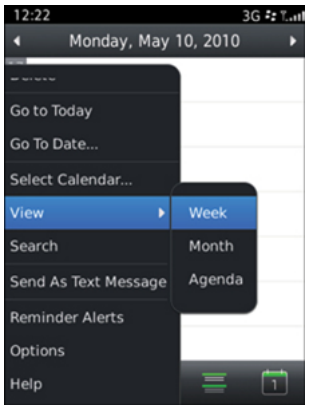


Figura 4.27: Exemplificação da quarta possibilidade de implementação de menu de três níveis.

- Implementação de subdivisões no menu, isto é, assim que o utilizador abre o menu principal, este já vem com o os sub-menus completamente expandidos. Esta opção pode-se tornar um pouco confusa, e o menu pode ficar demasiado grande, pois os sub-menus seriam parte do menu principal. Utilizando a captura de ecrã na figura 4.28 como exemplo, o *"communicate"* seria o item do menu e o *"share"* seria o sub-item que, ao ser clicado, abriria um sub-menu com os seus sub-sub-items.

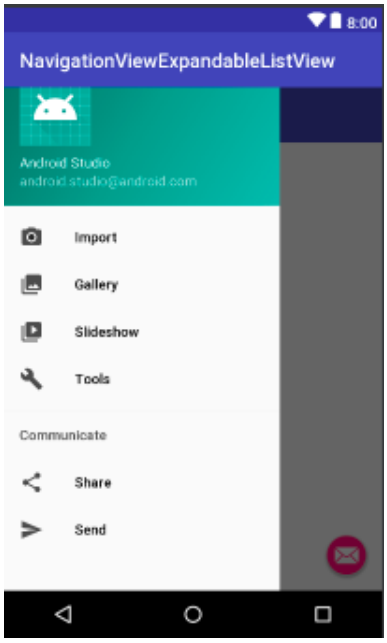


Figura 4.28: Exemplificação da quinta possibilidade de implementação de menu de três níveis.

Após o estudo das diferentes possibilidades, foi escolhida a primeira (a utilização de uma caixa de diálogo). Para a implementação deste menu, foram criados inicialmente itens menus *mokados* (informações exemplo, apenas utilizadas inicialmente para ocupar o espaço da informação final e real). Estes itens teriam uma listagem de outros itens, formando assim um menu com sub-menus.

```
@Override
public void onDynamicMenusStep0TaskSuccess(ResponseDynamicMenusStep0 response) {
        ArrayList<NavigationDrawerObject> navigationDrawerObjectArrayList
                                                        = new ArrayList();

        ArrayList<MenuChild> menuChildList = new new ArrayList();
        ArrayList<MenuChild> subMenuChildList = new new ArrayList();

        for (int i = 0; i < 100; i++) {
                subMenuChildList.add(new MenuChild("SubSubMenu",
                    true, false, false, null));
        }

        menuChildList.add(new MenuChild("SubMenu1", true, false, false,
                                        "hasExtras", subMenuChildList));
        menuChildList.add(new MenuChild("SubMenu2", true, false, false,
                                        "submenu2"));

        navigationDrawerObjectArrayList.add(new NavigationDrawerObject("menu1", "menu1",
                                                true, false, false, menuChildList));
```

Trecho de código 4.23: Criação de itens menu.

Como podemos ver no trecho de código representado pela figura 4.23, foi adicionado um item menu ao `navigationDrawer` que continha uma lista de menu. Esta lista é composta pelo item *"Submenu1"* e *"Submenu2"*, sendo que o *"Submenu1"* continha ainda uma listagem de itens composta apenas por um item de sub-sub-menu denominado por *"SubsubMenu"*. Para adicionar este item aos menus dinâmicos foi necessário criar um segmento que diz respeito ao funcionamento deste item menu, sendo que foi adicionado o seguinte à função `getGotoDynamicAction()` (referente ao trecho 4.24).

```
public static String getGotoDynamicAction(String menuGoTo) {

        if (menuGoTo == null || menuGoTo.isEmpty()){
                return GOTO_NO_MAPPING;
        }

        switch(menuGoTo) {
          case "menu1":
          case "submenu1":
                return null;
          case "submenu2":
          case "inicio":
                return ApplicationWorkFlowController.GOTO_HOMEPAGE_TAG;
```

Trecho de código 4.24: Adicionar menu a lista de ações de menu.

Posto isto, o menu um é composto pelo sub-menu um e sub-menu dois, onde o sub-menu dois executará o código incluído no trecho 4.25):

```
public void popupMenu(String title, ArrayList<MenuChild> subMenuChildList){

    ArrayList<String> labels = new ArrayList();
    for (int i = 0; i < subMenuChildList.size() ; i++) {
        labels.add(subMenuChildList.get(i).getLabelString());
```

```
    }
    String[] labelsArray = new String[ labels.size() ];
    labels.toArray( labelsArray );

    AlertDialog.Builder builder = new AlertDialog.Builder(this, R.style.AlertDialogMenu);
    builder.setTitle(title);
    builder.setItems(labelsArray, new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {

        }
    });
    AlertDialog dialog = builder.create();

    dialog.show();

    DisplayMetrics displayMetrics = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(displayMetrics);
    int displayWidth = displayMetrics.widthPixels;
    int displayHeight = displayMetrics.heightPixels;

    WindowManager.LayoutParams layoutParams = new WindowManager.LayoutParams();

    layoutParams.copyFrom(dialog.getWindow().getAttributes());

    int dialogWindowWidth = (int) (displayWidth * 0.7f);
    int dialogWindowHeight = (int) (displayHeight * 0.7f);

    layoutParams.width = dialogWindowWidth;
    layoutParams.height = dialogWindowHeight;

    dialog.getWindow().setAttributes(layoutParams);
}
```

Trecho de código 4.25: Classe `popupMenu`.

De uma forma simples, pode dizer-se que a função `popupMenu(...)` cria uma caixa de diálogo `AlertDialog` com largura de 70% do ecrã do dispositivo e 70% da altura do ecrã do dispositivo. Nesta caixa são apresentados os sub-Menus existentes na lista de `"subMenuChildList"` criada anteriormente. É atribuído também um *listener* a cada item, de forma a despoletar algo assim que esse item for selecionado.

Em relação à aparência, *layout* e estilos, foram adicionados os seguintes itens ao ficheiro de estilos (Trecho de código 4.26):

```
// menu 3 tears
<style name="AlertDialogMenu" parent="Base.Theme.AppCompat.Light.Dialog">>
  <item name="android:windowBackground">@android:color/white</item>
  <item name="android:windowTitleStyle">@style/MyAlertDialogTitle</item>
  <item name="android:windowContentOverlay">@null</item>
  <item name="android:windowMinWidthMajor">@android:dimen/dialog_min_width_major</item>
  <item name="android:windowMinWidthMinor">@android:dimen/dialog_min_width_minor</item>
  <item name="android:windowIsFloating">true</item>
  <item name="android:textAppearanceMedium">@style/MyAlertTextAppearance</item>
  <item name="android:buttonBarStyle">@style/MyButtonBar</item>
  <item name="android:buttonBarButtonStyle">@style/MyBorderlessButton</item>
</style>
```

```
<style name="MyAlertDialogTitle">
  <item name="android:maxLines">1</item>
  <item name="android:scrollHorizontally">true</item>
</style>
<style name="MyAlertTextAppearance">
  <item name="android:textSize">18dp</item>
  <item name="android:textColor">@color/black</item>
</style>
<style name="MyButtonBar">
  <item name="android:background">@color/white</item>
</style>
<style name="MyBorderlessButton">
  <item name="android:background">@color/white</item>
  <item name="android:textSize">10dp</item>
</style>
```

Trecho de código 4.26: Estilos para *layout*.

O primeiro estilo (`AlertDialogMenu`) é o tema e o estilo utilizado pela caixa de diálogo, o segundo estilo (`MyAlertDialorTitle`) foi criado para alterar o aspeto do título da caixa, o terceiro estilo (`MyAlertTextAppearance`) é utilizado para a aparência do restante texto da caixa e os dois últimos (`MyButtonBar` e `MyBorderlessButton`) são usados para estilizar de botões usados na caixa de diálogo.

4.7 Alteração de *SDK*

A alteração do `compileSdkVersion` e da `targetSdkVersion` da aplicação para 29 causou alguns erros que precisam ser endereçados. Esta secção pretende listar quais esses erros e como estes foram resolvidos. É de salientar que esta alteração foi proposta como desafio, criando um cenário e caso prático de *bug fixing*.

4.7.1 Erro Um – *Library Calligraphy*

Após a análise do erro apresentado na figura 4.30 foi determinado que o erro é causado pela biblioteca *Calligraphy*. Mais concretamente pela versão *Calligrafy* 2, que apresenta um erro que a versão *Calligrafy* 3 veio resolver. Contudo, sendo possível resolver o problema/erro através de uma solução nativa, essa solução é priorizada e implementada de forma a diminuir ou remover dependências exteriores e consecutivamente prevenir que o mesmo problema se repita.

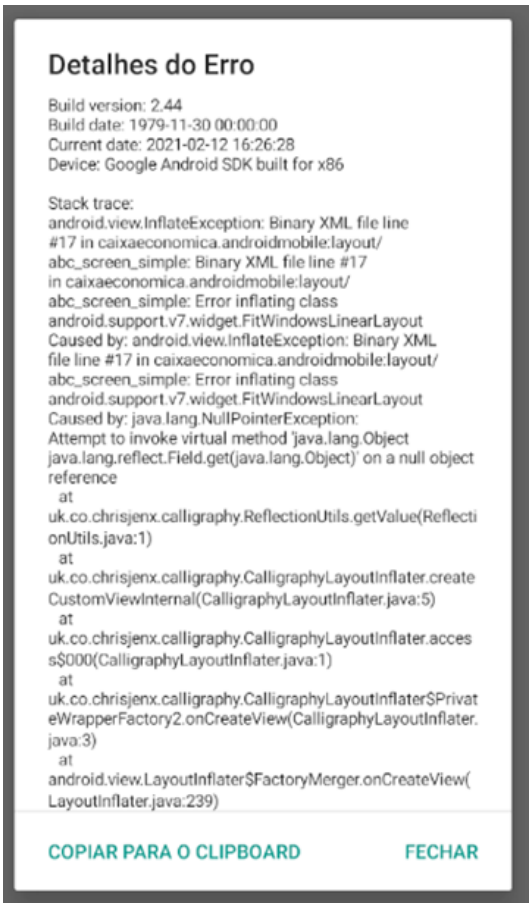


Figura 4.29: Erro apresentado pela aplicação para a biblioteca *Calligraphy*.

Para resolver o problema indicado, foi feita a migração da utilização da biblioteca *Calligrafy* para uma implementação nativa de fontes *Android* através do seguinte processo e passos:

1. Mover os ficheiros da pasta `/app/src/main/assets/fonts` para a pasta `/app/src/main/res/font`;
2. Foi necessário criar uma família de fontes com o ficheiro `exo2_font_family.xml`, sendo esta a família de fontes utilizada pela aplicação (Trecho de código 4.27);

```
<?xml version="1.0" encoding="utf-8"?>
<font-family xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <!-- Regular -->
    <font
        app:font="@font/exo2_regular"
        app:fontStyle="normal"
        app:fontWeight="400" />

    <!-- Light -->
```

```
    <font
        app:font="@font/exo2_light"
        app:fontStyle="normal"
        app:fontWeight="400" />

    <!-- Medium -->
    <font
        app:font="@font/exo2_medium"
        app:fontStyle="normal"
        app:fontWeight="400" />

    <!-- Bold -->
    <font
        app:font="@font/exo2_bold"
        app:fontStyle="normal"
        app:fontWeight="700" />

</font-family>
```

Trecho de código 4.27: Familia de fontes *exo2_font_family*.

3. Seguiu-se a remoção das dependências da biblioteca (Trecho de código 4.28);

```
implementation 'uk.co.chrisjenx:calligraphy:2.2.0'
```

Trecho de código 4.28: Dependência da biblioteca.

4. Removeu-se todo o código relativo à biblioteca *Calligraphy*:
    - Na classe `ApplicationClass.java` (Trecho de código 4.29);

```
MultiDex.install(this);
CalligraphyConfig.initDefault(new CalligraphyConfig.Builder()
    .setDefaultFontPath(Configs.APP_FONTS)
    .setFontAttrId(R.attr.fontPath)
    .build()
);*/
```

Trecho de código 4.29: Classe `ApplicationClass`.

    - Na classe `BaseActivity.java` (Trecho de código 4.30).

```
protected void attachBaseContext(Context newBase) {
    super.attachBaseContext(CalligraphyContextWrapper.wrap(newBase));
}
```

Trecho de código 4.30: Classe `BaseActivity`.

5. Por último passaram-se todas as referências, em *layouts*, de *fontPath* para `android:fontFamily`, assim como todos os *Styles*.

Esta solução foi alcançada através de um processo de análise do erro e pesquisa de diferentes soluções comprovadas, adaptando a solução apresentada no *website* https://davidmigloz.medium.com [Mig20] às especificidades da aplicação.

4.7.2 Erro Dois – *User 10134*

O erro *User 10134* acontece porque, a partir do Android 10, as aplicações devem ter a permissão `READ_PRIVILEGED_PHONE_STATE`, que concede privilégios para aceder aos identificadores não reiniciáveis do dispositivo, que incluem *International Mobile Equipment Identity* (IMEI) e número de série.

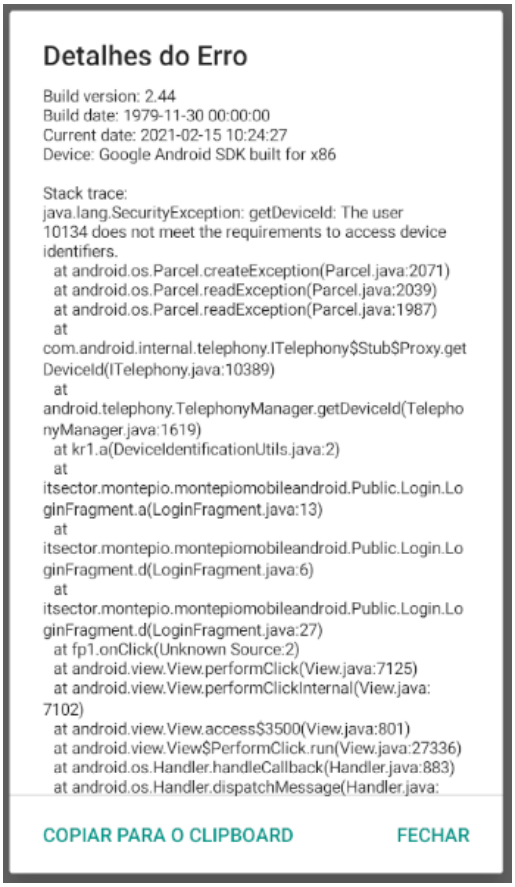


Figura 4.30: Erro apresentado pela aplicação para *user* 10 134.

Segundo os *developers* do *Android*, os métodos afetados por esta alteração são os seguintes:

- *Build* — `getSerial();`
- *TelephonyManager* — `getImei();`
- *TelephonyManager* — `getDeviceId();`
- *TelephonyManager* — `getMeid();`
- *TelephonyManager* — `getSimSerialNumber();`
- *TelephonyManager* — `getSubscriberId()`.

Para resolver este problema, foi necessário seguir os seguintes passos:

- Adicionar a seguinte permissão ao *Android Manifest* (Trecho de código 4.31):

```
<uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE" tools
```

Trecho de código 4.31: Permissão para leitura de estado.

- Alterar o método `generateDeviceId(Context context)` na classe `DeviceIdentificationUtils` para algo que se assemelhe ao que está no trecho de código 4.32 (que verifica versões):

```
public static String generateDeviceId(Context context) {
    String deviceId;

    if (android.os.Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        deviceId = Settings.Secure.getString(
                context.getContentResolver(),
                Settings.Secure.ANDROID_ID);
    } else {
        final TelephonyManager mTelephony = (TelephonyManager)
            context.getSystemService(Context.TELEPHONY_SERVICE);
        if (mTelephony.getDeviceId() != null) {
            deviceId = mTelephony.getDeviceId();
        } else {
            deviceId = Settings.Secure.getString(
                    context.getContentResolver(),
                    Settings.Secure.ANDROID_ID);
        }
    }
    return deviceId;
}
```

Trecho de código 4.32: Método `generateDeviceId`.

Este método gerará um ID para o dispositivo conforme a versão *Android* do dispositivo, i.e., dependendo da versão ser maior ou igual a *Android* 10 ou ser abaixo da versão *Android* 10.

Note-se que um dispositivo pode ter mais que um ID, os caracteres alfabéticos utilizados vão de `A` a `F` e cada tipo de ID tem tamanhos diferentes, sendo que existem os IMEI, os *Mobile Equipment Identifier* (MEID) e os *Electronic Serial Number* (ESN). Cada dispositivo Android tem um ID distinto, podendo este ser redefinido com um *factory reset* ao dispositivo ou alterado num dispositivo com privilégios *root* oum com versões costumizadas do sistema operativo.

Apesar das possibilidades acima descritas, neste caso o projeto utiliza um *SSAID* (*Android* ID) sendo este uma string de 64 bits gerada quando o dispositivo é configurado pela primeira vez. Este ID é obtido pela linha de código apresentada no excerto de código anterior, mais especificamente pela instrução (Trecho de código 4.33):

```
deviceId = Settings.Secure.getString(
                context.getContentResolver(),
                Settings.Secure.ANDROID_ID);
```

Trecho de código 4.33: ID do dispositivo.

No *Android 8 Oreo*, o valor `ANDROID_ID` é exclusivo para cada combinação de chave de assinatura da APP, utilizador e dispositivo, tendo como escopo a chave de assinatura e o utilizador [Gou20].

4.7.3 Erro Três – *mMaxWidth*

Este erro foi detetado através do sistema de *logs*, como pode ser visto na figura 4.31. Após alguma pesquisa foi averiguado que o mesmo erro foi detetado por vários utilizadores da mesma dependência.

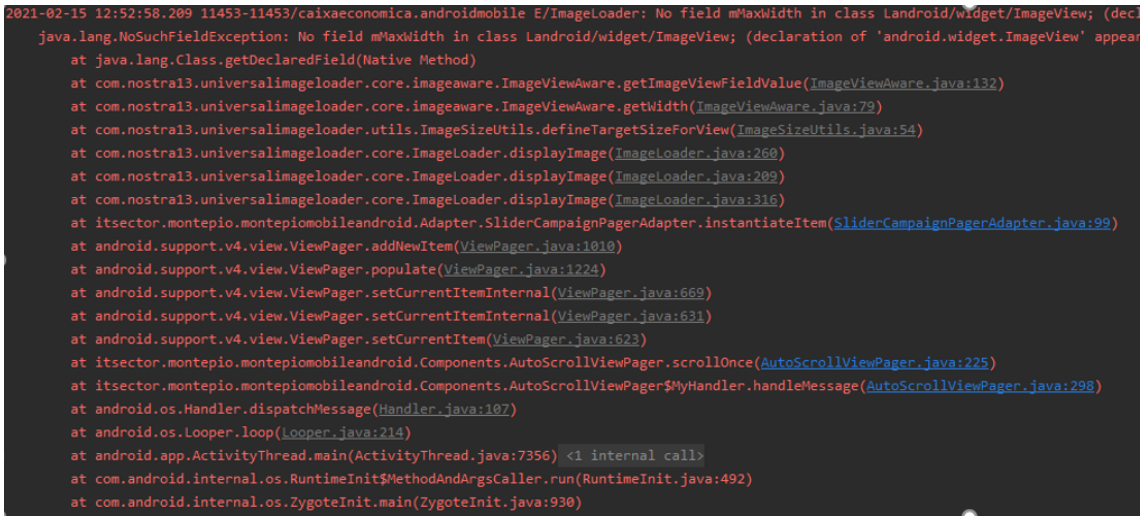


Figura 4.31: Erro apresentado no sistema de *logs*.

Como visto anteriormente, com a evolução das tecnologias algumas dependências ficam fora de uso ou com funções obsoletas. Neste caso foi necessário atualizar a seguinte dependência (Trecho de código 4.34):

```
implementation 'com.nostra13.universalimageloader:universal-image-loader:1.9.5'
```

Trecho de código 4.34: Dependência descontinuada do *load* de imagens.

Substituindo-a pela seguinte (Trecho de código 4.35):

```
implementation 'com.github.nostra13:Android-Universal-Image-Loader:458df4da2e'
```

Trecho de código 4.35: Dependência atualizada de *load* de imagens.

4.8 Sistema de *Logs*

A determinada altura do projeto, foi necessária a criação de um mecanismo de *logs* de raiz. Este envolve a *Plataforma MultiCanal* (PMC) a *Middleware* (MDW) e as APPs. Nesta parte do estágio, foi feita primeiro uma análise inicial do levantamento das vantagens e limitações expressas abaixo indicadas

- (vantagem) Flexibilidade na gestão e armazenamento dos dados – impacto médio;
- (vantagem) Solução criada à medida – impacto médio;
- (vantagem) Multiplataforma (*Android* e iOS) – impacto alto.
- (desvantagem) Exige criar um sistema de raiz – impacto médio;
- (desvantagem) Implica extremo cuidado com a arquitetura para evitar erros – impacto alto:
- (desvantagem) *Bottlenecks* (armazenamento, acesso aos dados);
- (desvantagem) Exposição indevida de dados sensíveis.

Foram apresentadas duas alternativas para a implementação deste mecanismo, descritas com mais detalhe em cada uma das subsecções seguintes.

4.8.1 Alternativa Um

Criação de um novo serviço unidirecional que permita o envio de *batches* de *logs* para a PMC a cada chamada, o que pode ser conseguido com um procedimento semelhante ao seguinte:

- Define-se um *trigger* para o envio de *logs* (após *login*, após cada abertura da APP, etc.);
- Aquando do *trigger*, a APP compila todos os *logs* pendentes que tem em memória local e envia para o servidor;
- Devem ser evitadas abordagens que obriguem a criação de um serviço de *background* (impossível garantir *uptime* do serviço e influência desempenho do dispositivo móvel e consumo de dados).

4.8.2 Alternativa Dois

Integrar nos objetos de *request* (em todos eles) um campo genérico correspondente aos (possíveis) *logs* a enviar:

- Do lado das APPs, todas as comunicações de saída (pedidos aos serviços) seriam intercetadas e, caso existam *logs* pendentes de envio, os mesmos seriam injetados nesse JSON;
- Exigiria uma reformulação mais atenta da arquitetura, quer da APP, quer dos serviços.

4.8.3 Objetivo

Este sistema visa preencher as lacunas na contextualização de cada incidente, como, por exemplo: erros silenciosos / não fatais nas APPs não são persistidos e não existir forma de

obter um contexto alargado do ambiente e processo que levou a determinada anomalia. Como tal foi posta em prática a implementação dois.

Foram criadas duas classes java: `M24Log` e `M24LogsManager`. A classe `M24Log` define o objeto `M24Log`, que tem dois atributos: *timestamp* e *message*. O *timestamp* tem o intuito de guardar o momento em que a ação foi feita e a *message* tem o intuito de guardar a descrição da entrada de *log*. Já a classe `M24LogsManager` visa a adição dos *logs* às preferências, à listagem de *logs* já existentes nas preferências, bem como a sua remoção dos *logs* das preferências. Foram também criados os métodos para as seguintes funções:

- Adicionar novo *log*;
- Adicionar lista de *logs*;
- Obter a lista de todos os *logs*;
- Apagar todos os *logs* das preferências.

4.8.4 Planificação da Implementação

De forma a concretizar esta funcionalidade, estruturou-se a implementação da seguinte forma:

1. Detetar todas as situações em que ocorrem erros não fatais;
2. Detetar todas as situações em que não existe forma de obter contexto alargado do ambiente e processo que levou a determinada anomalia;
3. Implementar as funções já criadas na classe `M24LogsManager` de forma a criar *logs* das situações detetadas e determinadas como anomalia;
4. Criar intercetor de forma a adicionar o novo campo com a informação de *logs* a todos os *requests* existentes;
5. Teste do novo sistema de *logs*.

4.8.5 Implementação do `Intercetor`

No estado atual da implementação seria necessário acrescentar o campo `logsToStore` a todos os *requests*. De forma a não ter de ir a cada *request* e inserir este novo campo terá de ser criado um `Intercetor`.

O INTERCETOR deteta o *body* dos JSON e adiciona um novo campo caso este não exista. Este *intercetor* é adicionado ao `OkHttpClient.Builder` com uma instrução semelhante à incluida no Trecho de código 4.36:

```
okHttpClientBuilder.addInterceptor(CustomInterceptor(getAllLogs()))
```

Trecho de código 4.36: Implementação do *intercetor*.

Após os *logs* serem enviados, deve ser executado o método `deleteAllLogs()`, de forma a não enviar *logs* repetidos.

4.8.6 Análise de Riscos

Existe um limite de 64 kB no armazenamento de *logs* na *Firebase Crashlytics*. Esta solução para base de dados não é suportada por dispositivos *Huawei* e o controlo de dados será delegado à *Firebase*.

De forma a mitigar qualquer problema causado pela *Firebase Crashlytics*, como projeto de longo prazo, é benéfico a criação e investimento numa plataforma centralizada e robusta, capaz de integrar todos os *logs* de cada ponto da arquitetura (APPs, MDW, PMC, etc.). É importante armazenar também o ID da sessão e o *Timestamp*, com o intuito de poder organizar e filtrar os *logs*.

Os *logs* devem apenas apresentar a informação essencial com o intuito de poder solucionar qualquer problema. Deste modo devem ter uma estrutura simples e objetiva. É imperativo que a estrutura da base de dados seja bem organizada e tenha uma arquitetura direcionada a diminuir/eliminar atrasos e pesquisas realizadas.

Todas as informações armazenadas devem ser devidamente encriptadas, e se possível, não armazenar informação pessoal e sensível. De forma a fortalecer a segurança da informação e proteção de dados, o sistema de segurança para estas informações é explicado e referido na secção seguinte (referente ao capítulo 4.9).

4.9 Segurança do Sistema de *Logs*

Um sistema de *logs* tem informações essenciais e críticas para o melhoramento da aplicação, e eventualmente algumas informações privadas no corpo da mensagem. Sendo estas informações críticas, é necessário ter atenção com a forma como se transmitem pela rede e se armazenam remotamente.

A chave utilizada para a encriptação e decifragem dos JSON deve ser de alta segurança, sendo que deve ser usada uma chave assimétrica. Com esta é possível encriptar os *logs* com a chave pública e apenas o utilizador com a chave privada poderá decifrar a informação (neste caso será alguém do lado da ITSector). No caso de haver fuga de informação da base de dados criada, apenas com a chave privada será possível a visualização do texto contido nos *logs*.

Ainda no ponto da segurança dos dados, no que toca ao lado da manutenção e acesso, todos os dispositivos com acesso à base de dados e a informação dos *logs* devem estar protegidos com uma *Virtual Private Network* (VPN). Neste caso apenas poderão aceder à base de dados se o próprio dispositivo for seguro e a sua ligação ao servidor for também segura.

4.9.1 Criptografia Simétrica vs. Assimétrica

Também conhecida como criptografia de chave secreta, a criptografia simétrica é usada por algoritmos como: *Data Encryption Standard* (DES), 3DES, *Advanced Encryption Standard* (AES) e *Rivest Cipher 4* (RC4). Estes algoritmos tendem a ser mais rápidos, mas ao contrário dos algoritmos usados na criptografia assimétrica, a mesma chave tem de ser partilhada entre o emissor e recetor, constituindo esse detalhe um ponto de falha que é critico em algumas situações e que deve ser sempre tomado em consideração no desenho da solução de segurança.

Também conhecida por criptografia de chave pública, a criptografia assimétrica usa um par de chaves assimétricas (privada e pública) onde a pública é usada para cifrar e a privada para decifrar. Em comparação a criptografia simétrica, a criptografia de chave pública tende a ser mais lenta e precisa de mais poder computacional das máquinas. Contudo, a *Internet* é um habitat bastante prolifero para estas técnicas, pois apenas a chave pública é partilhada entre o emissor e o recetor, e a chave privada é usada para decifrar a informação [Pin10].

4.9.2 AES vs. RSA

AES tornou-se no algoritmo de escolha pelos governos, instituições financeiras e empresas que no que toca à cifra de dados. Este algoritmo aplica uma série de transformações matemáticas a cada 128 bits de informação. Sendo este um algoritmo de baixos requisitos, pode ser usado em dispositivos como *smartphones* e computadores, como também para cifrar grandes quantidades de dados. AES é um algoritmo de chave simétrica que pode usar chaves de 128, 192 ou 256 bits. De forma a quebrar este algoritmo, seria necessário confirmar um número mínimo de 2^{127} possibilidades para chaves de 128 bits num ataque de *brute force*.

Rivest-Shamir-Adleman (RSA) é um algoritmo assimétrico que usa uma chave pública para encriptar e uma chave privada para decifrar. A chave pública é o produto da multiplicação de dois números primos em que apenas o produto é tornado público, combinado com outro número inteiro. Para decifrar são necessários os dois fatores principais desse produto, e como não há método conhecido de cálculo dos fatores primos de números tão grandes, apenas o criador da chave pública pode gerar a chave privada necessária para a decifragem.

Por ser necessário mais poder computacional para o algoritmo RSA (tornando-o, portanto, mais lento), este é tipicamente usado para cifrar apenas pequenas quantidades de dados, até porque chaves RSA tem um limite de dados que podem cifrar. E.g., usando o tipo mais comum de preenchimento (*padding* v1.5), o máximo de dados que podem ser cifrados é de 245 bytes [Por18].

4.9.3 Abordagem Hibrida

A criptografia híbrida consiste na união da segurança da distribuição de chaves dada pela criptografia assimétrica e a velocidade de processamento das cifras simétricas. Neste tipo de criptografia, são usadas chaves públicas, privadas e de sessão. A criptografia assimétrica serve então para distribuir a chave simétrica de forma segura, enquanto a criptografia simétrica serve para cifrar a informação em si, pois os respetivos algoritmos são mais rápidos [Dig].

4.9.4 *Android Keystore System*

O *Android keystore system* fornece primitivas de segurança e de extração/armazenamento de chaves criptográficas. Este sistema restringe a utilização das chaves nele guardadas e mitiga o uso não autorizado de chaves no dispositivo Android, fazendo com que os aplicativos especifiquem os usos autorizados das suas chaves e reforçando as restrições fora dos processos das aplicações.

É importante ainda referir que todas as chaves utilizadas para a encriptação de uma base de dados devem ser alteradas dentro de 6 a 12 meses após o início da sua utilização devido ao ciclo que uma chave de encriptação tem. Este aspeto, no caso do Android é controlado pelo sistema *keystore*.

Este sistema requer que o utilizador esteja autenticado e que esta autenticação tenha sido efetuada há pouco tempo. Requer também que todos os usos da chave sejam especificados seja esta chave para encriptar, decriptar, assinar ou verificar.

As classes `KeyStore` e `KeyPairGenerator` (ou `KeyGenerator`) devem ser utilizadas em junção com o provedor `AndroidKeyStore` introduzido no *Android* 4.3. Para gerar uma nova chave privada devem ser especificados os atributos X.509 iniciais, sendo que a implementação padrão para gerar uma chave simétrica válida de 256 bits é a seguinte (Trecho de código 4.37):

```
Context context = getApplicationContext();
MasterKey mainKey = new MasterKey.Builder(context)
        .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
        .build();
```

Trecho de código 4.37: Implementação padrão para gerar uma chave simétrica.

Pode-se também usar um trecho de código semelhante ao seguinte para gerar um par de chaves assimétricas (Trecho de código 4.38):

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance(
        KeyProperties.KEY_ALGORITHM_EC, "AndroidKeyStore");
kpg.initialize(new KeyGenParameterSpec.Builder(
        alias,
        KeyProperties.PURPOSE_SIGN | KeyProperties.PURPOSE_VERIFY)
        .setDigests(KeyProperties.DIGEST_SHA256,
            KeyProperties.DIGEST_SHA512)
```

```
        .build());

KeyPair kp = kpg.generateKeyPair();
```

Trecho de código 4.38: Gerar uma chave simétrica.

Este sistema suporta os seguintes conjuntos de algoritmos de encriptação (Sendo o nível de API mais abrangido o 23+):

- `Cipher`;
- `KeyGenerator`;
- `KeyFactory`;
- `KeyPairGenerator`;
- `Mac`;
- `Signature`;
- `SecretKeyFactory`.

4.10 Programa de Arredondamentos

O programa de arredondamentos consiste na implementação e criação de uma nova funcionalidade que permite ao utilizador arredondar qualquer transferência ou pagamento. Para isso o utilizador deve aceitar os termos deste processo, definir a conta onde aplicar o arredondamento e a partir de que cartão arredondar, como também definir o valor a arredondar e quando vão ser gerados estes arredondamentos. Por fim, o utilizador deve definir uma conta secundária que servirá para creditar caso não seja possível creditar a partir das contas fornecidas inicialmente. A adesão ao programa de arredondamentos é concluída quando o sistema de alertas é devidamente definido.

Inicialmente, a equipa de UI criou *mockups* de forma a definir a aparência do ecrã. Estes *mockups* foram partilhados a partir do *figma*. Com o *figma* é possível retirar informações como cor a utilizar, tipo de letra ou espaçamentos a definir no *layout* final. Dado que este programa foi criado seguindo a metodologia *Scrum*, as diferentes tarefaas foram criadas em diferentes *sprints*. Dito isto, é ainda necessário referir que a presença da estagiária só se deu a partir da *sprint* quatro até à *sprint* oito, tendo início no dia 12 de abril de 2021 e fim no dia 18 de junho de 2021.

4.10.1 Opção de Arredondamento Cartão Débito

Na quarta *sprint*, o objetivo no que diz respeito ao desenvolvimento *Android* dado à estagiária foi a implementação do *layout* para escolher a opção do valor a arredondar a partir dos cartões de débito definidos. A *mockup* que podemos ver na figura 4.32 é referente ao *layout* criado. Este ecrã é composto por:

- Barra de navegação que contem o título do programa;

- Barra de progresso que demonstra o ponto atual do *onBoarding*;
- Dois textos, um para a questão e outra com uma pequena explicação do que é pedido neste ecrã;
- Duas *tabs* para os diferentes tipos de ecrã;
- Os valores dos múltiplos dos quais arredondar;
- Um exemplo prático para cada um dos valores, sendo que altera conforme o valor escolhido.

O pedido tinha ainda as seguintes validações requeridas para a sua aceitação:

- Nenhuma opção de valor deve estar selecionada por defeito quando o ecrã é apresentado;
- O utilizar pode sempre alterar para outro valor;
- Sendo que o utilizador definiu pelo menos um cartão de débito, este deve definir uma opção para essa origem;
- Caso o utilizador não tenha escolhido qualquer cartão de débito as diferentes opções não devem ser apresentadas.

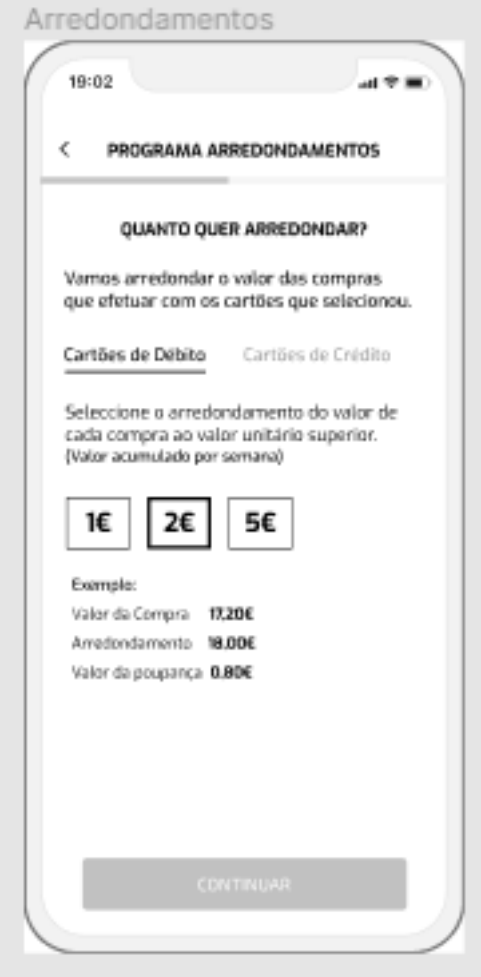


Figura 4.32: *Mockup* do ecrã de Opção de Arredondamento Cartão Débito.

Na implementação deste ecrã foram criados três fragmentos, um para tratar o comportamento geral do ecrã, o segundo para tratar da *tab* dos cartões de débito e o terceiro para tratar da *tab* dos cartões de crédito, apesar deste último não ser ainda necessário, pois neste caso apenas os cartões de débito são tratados. A informação das *tabs* presentes vêm a partir dos serviços e criadas através de um *adapter* (Trecho de código 4.39).

```
private void setupTabLayout(ArrayList<String> listOfTabs,
        List<RoundUpAmountSelectionItem> tabOptionList) {
    if (mTabLayout == null || mTabPager == null) {
        return;
    }

    for (int i = 0; i < listOfTabs.size(); i++) {
        mTabLayout.addTab(mTabLayout.newTab()
        .setText(listOfTabs.get(i)));
```

```
    }

    mTabPager.addOnPageChangeListener(new TabLayout
        .TabLayoutOnPageChangeListener(mTabLayout));

    pagerAdapter = new RoundUpViewPagerAdapter(getChildFragmentManager(),
        mTabLayout.getTabCount(), tabOptionList);
    mTabPager.setAdapter(pagerAdapter);

    setListenerTabLayout();
}
```

Trecho de código 4.39: Metodo `setupTabLayout`.

O código anterior é referente a como o *layout* das *tabs* é criado. É fornecida uma lista de *strings* com o nome de cada uma das *tabs* e uma lista de *tabs* do tipo `List<RoundUpAmountSelectionItem>`, sendo que esta lista de itens contém um índice, um ID, um descritivo, uma lista de itens com os valores a apresentar para os arredondamentos e um *boolean* para saber se esta *tab* é permitida ou não. Através de um ciclo `for` são adicionadas as *tabs* com os devidos nomes ao *layout*. Com um *listener* é possível saber se a *tab* foi alterada e para qual foi. Através do *adapter* a *tab* pode ser apresentada em conformidade com a escolhida.

É no *adapter* que é feita a ligação com os outros dois fragmentos, através de um *switch case* (Trecho de código 4.40):

```
RoundUpAmountSelectionItem item = optionsList.get(position);
switch (item.getId()) {
    case DEBIT_CARD_TAB_ID:
        return RoundUpOnboardingAmountSelectionDebitCardTabFragment
        .newInstance(position, optionsList.get(0).getList());
    case CREDIT_CARD_TAB_ID:
        //return RoundUpOnboardingAmountSelectionCreditCardTabFragment
        .newInstance(position, item.getList());
    default:
        return RoundUpOnboardingAmountSelectionCreditCardTabFragment
        .newInstance(position,  optionsList.get(1).getList());
}
```

Trecho de código 4.40: *Switch case* para interligação dos fragmentos.

Este *switch case* dá o retorno da *tab* dos cartões de débito caso o ID da *tab* selecionada pelo utilizador seja igual à string `DEBIT_CARD_TAB_ID`, criando um fragmento do tipo `RoundUpOnboardingAmountSelectionDebitCardTabFragment`. Caso seja selecionada a *tab* com o ID `CREDIT_CARD_TAB_ID` é criado um fragmento para a *tab* dos cartões de crédito a partir de `RoundUpOnboardingAmountSelectionCreditCardTabFragment.newInstance()` definido pelo *default*.

Para apresentar os valores para arredondar é utilizada uma função denominada `setAmountList(List <RoundUpAmountSelectionItemInnerList> tempList)`, que recebe a lista de itens com os valores a apresentar e, através de um *Linear Layout Manager*, cria

os itens com os respetivos valores. Quando o item é selecionado, são então mostrado os valores de exemplo provenientes dos serviços.

Posteriormente, na quinta *sprint* do processo de criação do programa de arredondamentos, o *layout* deste ecrã foi alterado para o representado na figura seguinte (figura 4.33). Com estas alterações o elemento da barra de progresso foi alterado (exposto na subsecção 4.10.2), o *style* das *tabs* foi alterado e os elementos para a seleção foram também alterados.

Figura 4.33: *Mockup* final do ecrã de escolha do múltiplo de arredondamento.

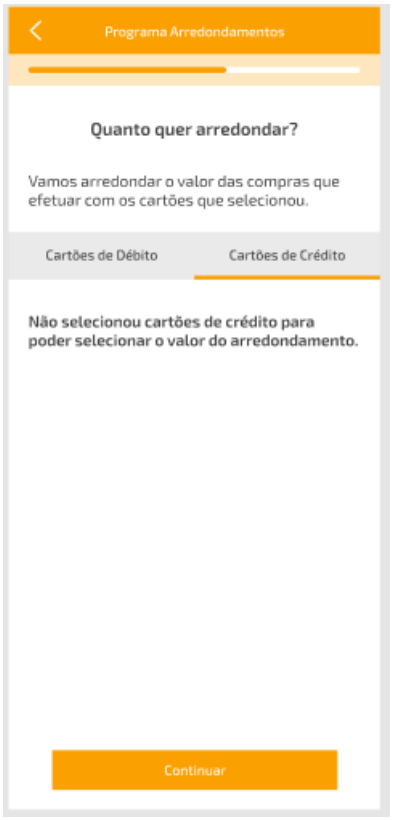


Figura 4.34: *Mockup* final do ecrã de escolha do múltiplo de arredondamento sem cartões selecionados.

Note-se que os elementos eram inicialmente quadrados (no *mockup*) e posteriormente circulares, alterando também a cor conforme esteja selecionado ou não. No caso de nenhum cartão de certo tipo seja selecionado, o *layout* é alterado para o que está representado na figura 4.34, em que o texto *"Não selecionou qualquer cartão para poder selecionar as opões de arredondamento."* é apresentado.

4.10.2 Barra de Progresso

A barra de progresso serve para fornecer ao utilizador a informação do ponto em que está no processo de adesão ao produto de arredondamentos. Este elemento visual precisa de dois valores: os passos totais que o utilizador tem de dar para finalizar o processo e o passo atual. Esses valores são fornecidos pelos serviços em todos os ecrãs (Trecho de

código 4.41).

```
public void setProgressBarValue(int currentStep, int totalSteps) {
    double progress = ((double) currentStep / totalSteps) * 100.0;

    setProgressBarValue((int) progress);
}
```

Trecho de código 4.41: Valores do progresso para barra de progresso.

Como podemos ver no excerto de código mencionado antes, o progresso é definido pela divisão do passo atual pelos passos totais a multiplicar por 100 de forma a fornecer esse valor para a barra de progresso. O XML do *layout* da barra de progresso tem o seguinte (Trecho de código 4.42):

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:id="@+id/roundup_onboarding_progress_frame"
  android:layout_width="match_parent"
  android:layout_height="@dimen/general_margin_30"
  android:layout_below="@+id/toolbar"
  android:background="@color/colorPrimaryOpacity"
  android:visibility="gone">
  <ProgressBar
    android:id="@+id/roundup_onboarding_progress_pb"
    style="@style/RoundUpHorizontalProgressBar"
    android:layout_width="match_parent"
    android:layout_centerInParent="true"
    android:layout_gravity="center_vertical"
    android:layout_marginLeft="@dimen/general_margin_25"
    android:layout_marginRight="@dimen/general_margin_25"
    android:visibility="gone"
    tools:progress="25"
    tools:visibility="visible" />
</FrameLayout>
```

Trecho de código 4.42: XML para *Layout* para barra de progresso.

Este *layout* forma um elemento para se utilizar em todos os ecrãs, o *FrameLayout*, que funciona como *background* e *parent* para a barra de progresso. Para definir o progresso é utilizada a *tool progress* que, neste caso, tem 25% como valor inicial. O *Style* da barra de progresso é utilizado para fornecer a cor da barra, a cor do progresso e arredondar os cantos da barra de forma a ficar o mais parecido com a barra de progresso utilizada em iOS (Trecho de código 4.43).

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

  <item android:id="@android:id/background">
    <shape>
      <corners android:radius="5dp" />
      <solid android:color="@color/progressbar_background_white" />
    </shape>
  </item>
```

```
  <item android:id="@android:id/progress">
    <scale
        android:drawable="@drawable/progressbar_rounded_progress"
        android:scaleWidth="98%" />
  </item>
</layer-list>
```

Trecho de código 4.43: *Layout* de barra do progresso.

O excerto de código anterior pertence ao *drawable* da barra de progresso, tendo então a *layer* do progresso e a *layer* do *background*. Este elemento foi criado na quinta *sprint*, tendo início a 26 de abril e fim a sete de maio.

4.10.3 Layout Tipo de Cartão

Ainda na quinta *sprint* do processo de criação da adesão ao programa de arredondamentos, foi refeito o *Layout* criado inicialmente para a escolha do cartão.

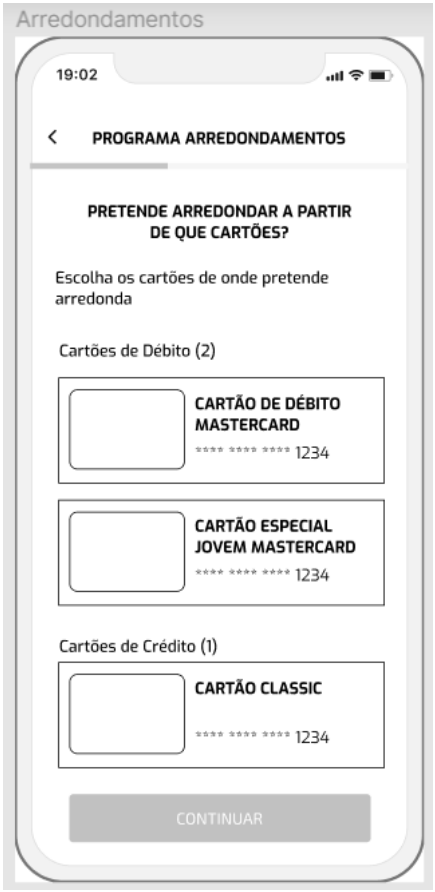


Figura 4.35: *Mockup* inicial do ecrã de escolha do cartão.

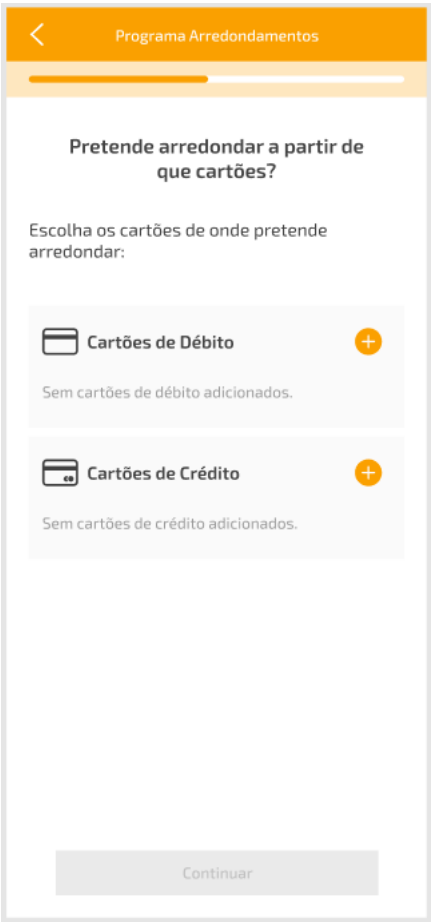


Figura 4.36: *Mockup* final do ecrã de escolha do cartão.

Inicialmente, como podemos ver na figura 4.35, os cartões eram listados e selecionados todos neste ecrã. Já no *layout* final, os cartões não são listados neste ecrã como podemos ver na figura 4.36. Para efetuar esta alteração foi criado um *recycler view* populado por

elementos compostos pelo *drawable* da imagem do cartão, o título do cartão a escolher, um descritivo e um botão com o símbolo mais.

4.10.4 Regras de Aplicação

Na sexta *sprint* foram tratadas as regras de aplicação dos arredondamentos, ou seja, como estes serão aplicados. Para isso foram necessários quatro fragmentos. O primeiro é o ecrã inicial sem qualquer *tab* selecionada, e os outros três para cada uma das *tabs*. As três *tabs* possíveis são: *"Montante"*, *"Periodicidade"* e *"Mais Tarde"*.

Na primeira imagem apresentada a seguir (figura 4.37) podemos ver o ecrã inicial, e como este é composto pela barra de navegação, a barra de progresso, dois textos para explicar o que se pretende a partir deste ecrã, três *tabs* e um *date picker* (um seletor de datas).

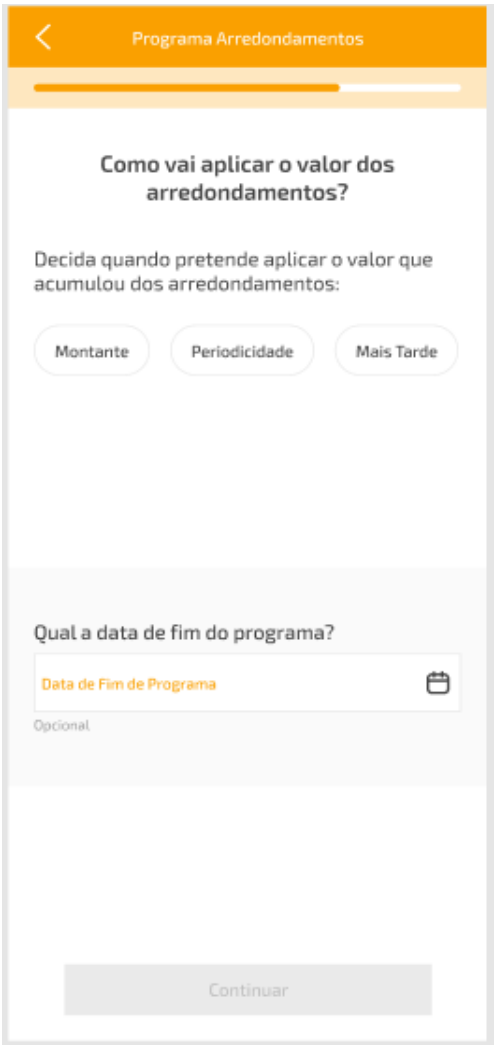


Figura 4.37: Captura de ecrã das regras de aplicação.

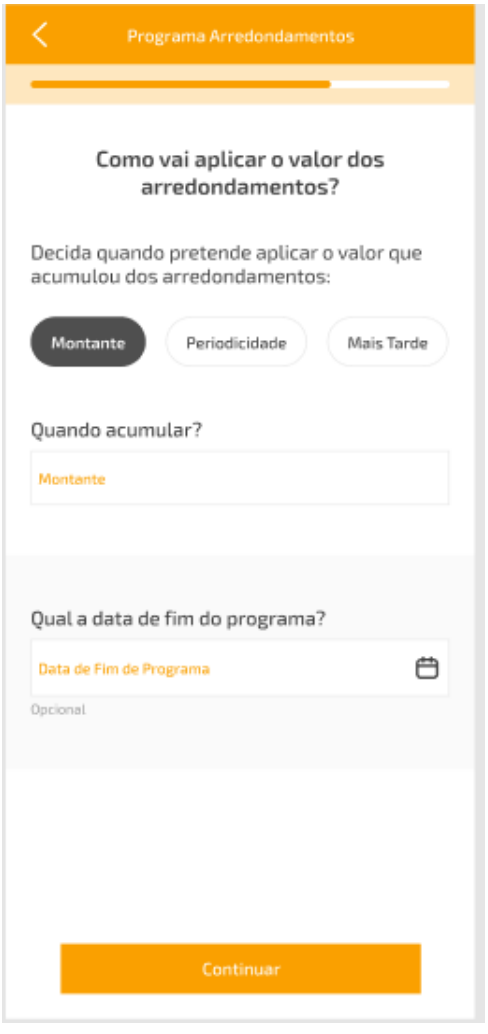


Figura 4.38: Captura de ecrã das regras de aplicação da *tab* Montante.

O *date picker* é estático, estando presente em todas as *tabs*. Este elemento foi importado de outro fragmento que já existia na aplicação e apenas ajustado a este ecrã, sendo que deve conter o símbolo do calendário do lado direito e do lado esquerdo deve ter o texto

"Data de Fim de Programa". Esta data é opcional, pelo que o utilizador pode avançar sem escolher uma data. Contudo, para avançar, deve escolher uma das *tabs* dispostas.

A figura 4.38 representa a primeira *tab*, denominada *"Montante"*. Nesta perspetiva, o utilizador pode escolher quanto acumular, sendo o valor mínimo de 5 e o máximo de 100. Este campo apenas aceita valores inteiros e numéricos, caso o utilizador tente avançar com valores menores ou superiores ao estipulado aparece um erro, não deixando o utilizador avançar sem alterar o valor. Estes valores vêm dos serviços, sendo então utilizados no seguinte código.

```
 if (mMax.equals("0")) {
    if (Double.parseDouble(mMin) <= Double.parseDouble(amount)) {
        flag = true;
    }
} else {
    if (Double.parseDouble(mMin) <= Double.parseDouble(amount)
    && Double.parseDouble(mMax) >= Double.parseDouble(amount)) {
        flag = true;
    } else if (Double.parseDouble(amount) <= Double.parseDouble(mMin)) {
        if (isToShowMessage) {
            field.setError(context.getResources()
                .getString(R.string.error_invalid_amount_min, mMin));
        }
    } else {
        if (isToShowMessage) {
            field.setError(context.getResources()
                .getString(R.string.error_invalid_amount_max, mMax));
        }
    }
}
```

Trecho de código 4.44: Vareficação do valor máximo e minimo.

Este código (trecho de código 4.44) verifica se o valor máximo não é igual a zero, se não é confirmado que o valor inserido pelo utilizador é maior que o mínimo estipulado proveniente dos serviços e menor que o máximo. Caso o valor seja maior que o mínimo o erro "Deverá inserir um valor superior a (mínimo) Euros." em que o valor mínimo aparece em vez dos parênteses é mostrado. Caso o valor seja superior ao máximo é mostrado o seguinte erro: "Deve inserir um valor inferior a (máximo) Euros." em que em vez dos parênteses aparece o valor máximo proveniente dos serviços.

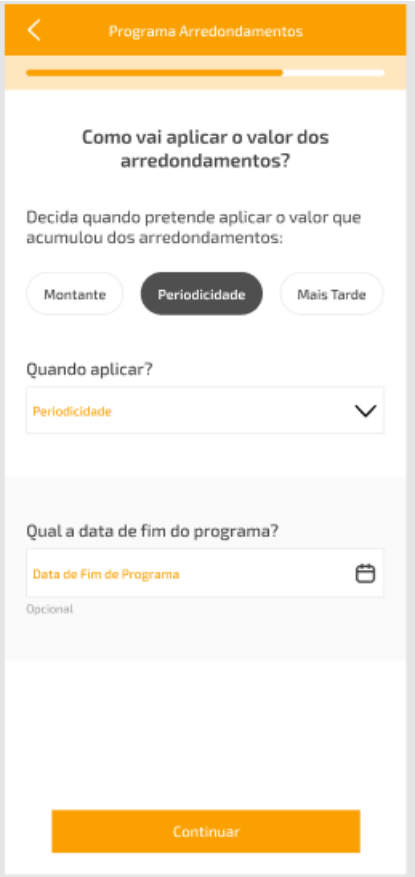


Figura 4.39: Captura de ecrã das regras de aplicação *tab* Periodicidade.

Figura 4.40: Captura de ecrã das regras de aplicação da *tab* Mais Tarde.

A *tab "Periodicidade"*, exposta na figura 4.39, possibilita ao utilizador escolher o período para aplicar o valor dos arredondamentos, sejam estes mensalmente, diariamente ou semanalmente. O utilizador pode escolher de entre as opções provenientes dos serviços através de um *dropdown* que demonstra essas mesmas opções.

Caso o utilizador não queira decidir como aplicar o valor dos arredondamentos no momento da adesão, este pode escolher a opção *"Mais Tarde"* (figura 4.40), que permite ao utilizador avançar.

4.10.5 Definição da Conta a Ordem

Na sétima *sprint* foi criado o ecrã para a escolha da conta a ordem no caso de não ser possível creditar nos destinos escolhidos. O ecrã criado (representado na figura 4.41) é formado pela barra de navegação, dois textos descritivos para o utilizador saber para que serve este ecrã e um elemento para escolha da conta a ordem. Este elemento já estava criado na aplicação e foi apenas reutilizado, desta feita utilizando as contas à ordem provenientes dos serviços para este ecrã.

Figura 4.41: Captura de ecrã para a definição da conta a ordem.

4.10.6 Refatorização e *Bugs*

A oitava *sprint* serviu para rever e refazer certos elementos (refactorização e correção de *bugs*), tais como: tornar textos dinâmicos e alteração de textos e cores. A titulo de exemplo, mostra-se o ecrã de distribuição de percentagens (figura 4.42), onde foram feitos ajustes gráficos. Inicialmente, quando a soma das percentagens excedia os 100% o utilizador não era notificado. Com as alterações operadas, verifica-se se o somatório das percentagens não excede 100% e, se tal acontecer, surge a nota indicativa que *"Aplicou x% a mais."* em vez de *"Falta aplicar x%."* apresentado na figura 4.42.

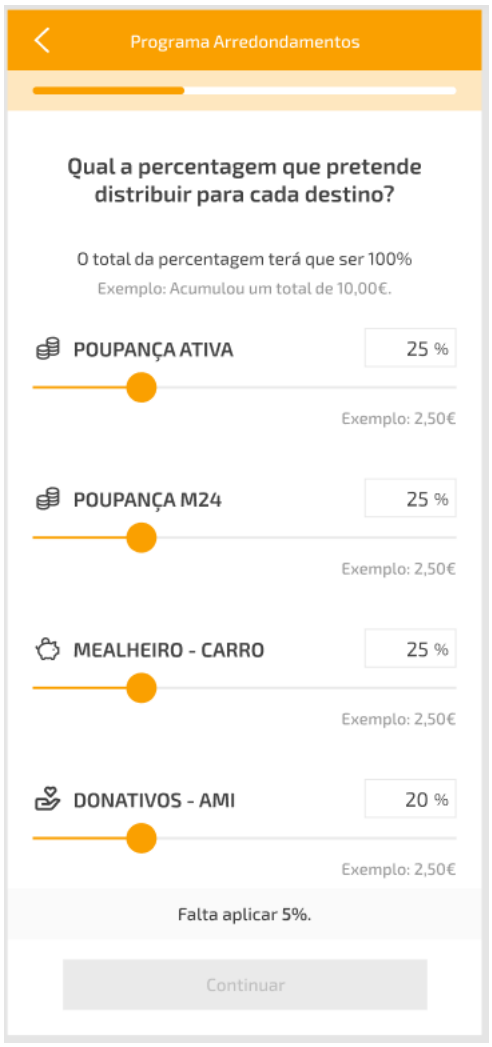


Figura 4.42: Captura de ecrã para a distribuição percentual.

4.11 Gerar Novas Versões

Para que a aplicação e o código possa ser testado e disponibilizado na *appstore*, é necessário gerar uma versão da aplicação para a instalação num dispositivo. É de realçar que estes testes não são automatizados, mas sim testes manuais. São feitos em paralelo com a aplicação iOS de forma a detetar diferenças e incongruências nas aplicações. Como tal, dependendo da funcionalidade a ser testada, podem ser geradas duas versões, uma para clientes particulares e outra para empresas. O exemplo que veremos nesta secção é relacionado com gerar uma versão de testes para o programa de Arredondamentos (secção 4.10).

O *Android Studio* fornece uma opção de gerar *Android Package* (APK), clicando na aba `Build` e selecionando a opção `Generate Signed Bundle / APK...` como podemos ver na figura 4.43.

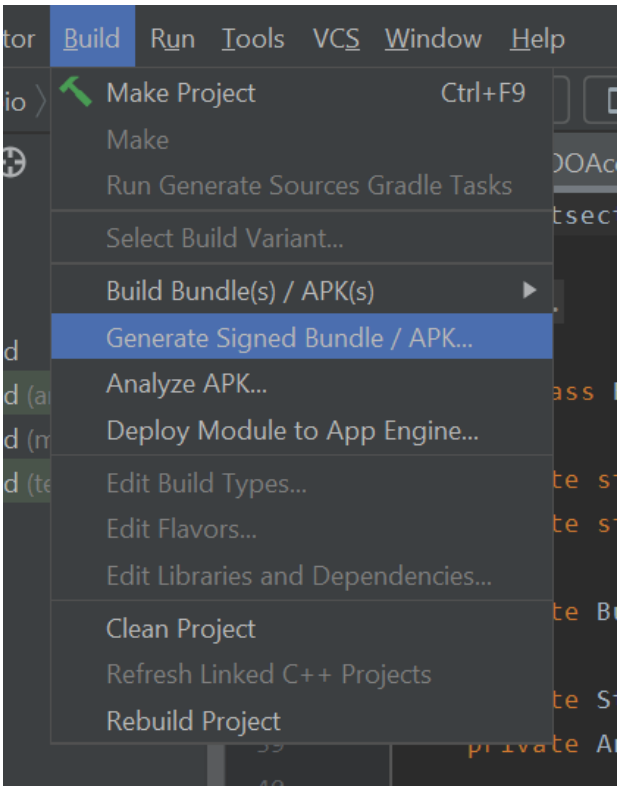


Figura 4.43: Captura de ecrã da aba `Build` do *Android studio*.

Abrindo a janela denominada `Generate Signed Bundle / APK...`, é necessário escolher entre um *Bundle* e uma APK, sendo que neste projeto só trabalhamos com APKs, esta opção é escolhida como demonstra a figura 4.44.

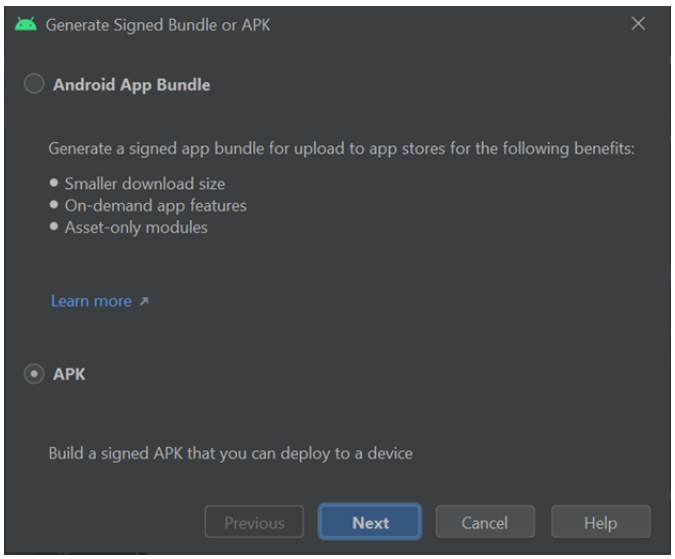


Figura 4.44: Captura de ecrã da janela de diálogo `Generate Signed Bundle / APK`.

Após a seleção de APK é necessário fornecer uma chave para posteriormente publicar a aplicação na *appStore* (figura 4.45). Dado existirem duas aplicações (clientes particulares e empresas) é preciso ter em atenção qual a chave utilizar para ir em conformidade com

a versão a ser gerada.

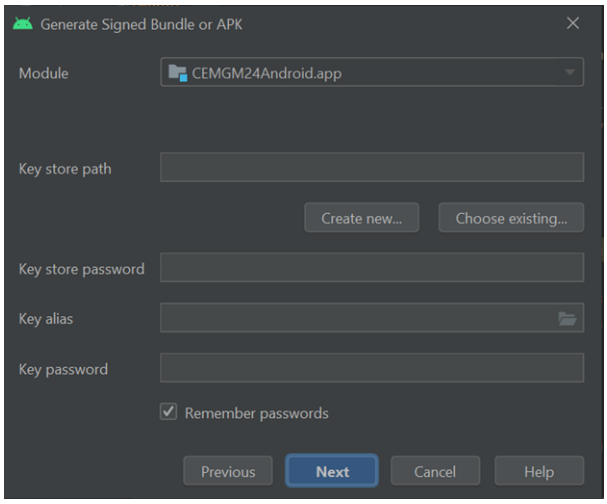


Figura 4.45: Escolha das chaves para gerar versão.

Por fim, é escolhida a variante que queremos gerar, neste caso sendo uma funcionalidade para clientes particulares é gerada uma versão no ambiente de qualidade 4.46.

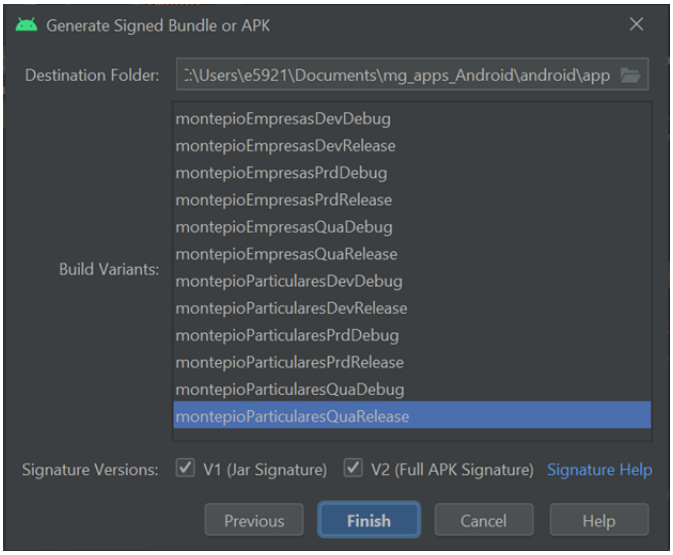


Figura 4.46: Escolha da *Build Variant*.

Depois da APK gerada, é então feito o *upload* da mesma para o *APP Center*, que é um repositório de versões da aplicação criado para a partilha das mesmas entre a equipa. Este tem uma listagem de aplicações que foram criadas para diferentes versões da aplicação e diferentes funcionalidades da aplicação.

Indo de encontro ao exemplo de gerar versão para clientes particulares com a funcionalidade dos arredondamentos, é escolhida a aplicação dos arredondamentos na listagem de APPs do *APP Center* e criada uma *release*. Podendo também escolher quais os destinatários desta *release* e adicionar notas, caso seja necessário (figura 4.47).

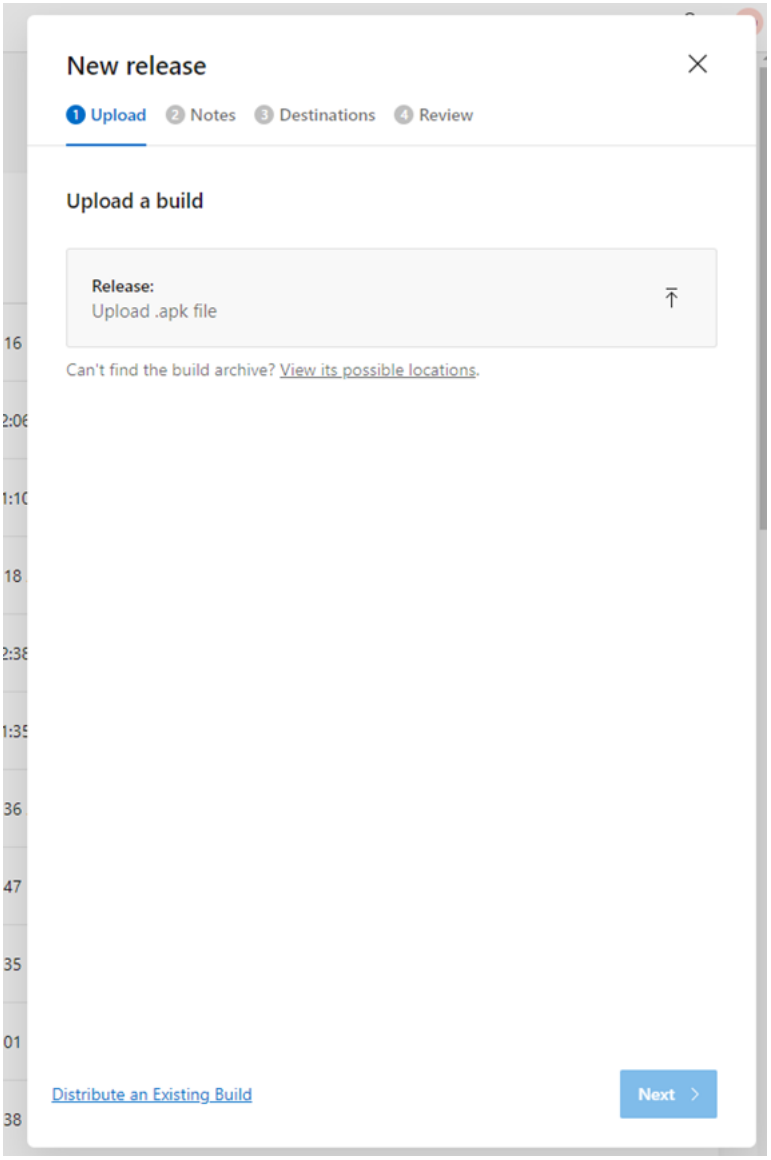


Figura 4.47: Upload da *release* no *APP Center*.

4.12 Conclusão

Na finalização deste capítulo é possível afirmar que a implementação de novas funcionalidades, criação de *layouts* e criação de testes automatizados utilizando as tecnologias descritas e escolhidas no subcapítulo 2.6 foram bem sucedidas neste estágio. Tendo também em conta o planeamento inicial e o mapa de *Gantt* criado para as tarefas previstas, foram todas executadas com sucesso e expostas neste capítulo, dando como cumprido o objetivo do estágio. Novos conhecimentos foram adquiridos tanto na utilização das ferramentas como na escrita de código e execução do mesmo para a criação de funcionalidades de uma aplicação *Android* e testes das mesmas.

Capítulo 5

Conclusão

Rematando este estágio e relatório de estágio, este capítulo resume as conclusões principais do mesmo, bem como a indicação de trabalho futuro.

5.1 Conclusões Principais

Em relação aos objetivos originais e principais do estágio, pode-se afirmar que todos foram atingidos com sucesso, nomeadamente através da concretização bem sucedida de todas as diferentes tarefas e através da criação de diferentes funcionalidades e testes automatizados. Pode-se ainda dizer que o contacto com o mundo real do trabalho foi iniciado e cristalizado com sucesso, contando com diferentes vertentes, desde funcionamento de uma equipa real de desenvolvimento de software, à introdução inicial com os recursos humanos da empresa, no que toca a entrevistas e à academia inicial.

Com este estágio foi possível desenvolver aptidões, não só de trabalho em contexto profissional e de *developer*, como também algumas *soft skills* importantes a manter e desenvolver para o resto da vida. Essas aptidões vão desde a planificação, ao estudo do estado da arte e das diferentes tecnologias, à implementação e testes. Foi também possível desenvolver diferentes capacidades no que diz respeito ao conhecimento da área *Android* e no desenvolvimento de funcionalidades e de uma aplicação móvel.

É importante ainda referir que o mundo real de trabalho é atravessado por diferentes dificuldades e incertezas, e que é apenas através da perseverança, estudos e tentativas que os objetivos são possíveis de se alcançar.

5.2 Trabalho Futuro

Na área e nas aplicações *Android*, as possibilidades são infindáveis. Hoje em dia há aplicações móveis praticamente para tudo e, focando no ramo financeiro e, neste caso, do projeto M24, a evolução e manutenção é um imprescindível linha de trabalho futuro.

Sendo que a empresa onde o estágio foi realizado trabalha com clientes externos, muitas das funções criadas foram pedidas, ajustadas e seguidas pelo cliente. Neste sentido, a implementação vai ao encontro das necessidades do cliente e dos pedidos do mesmo, com o passar do tempo diferentes pedidos surgem e diferentes funcionalidades são criadas para manter a aplicação atualizada e funcional.

Bibliografia

[Agi14] Desenvolvimento Agil. SCRUM [online]. 2014. Available from: https://www.desenvolvimentoagil.com.br/scrum/ [cited 24 Maio 2021]. 19

[Agr20] Crédito Agrícola. moey! – conta digital com zero custos [online]. 2020. Available from: https://play.google.com/store/apps/details?id=pt.moey.app [cited 30 Novembro 2020]. 7

[atl20] atlassian. Gitflow Workflow [online]. 2020. Available from: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow [cited 7 Dezembro 2020]. 15

[BB20] S.A Banco BPI. BPI APP [online]. 2020. Available from: https://play.google.com/store/apps/details?id=pt.bancobpi.mobile.fiabilizacao [cited 30 Novembro 2020]. 7

[Cri16] Pedro Crisóstomo. Onde começa e acaba o sigilo da nossa conta no banco? [online]. 2016. Available from: https://www.publico.pt/2016/08/26/economia/noticia/onde-comeca-e-acaba-o-sigilo-da-nossa-conta-no-banco-1742284 [cited 20 Janeiro 2021]. 103

[dB21a] Associação Portuguesa de Bancos. Chamadas de Vishing [online]. 2021. Available from: https://www.apb.pt/content/files/Vishing_PT.pdf [cited 18 Janeiro 2021]. 12

[dB21b] Associação Portuguesa de Bancos. E-mails de Phishing [online]. 2021. Available from: https://www.apb.pt/content/files/Phishing_PT.pdf [cited 18 Janeiro 2021]. 11

[dB21c] Associação Portuguesa de Bancos. Fraudes de investimento [online]. 2021. Available from: https://www.apb.pt/content/files/Investment_scams_PT.pdf [cited 18 Janeiro 2021]. 11

[dB21d] Associação Portuguesa de Bancos. Sites Bancários Falsos [online]. 2021. Available from: https://www.apb.pt/content/files/Spoof_bank_websites_PT.pdf [cited 18 Janeiro 2021]. 12

[dB21e] Associação Portuguesa de Bancos. SMSs de fishing [online]. 2021. Available from: https://www.apb.pt/content/files/Smishing_PT.pdf [cited 18 Janeiro 2021]. 11

[Dig] IMD Metrópole Digital. Estratégia Híbrida – Criptografia Simétrica e Assimétrica [online]. Available from: https://materialpublic.imd.ufrn.br/curso/disciplina/4/62/3/9 [cited 31 Maio 2021]. 70

[Dri10] Vincent Driessen. A successful Git branching model [online]. 2010. Available from: https://nvie.com/posts/a-successful-git-branching-model/ [cited 7 Dezembro 2020]. xv, 16

[Eco20] O Jornal Económico. Estas aplicações de bancos ajudam-no a manter as finanças em ordem [online]. 2020. Available from: https://jornaleconomico.sapo.pt/noticias/estas-aplicacoes-de-bancos-ajudam-no-a-manter-as-financas-em\-ordem-531340 [cited 26 Novembro 2020]. 7, 8

[fBoV21a] Global Alliance for Banking on Values. Find Members [online]. 2021. Available from: http://www.gabv.org/the-community/find-members [cited 18 Janeiro 2021]. xvi, 100

[fBoV21b] Global Alliance for Banking on Values. Members [online]. 2021. Available from: http://www.gabv.org/the-community/members [cited 18 Janeiro 2021]. 99

[Ger20] Caixa Económica Montepio Geral. M24 [online]. 2020. Available from: https://play.google.com/store/apps/details?id=caixaeconomica.androidmobile [cited 30 Novembro 2020]. 10

[Goo20] Google. Android Studio release notes [online]. 2020. Available from: https://developer.android.com/studio/releases#older-releases [cited 7 Dezembro 2020]. 14

[Gou20] William Gouvea. How to generate Android Unique ID [online]. 2020. Available from: https://proandroiddev.com/how-to-generate-android-unique-id-38362794e1a8 [cited 31 Maio 2021]. 65

[Gra20] Dan Graham. Appium Desktop [online]. 2020. Available from: https://github.com/appium/appium-desktop [cited 16 Dezembro 2020]. 17

[ITS20a] ITSector. Sobre Nós [online]. 2020. Available from: https://www.itsector.pt/pt/sobre-nós [cited 20 Novembro 2020]. 1

[ITS20b] ITSector. Visão geral [online]. 2020. Available from: https://www.linkedin.com/company/itsector/about/ [cited 20 Novembro 2020]. 2

[Kag20] Julia Kagan. Home Banking [online]. 2020. Available from: https://www.investopedia.com/terms/h/home-banking.asp [cited 26 Novembro 2020]. 6

[lem18] lemuel. Find Members [online]. 2018. Available from: https://blog.firgun.com.br/bancos-eticos/ [cited 18 Janeiro 2021]. 99

[Mig20] David Miguel. Migrating custom fonts from Calligraphy to Android fonts [online]. 2020. Available from: https://medium.com/@davidmigloz/

migrating-custom-fonts-from-calligraphy-to-android-fonts-3b635fd945dc [cited 31 Maio 2021]. 62

[Mil20] Milleniumbcp. Millenniumbcp [online]. 2020. Available from: https://play.google.com/store/apps/details?id=wit.android.bcpBankingApp.millennium [cited 30 Novembro 2020]. 8

[Mon19] Banco Montepio. Política de Privacidade [online]. 2019. Available from: https://www.bancomontepio.pt/institucional/modelo-governo/politicas-regulamentos/politica-privacidade [cited 18 Janeiro 2021]. 14

[Mon21] Banco Montepio. Sobre nós [online]. 2021. Available from: https://pt.linkedin.com/company/montepio [cited 18 Janeiro 2021]. 12

[Pa□02] Dr. Jorge Patrício Paúl. Jorge Patrício Paúl - O Sigilo Bancário e a sua relevância Fiscal [online]. 2002. Available from: https://portal.oa.pt/publicacoes/revista/ano-2002/ano-62-vol-ii-abr-2002/temas-e-referencias/jorge-patricio-paul-o-sigilo-bancario-e-a-sua-relevancia\-fiscal/ [cited 20 Janeiro 2021]. 103

[Pin10] Pedro Pinto. Criptografia simétrica e assimétrica. Sabe a diferença? [online]. 2010. Available from: https://pplware.sapo.pt/tutoriais/networking/criptografia-simetrica-e-assimetrica-sabe-a-diferenca/ [cited 31 Maio 2021]. 69

[Por92] Portugal. Regime Geral das Instituições de Crédito e Sociedades Financeiras - Decreto-Lei n.º 298/92 - Diário da República n.º 301/1992, 6º Suplemento, Série I-A de . *Diário da República Eletrónico de Portugal*, 1992. Available from: https://dre.pt/web/guest/legislacao-consolidada/-/lc/141459985/202101180000/73871820/diplomaPagination/diploma/1?did=70072322&filter=Filtrar. 101, 102, 103

[Por18] Thomas Pornin. RSA maximum bytes to encrypt, comparison to AES in terms of security? [online]. 2018. Available from: https://security.stackexchange.com/questions/33434/rsa-maximum-bytes-to-encrypt-comparison-to-aes-in-terms-of-security [cited 31 Maio 2021]. 69

[PT19] Bancos PT. O que é o Homebanking [online]. 2019. Available from: https://bancos.pt/homebanking/ [cited 26 Novembro 2020]. 6

[Tei21] Alberto Teixeira. Banco Montepio tem luz verde do Governo para a saída de até 400 trabalhadores [online]. 2021. Available from: https://eco.sapo.pt/2021/01/12/banco-montepio-tem-luz-verde-do-governo-para-a-saida-de-ate\-400-trabalhadores/ [cited 18 Janeiro 2021]. 12

Apêndices

Apêndice A

Mapa de *Gantt*

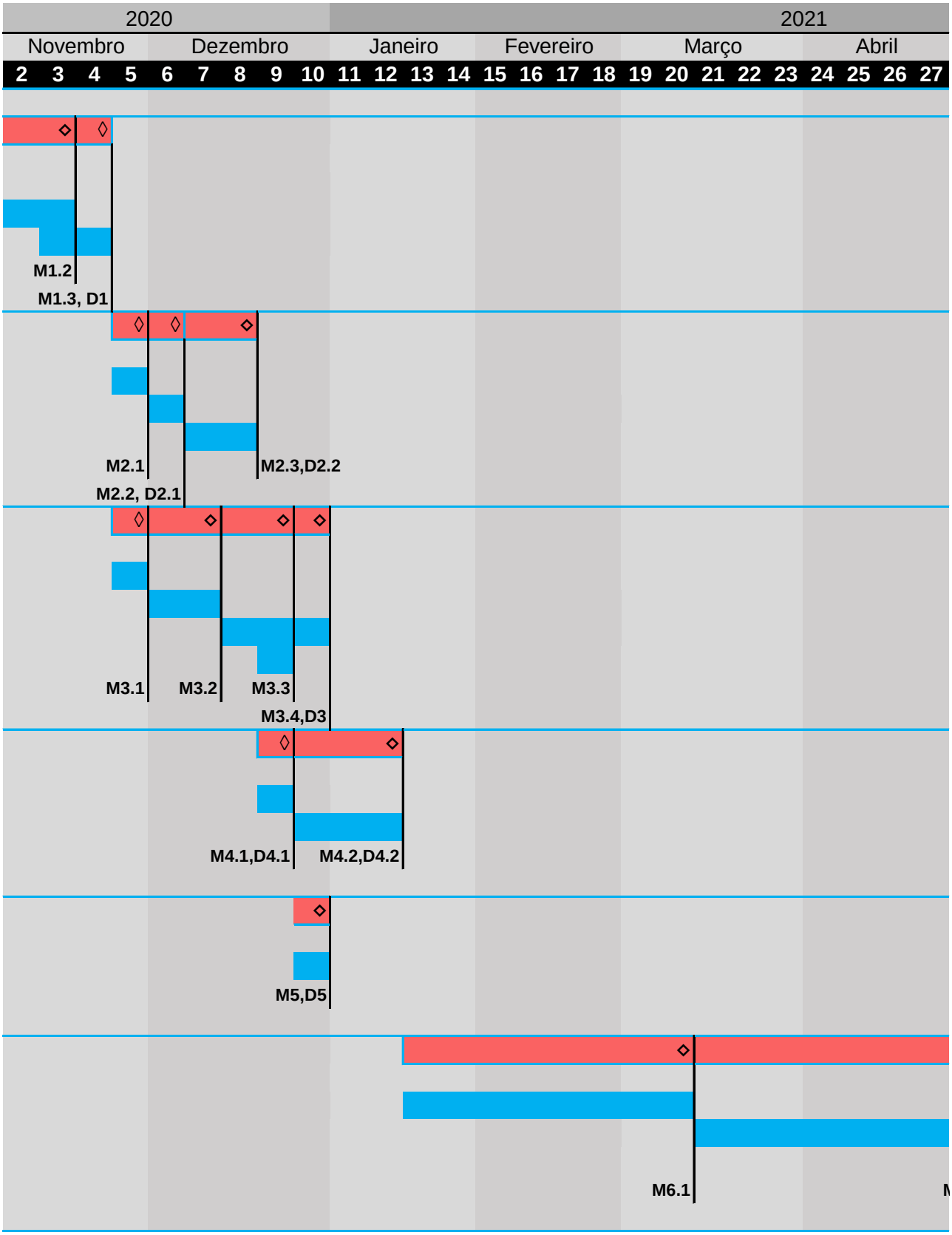
Este apêndice tem como intuito mostrar com mais detalhe o mapa de *Gantt* criado para a secção 3.4. As páginas seguintes são o ficheiro *Portable Document Format* - Formato de Documento Portátil (PDF) criado a partir do *Excel* do mapa de *Gantt* de forma a uma melhor visibilidade do mesmo.

M24 - Aplicação Android Banco Montepio

ITSector

Tarefa	semanas →	Out 1
Tarefa 1 - Academia Android → Tarefa 1.1 - Modulo 1: Tutoriais de iniciação. → Tarefa 1.2 - Modulo 2: Tutoriais avançados. → Tarefa 1.3 - Projeto final da academia.		◊ M1.1
Tarefa 2 - Ambientação ao projeto M24 → Tarefa 2.1 - Estudo da estrutura do projeto Android e da sua ramificação Git. → Tarefa 2.2 - Criação de novo item no menú da Aplicação Android. → Tarefa 2.3 - Implementação de nova funcionalidade ("Consulta de Posição Integrada").		
Tarefa 3 - Estudo das tecnologias e Estado da arte → Tarefa 3.1 - Estudo do Git e da sua funcionalidade/ramificações. → Tarefa 3.2 - Exploração das aplicações já existentes no ramo de HomeBanking. → Tarefa 3.3 - Estudo da ferramenta Appium e dos testes automatizados. → Tarefa 3.4 - Estudo dos testes automatizados.		
Tarefa 4 - Planificação e Implementação de testes para a funcionalidade OpenBanking → Tarefa 4.1 - Estudo da funcionalidade e planificação das partes a serem testadas. → Tarefa 4.2 - Implementação de testes.		
Tarefa 5 – Implementação de Layouts → Tarefa 5.1 - Estudo do Layout e implementação deste.		
Tarefa 6 – Planificação, implementação e testes de novas funcionalidades → Tarefa 6.1 - Planificação → Tarefa 6.2 – Implementação → Tarefa 6.3 – Testes		

Tarefa 7 – Escrita do relatório de estágio
→ Tarefa 7.1 - Relatório de planificação de estágio → Tarefa 7.2 - Relatório de Testes automatizados → Tarefa 7.3 - Relatório de estágio


2020
2021
Novembro
Dezembro
Janeiro
Fevereiro
Março
Abril
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
M1.2
M1.3, D1
M2.1
M2.2, D2.1
M2.3,D2.2
M3.1
M3.2
M3.3
M3.4,D3
M4.1,D4.1
M4.2,D4.2
M5,D5
M6.1


M7.1,D7.1
M7.2,D7.2

Apêndice B

Ética Bancária

Este apêndice visa o estudo e a inicialização do que é a ética no mundo financeiro, quais os deveres e princípios das instituições e o sigilo profissional e bancário. Este é um ponto sempre importante estudar antes de qualquer envolvimento com um novo setor. Estando em contacto com um banco e com os seus serviços, o programador entra em contacto com os utilizadores das aplicações e do banco, como tal, ter uma boa ética é imperativo.

B.1 Banco Ético

Um banco não tem dinheiro, ele usa o dinheiro dos seus clientes. O dinheiro de um cliente que poupa é emprestado a um cliente que quer um empréstimo, ou investido em empresas e títulos públicos.

Os bancos éticos são instituições financeiras cujos produtos não estão voltados para o lucro. Estes bancos investem na economia real e em coisas que realmente vão gerar riquezas e benefícios para as pessoas, tornando a atividade económica não especulativa.

Estes bancos conectam os poupadores com os investidores que querem transformar o mundo em algo melhor com empresas sustentáveis. Por norma estes financiam empresas que agregam valor cultural e beneficiam pessoas e o meio ambiente[lem18].

Em 2009 foi fundada a aliança global de bancos com valores, esta conta com a participação de 65 instituições financeiras dos quais nenhum pertence a Portugal. Servindo mais de 70 Milhões de clientes e para além de 210 Biliões de Dólares americanos[fBoV21b].

Figura B.1: Mapa de membros da aliança global de bancos com valores [fBoV21a].

Podemos observar no mapa da figura B.1 a localização dos bancos éticos pertencentes a esta aliança, sendo estes dados de janeiro de 2021.

B.2 Deveres das Instituições Financeiras

Todas as instituições devem seguir o regime geral das instituições de crédito e sociedades financeiras, segundo o Decreto-Lei n.º298/92. Este regime entrou em vigor no dia 1 de janeiro de 1993.

Neste Decreto-Lei:

> ”prevê-se um conjunto de regras de conduta que devem guiar a actuação das instituições de crédito, seus administradores e empregados nas relações com os clientes. Enquanto no capítulo I são definidos os deveres gerais da conduta a observar pelas instituições de crédito e seus representantes, nos capítulos seguintes referem-se grupos específicos de normas de conduta, designadamente as relacionadas com o segredo profissional, defesa da concorrência e publicidade.
>
> A preocupação de fazer assentar cada vez mais a actuação das instituições de crédito e outras empresas financeiras em princípios de ética profissional e regras que protejam de forma eficaz a posição do «consumidor» de serviços financeiros não se manifesta apenas pela consagração expressa dos apontados deveres gerais de conduta e das demais normas referidas, mas explica ainda o incentivo que se pretende dar à elaboração de códigos deontológicos de conduta pelas associações representativas das entidades interessadas (artigo 77.º,

n.os 2 a 4). Desta forma, a orientação que já constado Código do Mercado de Valores Mobiliários, confinada aí às atividades de intermediação de valores mobiliários, é alargada às restantes atividades desenvolvidas pelas instituições de crédito e demais empresas financeiras." Página 2 do Decreto-Lei nº298/92 [Por92].

Alguns dos artigos mais importantes neste decreto-lei no que conta a deveres são:

- Artigo 74.º - "Outros deveres de conduta";
- Artigo 77.º - "Dever de informação e de assistência";
- Artigo 77.º-C - "Publicidade";
- Artigo 77.º-E - "Deveres especiais na comercialização ao retalho de produtos e instrumentos financeiros pelas instituições de crédito";
- Artigo 78.º - "Dever de segredo";
- Artigo 79.º - "Exceções ao dever de segredo";
- Artigo 80.º - "Dever de segredo do Banco de Portugal";
- Artigo 81.º-A - "Base de dados de contas";
- Artigo 84.º - "Violação do dever de segredo";
- Artigo 90.º-A - "Registos e arquivo" - primeiro ponto;
- Artigo 115.º-C - "Comunicação e divulgação da política de remuneração" - terceiro ponto alineas a) e b);
- Artigo 115.º-I - "Dever de divulgação no sítio na Internet";
- Artigo 116.º-M - "Deveres de comunicação de informação para elaboração dos planos de resolução";
- Artigo 116.º-N - "Dispensa parcial do dever de comunicação de informação para elaboração dos planos de resolução";
- Artigo 116.º-Z - "Dever de comunicação";
- Artigo 118.º-A - "Dever de abstenção e registo de operações";
- Artigo 119.º - "Dever de accionista";
- Artigo 120.º - "Deveres de informação";
- Artigo 123.º - "Deveres das instituições autorizadas em outros Estados-Membros da União Europeia";
- Artigo 145.º-A - "Designação de administradores provisórios" - sétimo ponto;
- Artigo 145.º-AP - "Deveres gerais das instituições de crédito objeto de resolução";

- Artigo 157.º - ”Dever de informação”;
- Artigo 207.º - ”Injunções e cumprimento do dever violado”;
- Artigo 210.º - ”Coimas”;
- Artigo 218.º - ”Deveres de testemunhas e peritos” [Por92].

B.3 Sigilo Profissional / Bancário

A partir de 2017 o fisco passou a receber informações dos bancos anualmente sobre saldos de contas. Isto visa ao fim do sigilo bancário, contudo compromete dados pessoais e até mesmo a privacidade dos cidadãos.

Visto que as administrações tributárias têm vindo a ganhar poder no conhecimento de dados pessoais o debate entre o equilíbrio entre transparência fiscal e proteção da vida privada foi aberto.

Como visto na secção anterior, no artigo 78.º, o segredo bancário é previsto por lei. Mais concretamente:

> ”1—Os membros dos órgãos de administração ou de fiscalização das instituições de crédito, os seus empregados, mandatários, comitidos e outras pessoas que lhes prestem serviços a título permanente ou ocasional não podem revelar ou utilizar informações sobre factos ou elementos respeitantes à vida da instituição ou às relações desta com os seus clientes cujo conhecimento lhes advenha exclusivamente do exercício das suas funções ou da prestação dos seus serviços.
>
> 2—Estão, designadamente, sujeitos a segredo os nomes dos clientes, as contas de depósito e seus movimentos e outras operações bancárias.
>
> 3—O dever de segredo não cessa com o termo das funções ou serviços.” [Por92].

Ou ainda no artigo 80.º :

> ”1—As pessoas que exerçam ou tenham exercido funções no Banco de Portugal, bem como as que lhe prestem ou tenham prestado serviços a título permanente ou ocasional, ficam sujeitas a dever de segredo sobre factos cujo conhecimento lhes advenha exclusivamente do exercício dessas funções ou da prestação desses serviços e não poderão divulgar nem utilizar as informações obtidas.
>
> 2—Os factos e elementos cobertos pelo dever de segredo só podem ser revelados mediante autorização do interessado, transmitida pelo Banco de Portugal, ou nos termos previstos na lei penal e de processo penal.” [Por92].

Isto é, os trabalhadores nestas instituições não podem revelar informações sobre a relação dos clientes com o banco. Estes têm de manter sigilo sobre: nomes dos clientes, contas de depósito, movimentos e operações bancárias dos clientes.

O que mudou em 2017 foi o conhecimento do fisco sobre os saldos e rendimentos das contas, contudo mantendo estes dados confidenciais não tendo os funcionários da Autoridade Tributária a obrigação de sigilo profissional. Estas medidas foram tomadas com o intuito de diminuir as infrações fiscais, partindo pela existência de acréscimos de património não justificados[Cri16].

As sanções do incumprimento dos artigos referidos também estão descritas no mesmo documento no artigo 84.º:

> ”Quem, sem consentimento, revelar segredo alheio de que tenha tomado conhecimento em razão do seu estado, ofício, emprego, profissão ou arte é punido com pena de prisão até 1 ano ou com pena de multa até 240 dias.” [Por92].

Para além da sanção penal prevista este incumprimento é também qualificado como contraordenação. Que segundo o artigo 210.º, alínea i) do mesmo documento punível com coima de 748,20 a 748.200 euros se aplicável a ente coletivo, ou de 249,40 a 249.400 euros se aplicável a pessoa singular [Por92].

Ainda conforme previsto no Acordo Coletivo de Trabalho para o Sector Bancário na clausula 24, nº 1, alínea c), poderá haver sanções disciplinar a trabalhadores bancários por dever de guardar sigilo profissional [Pa□02].

A violação destes segredos é um crime semi-público, como referido no artigo 198.º do mesmo artigo o procedimento criminal depende de queixa ou de participação.

B.4 Conclusão

A ética é um tema sempre pertinente independentemente do ramo, neste caso foi estudada um pouco na sua relação com o ramo financeiro e bancário de forma a ter uma ideia dos deveres das instituições financeiras e para que haver um sigilo.

As novas gerações são aquelas que vão cuidar do planeta Terra, por isso, é pertinente encontrar formas de ajudar e o tornar mais sustentável. Posto isto, um Banco Ético é uma solução para o futuro e uma nova perspetiva no mundo bancário e financeiro.

Apêndice C

Testes Automatizados

O documento seguinte foi criado com o intuito de estudar as diversas variáveis de testes automatizados com a finalidade de ir ao encontro das competências necessárias para realizar os testes automatizados no projeto.

Este documento encontra-se devidamente estruturado e identificado como também foi revisado pelo orientador António Faneca.

Testes Automatizados

Mariana Magalhães Dantas
Departamento de Informática
Universidade da Beira Interior, Covilhã, Portugal
Email: mariana.dantas@ubi.pt

***Abstract*—This document aims to study different automatized tests, describing what they are, different types of test that can be automatized and different tools to automatize tests. It is important to state the advantages and disadvantages of automatized tests and how they differ from the manual tests. In this report the different approaches of outside-in and inside-out testing are also studied, with the addition of definitions of defect, error, failure and Arrange, Act, Assert. Given that the objective of this report is also the study of which of this tests are used on the app M24, there is a section dedicated to the exposition of the process of testing for the app.**

***Index Terms*—Aceitação, Appium, Automatização, Carga, Cucumber, Desempenho, Inside-out, Integração, Outside-in, Regressão, Robotium, Selenium, Sistema, TestComplete, Teste, Unidade.**

I. Introdução

– "There's a way to do it better - find it."

Thomas A. Edison, inventor.

O teste de software é definido como uma atividade para verificar se o resultado real corresponde aos resultados esperados e para garantir que o sistema de software está livre de defeitos. O objetivo principal do teste pode ser garantia de qualidade, estimativa de confiabilidade, validação e verificação. O teste pode ser feito manualmente ou por meio de ferramentas automatizadas, podendo ser dividido em *White Box, Black Box* ou *Grey Box*. Bugs de software podem potencialmente causar perdas, tanto monetárias quanto humanas. É por isso que os testes são tão importantes.

Neste relatório serão estudadas diversas variáveis de testes automatizados, no intuito de ir ao encontro com as competências necessárias para realizar testes no estágio curricular na empresa ITSector na área de Android. Foi proposto pelo orientador de estágio António Faneca no mês de Dezembro de 2020 de forma a complementar o relatório de estágio correspondente ao segundo semestre do segundo ano do Mestrado em Engenharia Informática da Universidade da Beira Interior.

Este documento tem como objetivo a exposição de diferentes testes automatizados, para atingir este objetivo foram criados sub-objetivos:

- Definição do que é um teste automatizado;
- Vantagens e Desvantagens da utilização de testes automatizados;
- Comparação de um teste manual a um teste automatizado;
- Estudo das abordagens Outside-in e Inside-out Testing;
- Estudo de testes de regressão, carga, desempenho, integração, sistema, unidade e aceitação;
- Analise do padrão AAA;
- Definição de BDD;
- Definição de defeito, erro e falha;
- Estudo de ferramentas de testes automatizados;
- Exposição dos testes e técnicas utilizadas no contexto da aplicação M24.

II. Testes Automatizados

A automatização de testes é uma técnica de teste de software com o intuito de comparar o resultado com o resultado esperado. Esta comparação pode ser alcançada com a criação de *scripts* de teste ou usando ferramentas de automatização de testes.

Por norma a automatização de testes é usada para automatizar tarefas repetitivas e tarefas de teste que são difíceis de executar manualmente[1].

Posto isto, testes automatizados utilizam ferramentas de automatização de testes para controlar a execução dos testes de software, através da aplicação de estratégias onde o objetivo é criar um *script* ou software que testem o programa pelo utilizador[2].

A. Vantagens

A utilização destas ferramentas podem trazer vários benefícios, tais como:

- **Retorno do investimento –** também conhecido por *Return On Investment (ROI)*, está provado que a longo prazo as empresas que realizam testes automatizados têm o retorno do seu investimento inicial, na secção II-C esta vantagem encontra-se melhor exposta;
- **Reutilização –** vários testes podem ser feitos com a reutilização de um *script*;
- **Prevenção de bugs –** com estes métodos é possível encontrar bugs num estado inicial do software, o que a longo prazo poupa tempo e dinheiro;
- **Confiabilidade –** um teste automatizado é mais rápido e confiável ao executar testes padronizados pois repetem sempre os mesmos passos que podem causar erros se forem ignorados;
- **Simultaneidade –** podem ser testados diversos dispositivos simultaneamente, ajudando também na comparação de velocidade de cada dispositivo a executar o mesmo *script*;

- **Cobertura –** há uma variedade de testes que podem ser feitos automaticamente: testes unitários, regressão, integração, carga, stress, entre outros. O que ajuda a testar uma maior cobertura do que manualmente, no que conta as capacidades da aplicação, dos serviços e da infraestrutura operacional;
- **Volume –** é possível executar testes em milhares de dispositivos, que comparativamente com testes manuais seria impossível em tempo real e seriam usados muitos recursos;
- ***Feedback* –** é possível ter *feedback* do teste continuamente e rapidamente, ver em que passo falha ou até obter um gráfico/relatório onde é fácil analisar os resultados da execução do teste.

B. *Desvantagens*

Apesar de terem grandes vantagens para serem utilizados, os testes automatizados também tem as suas desvantagens. É mais difícil encontrar problemas muitos específicos que utilizadores possam ter.

Este tipo de testes não testa a usabilidade efetiva do design. Por exemplo, se a posição dos botões é confortável para o utilizador, ou se a aplicação é de fácil uso ou até intuitiva.

Na execução de cada *script*, apenas estão a ser testados alguns cenários e não todos, cabe ao programador repartir o teste por passos e discriminar o que cada passo faz nesse teste.

Um teste automatizado é apenas para testar um fluxo simples, por exemplo, fazer *login*, criar uma nova conta, enviar um email.

As aplicações devem ser testadas manualmente, pois há erros que um computador ou software não vão encontrar se não tiver a experiência humana de utilização [3].

C. *Testes Manuais vs Testes Automatizados*

Um teste automatizado, como já referido, possui alta escalabilidade e velocidade de execução, sendo então adequado para testes de *back-end*, investigação, regressão e unitários além de possuir esforço adicional reduzido.

Por outro lado os testes manuais trazem a visão real dos utilizadores, contudo pode ser lento e o seu esforço é exponencial à escala do teste.

Um dos pontos fulcrais e de divergência entre testes manuais e testes automatizados é o retorno de investimento inicial (ROI). De forma a expor esta divergência temos a figura 1, em que à medida que são feitos mais ciclos de desenvolvimento do software os testes manuais apresentam despesas exponenciais, enquanto que os testes automatizados quando chega ao quarto ou quinto ciclo de desenvolvimento as suas despesas quase se mantém iguais.

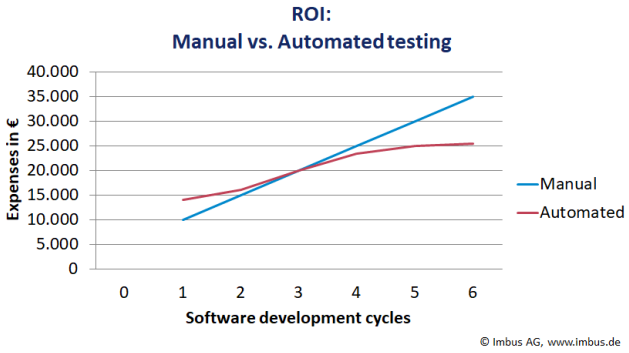


Figure 1. Testes Manuais vs Testes Automatizados - Retorno de investimento.

Acrescentando ainda que apesar das despesas serem maiores para os testes automatizados inicialmente, a partir do ciclo três isso deixa de ser verdade, passando os testes manuais a serem mais dispendiosos. Provando assim a primeira vantagem apresentada para a utilização de testes automatizados. Este gráfico foi criado pela empresa Imbus AG [4].

É necessário então examinar o tipo de software que vai ser testado e qual o teste mais indicado para este ou parte deste.

Avalizando então os tipos de testes, os mais indicados para serem testes manuais são:

- Testes exploratórios;
- Testes de usabilidade;
- Testes *Ad-hoc*.

Em relação aos testes mais indicados para serem testes automatizados, são:

- Testes de regressão;
- Testes de carga;
- Testes de desempenho.

Porem há tipos de teste que podem ser feitos por ambas as abordagens, estes sendo:

- Testes de integração;
- Testes do sistema;
- Testes de unidade;
- Testes de aceitação.[5]

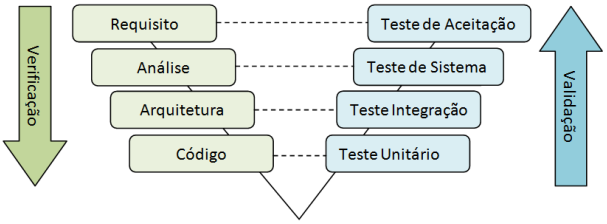


Figure 2. Testes de ambas as abordagens.

Na figura 2 podemos observar quais as componentes que são verificadas para cada teste e qual o sentido de validação, quais os elementos a verificar primeiro e quais os testes validar primeiro.

Neste relatório apenas serão abordados os testes automatizados, deste modo, os testes listados como apenas manuais não serão estudados.

D. Outside-in vs. Inside-out Testing

Existem duas abordagens quando se trata de testar, pode ser de fora para dentro (*Outside-in*) ou de dentro para fora (*Inside-out*).

Na abordagem de *outside-in*, começa-se por se focar na perspectiva do utilizador final e tentam descrever a funcionalidade desejada de alto nível e os objetivos do software em teste na forma de *user stories*.

Em cada iteração ou *sprint*, as *user stories* são refinadas até que a equipa de testers e o Dono do Produto / Representante do Cliente possam concordar com os critérios de aceitação, que determinam que uma *user stories* funcione conforme planeado.

O teste então vai 'para dentro' e o código é escrito para testar componentes cada vez menores até que alcance o nível de teste de unidade.

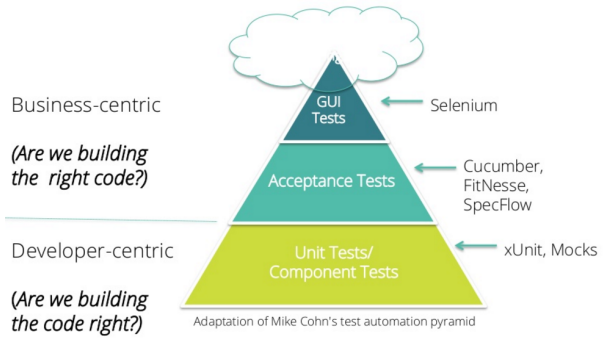


Figure 3. Pirâmide de Automação de testes.

Na abordagem de dentro para fora ou de baixo para cima, a equipa começam com testes de unidade no nível mais baixo da pirâmide de automação de teste (figura 3). Conforme o código evolui devido à refatoração, os esforços de teste evoluem e a equipa avança para o teste de nível de aceitação, que testa a lógica de negócios no nível de API ou serviço. O topo da pirâmide e a última coisa testada é a interface do utilizador (IU).

De dentro para fora e de fora para dentro são abordagens diferentes, mas complementares aos testes. O controle de qualidade de software depende das noções relacionadas de verificação e validação (V&V) (figura 2) que verificam se um sistema de software atende às especificações e que atende a sua finalidade pretendida. Os termos verificação e validação são frequentemente usados alternadamente, mas têm significados diferentes[6].

III. Testes de Regressão

Segundo *Pressman*, testes de regressão é a re-execução de algum subconjunto de testes que já foram conduzidos para garantir que as modificações não propagaram efeitos colaterais indesejados[7].

Este tipo de teste ocorre sempre que o sistema recebe alguma alteração. Seja a alteração de uma funcionalidade ou a adição de uma funcionalidade[8]. Isto previne que funcionalidades já existentes deixem de funcionar devido à adição de uma nova funcionalidade ou até mesmo de alterações numa funcionalidade que funcionava previamente ás alterações.

Estes testes podem ser de cinco tipos:

- Total;
- Baseado em casos de uso de maior risco;
- Por perfil;
- Dos segmentos modificados;
- Com *firewall*.[8]

Para que possam ser realizados é necessário ter criado uma suite de testes de regressão, as quais armazenam os casos de teste que devem ser executados.[7] Para além de que devem ser realizados testes de regressão sempre que uma nova versão da aplicação é feita.

É oportuno criar uma lista de testes que possuem maior risco, sendo estes que tem a maior possibilidade de revelar a presença de falhas. Essa lista é oportuna para que os erros sejam revelados mais cedo e as funcionalidades alteradas o mais rápido possível.

Testes de regressão são testes do tipo *Black Box* por serem testes onde as funcionalidades são testadas sem ter qualquer conhecimento da estrutura interna do código, da sua implementação ou *paths*. [9]

Estes testes podem ser feitos usando ferramentas como: *test link*, *selenium* ou *mantis bug traker*.[8]

IV. Testes de Carga

Testes de carga também conhecidos por *Load Test* são por norma realizados com o intuito de responder à questão: "Quantas transações serão suportadas por minuto quando aumentarmos os utilizadores simultâneos para 2.000, 3.000, 4.000?" [10].

Testes de carga identificam a capacidade máxima da aplicação, determinam se a infraestrutura atual é suficiente para executar o software, indica qual a sustentabilidade da aplicação em relação ao pico de carga do utilizador e aponta o número de utilizadores que a aplicação pode ter simultaneamente de forma a que cada um destes possam aceder às funcionalidades.

Estes são importantes por simular cenários onde a aplicação e os serviços são levados aos limites, pois a aplicação pode ter boa performance para um utilizador e não ter quando estão centenas ou milhares a utilizar estes serviços.

As aplicações devem ser testadas com este tipo de testes quando há alterações ao código pois estas podem ter alterado a forma que a aplicação suporta a carga de pedidos e respostas.

Automatizar este tipo de testes é importante pois estar a empregar milhares de *testers* para aceder à aplicação como teste de carga não é conveniente[11].

V. Testes de Desempenho

Os testes de desempenho visam a responder a questão: "A aplicação suporta 1.000 transações por minuto com 1.000 utilizadores simultâneos?" [10]. Sendo muitas vezes os testes de carga considerados testes de desempenho por serem testes do mesmo tipo contudo os de carga serem escalados.

Tendo isto em mente podemos ter cinco tipos de testes de desempenho:

- *Load testing* - testes de carga, como já estudados, servem para verificar a capacidade da aplicação sob cargas de utilizador previstas. Sendo o objetivo identificar *performance bottlenecks* antes da aplicação ser executada.
- *Stress testing* - testes de stress envolvem o teste de uma aplicação sob cargas de trabalho extremas para ver como esta lida com alto tráfego ou processamento de dados. O objetivo é identificar o ponto de ruptura de uma aplicação.
- *Endurance testing* - testes de resistência são feitos para garantir que o software pode lidar com a carga esperada por um longo período de tempo.
- *Spike testing* - testes de pico testam a reação do software a grandes picos repentinos de carga gerada pelos utilizadores
- *Volume testing* - testes de volume os dados são preenchidos por uma base de dados e o comportamento do sistema é controlado. Sendo o objetivo testar o software com diferentes números de dados.
- *Scalability testing* - testes de escalabilidade têm como objetivo determinar a eficácia do software quando houver uma ampliação para suportar um maior número de utilizadores. Ajuda a planear a adição de capacidade ao sistema de software.

Com tais características, os testes de desempenho têm como objetivo geral de verificar o desempenho do software segundo a velocidade, a escalabilidade e a estabilidade[12].

VI. Testes de integração

Testes de integração têm como objetivo encontrar falhas de integração entre as unidades que constituem o software. Nesta fase as categorias de testes são: testes de interface e testes de dependências[13].

Nestes são observadas requisições HTTP (por exemplo), e é verificado o resultado de uma requisição completa, analisando o formato de resposta, o código de status na resposta HTTP, o formato de dados e validações [14].

Estes testes são realizados após a validação dos componentes do sistema realizados em testes de unidade (secção VIII). Estes componentes podem estar já em produção, reutilizados e adaptados ou novos componentes. São testes que devem ser executados pela equipa de desenvolvimento pois exigem conhecimento da estrutura do programa.

Existem três estratégias para realizar estes testes:

- *Top-down* - as unidades são integradas de cima para baixo;
- *Bottom-up* - a integração é feita a partir de baixo para cima da hierarquia de componentes;
- *Big-Bang* - integração dos módulos feita toda ao mesmo tempo.[15]

VII. Testes do Sistema

Testes de sistema são executados para testar o sistema como um todo, ou seja, todos os módulos e componentes são integrados por ordem para verificar se o sistema funciona como esperado (envolvendo componentes de outros software e/ou hardware). Como vimos anteriormente na figura 2 estes testes são validados após os testes de integração. São testes considerados *black-box*[14].

O processo de teste de um sistema integrado de hardware e software serve para verificar se o sistema atende aos requisitos especificados.

É importante realizar um ciclo completo de teste e este é o estado da aplicação mais indicado para o fazer, por serem realizados num ambiente mais semelhante ao ambiente de produção pode ser visto como o utilizador reage a certas respostas da aplicação. É também nestes testes que são testados os requisitos de arquitetura da aplicação e os de negócios[16].

Há vários tipos de teste de sistema, na figura seguinte (fig. 4) podemos ver uma listagem destes seguidamente por uma pequena explicação de cada um.

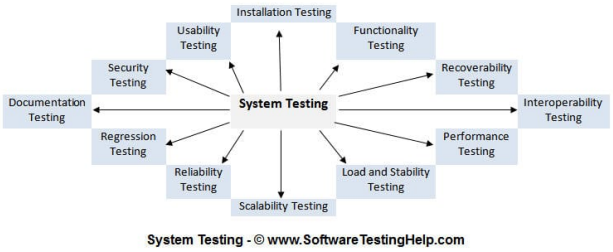


Figure 4. Tipos de Teste de sistema.

- **Teste de Funcionalidade** - Para garantir que a funcionalidade do produto está de acordo com os requisitos definidos, dentro das capacidades do sistema.
- **Teste de Recuperabilidade** - Para certificar-se de quão bem o sistema recupera de vários erros de entrada e outras situações de falha.
- **Teste de interoperabilidade** - Para ter certeza se o sistema opera bem com produtos de terceiros ou não.
- **Teste de desempenho** - Para garantir o desempenho do sistema nas várias condições, em termos de características de desempenho.
- **Teste de escalabilidade** - Garantir a capacidade de escalonamento do sistema em vários termos, como escalonamento de utilizador, escalonamento geográfico e escalonamento de recursos.
- **Teste de confiabilidade** - Para certificar-se de que o sistema pode ser operado por um período mais longo sem desenvolver falhas.
- **Teste de regressão** - Para garantir a estabilidade do sistema conforme ele passa por uma integração de diferentes subsistemas e tarefas de manutenção.

- **Teste de documentação** - para garantir que o guia do utilizador do sistema e outros documentos de tópicos de ajuda estejam corretos e utilizáveis.
- **Teste de segurança** - Para certificar-se de que o sistema não permite acesso não autorizado a dados e recursos[16].

Chegando a conclusão que é imprescindível realizar este teste por desempenhar um papel tão significativo na entrega de um produto de qualidade ao cliente.

VIII. Testes de unidade

Testes de unidade testam funções simples e de resultado constante. Sendo a unidade a menor parte testável de um sistema por ser algo indivisível [14], todas elas são testadas isoladamente para determinar se cada uma delas realiza o que foi especificado. Nesta fase as categorias de teste aplicáveis são: teste de estrutura interna, de funcionalidade e de segurança[13].

Estes testes como quaisquer testes automatizados não servem apenas para verificar se uma função especifica está a funcionar mas para garantir que a sua aplicação continua a funcionar após a alteração do código[17].

Estes testes são feitos com base todos os inputs, erros e outputs possíveis.

A. Padrão AAA

O padrão AAA (*Arrange, Act, Assert*) ajuda a organizar o código de teste, dividindo os testes nas seguintes secções:

- Organizar (*Arrange*) - num teste de unidade inicializa objetos e define o valor dos dados que são passados para o caso de teste;
- Agir (*Act*) - invoca o caso de teste com os parâmetros organizados;
- Declarar (*Assert*) - verifica se o caso de teste se comporta conforme o esperado.[18]

B. Behavior-Driven Development (BDD)

O BDD é uma metodologia ágil de segunda geração, externa, baseada em *pull*, de múltiplas partes interessadas, em escala múltipla, de alta automação. Ele descreve um ciclo de interações com saídas bem definidas, resultando na entrega de software funcional e testado que importa.

BDD usa sintaxe para descrever três estados de teste que são aproximadamente equivalentes aos do padrão AAA: *Given = Arrange; When = Act; Then = Assert*. [6]

IX. Testes de Aceitação

O teste de aceitação é o ultimo teste feito antes da implementação do software, tendo como objetivo verificar se o software está pronto e pode ser utilizado pelos utilizadores, para desempenhar as funções / tarefas para que o software foi construído.

Há três estratégias para este tipo de teste:

- **Aceitação formal** - Os testes são planeados e projetados com o mesmo cuidado e no mesmo detalhe que o teste do sistema. Os casos de teste escolhidos devem ser um subconjunto dos que foram realizados no teste do sistema;
- **Aceitação Informal ou teste Alfa** - os procedimentos para executar o teste não são definidos com tanto rigor como no teste de aceitação formal. As funções e as atividades de negócios a serem exploradas são identificadas e documentadas, mas não há casos de teste específicos para seguir;
- **Teste Beta** - é o menos controlado das três estratégias de teste de aceitação. No teste beta, a quantidade de detalhes, os dados e a abordagem adotadas são de inteira escolha do *tester* individual.[19]

A. Defeito, Erro ou Falha

Conceitos como defeito, erro ou falha podem soar parecidos mas dentro dos testes de software estes têm diferenças:

- **Defeito** – Representa um ato inconsistente realizado por um indivíduo ao tentar compreender uma informação, pode ser uma instrução ou um comando incorreto.
- **Erro** - É um defeito encontrado numa parte do software, diferença entre um valor obtido e um valor esperado, ou seja, um estado intermediário incorreto ou resultado inesperado na execução de um programa.
- **Falha** - Comportamento do software diferente do comportamento esperado pelo utilizador final[20].

Figure 5. Defeito, erro ou falha.[20]

Estas diferenças são importantes ter em mente por poderem despoletar diferentes acções sobre um software, por vezes é necessário rever os objetivos e o que o cliente pretende, como também, atualizar cargas de trabalho para os diferentes actores no software.

X. Ferramentas de Automatização de testes

A automação de testes é, em última instância, a execução de um software para testar outros softwares. É muito comum ocorrerem atrasos em projetos de automação em virtude de problemas causados pela ferramenta de automação de testes. [21] Por esse motivo é importante a escolha desta ferramenta de forma a não criar mais inconveniente e de ser de fácil e rápida execução.

Seguem-se então algumas das ferramentas mais utilizadas e recomendadas para a automação de testes.

A. Selenium

Selenium é a ferramenta *open source* mais utilizada hoje em dia para a automatização de testes. Sendo a sua maior adesão para testes de código aberto para aplicações Web.

Criada em 2000 esta ferramenta constitui a base de outras ferramentas de teste como o *Katalon Studio, Robot Framework, Protractor* e *Watir*.

Suporta ainda diversos ambientes operacionais como: Windows, Mac e Linux, além das várias opções de navegadores, como *Chrome, Firefox, Internet Explorer* e *Headless*.[22]

O Selenium IDE, sendo um *add-on* de navegador, permite gravar e reproduzir. Enquanto o Selenium *WebDriver* ajuda a criar *scripts* de automação de testes mais complexos e avançados.

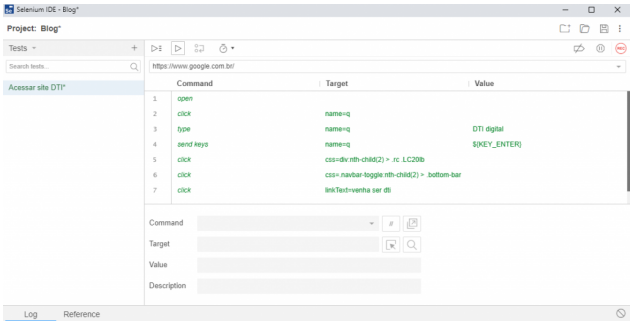


Figure 6. Selenium ferramenta de testes.

Esta ferramenta pode ser utilizada com diferentes linguagens como: Java, Perl, JavaScript, PHP, Python, C#, Ruby e Groovy.[23]

B. Appium

Appium é uma ferramenta de automação de aplicações nativas ou híbridas de código aberto, multi-plataforma, que oferece suporte para emuladores ou dispositivos moveis reais.

Tal como o Selenium, é possível realizar testes em diferentes linguagens tais como: Ruby, Java, Node.js, PHP, C#, Clojure e Perl.

Para alem do sistema Android o Appium também realiza testes de software da plataforma iOS. [22]

Esta ferramenta foi desenvolvida com base na ideia de que o teste de aplicações não deve exigir a inclusão de um SDK ou a recompilação da aplicação. Visa a automatizar qualquer aplicação móvel independente da linguagem ou estrutura de teste, com acesso total a APIs e base de dados. [24]

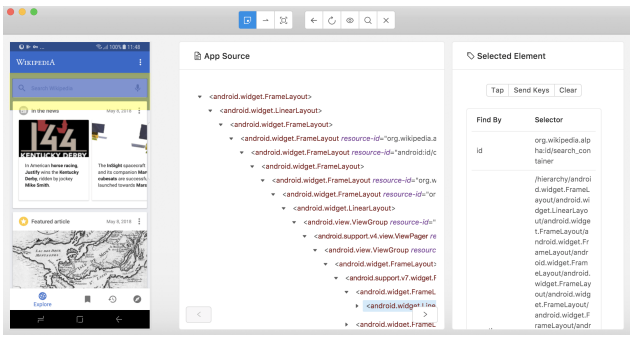


Figure 7. Appium ferramenta de testes direcionada a dispositivos moveis.

C. Visual Studio Test Professional

Visual Studio Test Professional (VST) é uma solução totalmente instrumentada, confiável e intuitiva paras as plataformas da Microsoft, isto incluindo dispositivos moveis como telemóveis e tablets como também desktops, servers e a nuvem [23].

Esta ferramenta permite obter acesso aos planos de teste do Azure, coordenar as atividade de gerar testes, como o planeamento, criação, execução e rastrear testes.

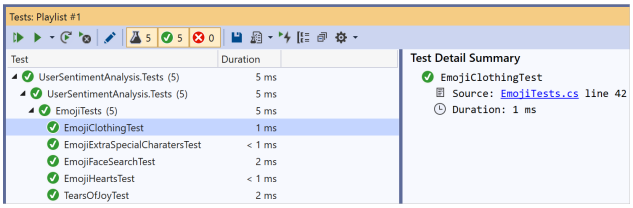


Figure 8. Output de resultados da ferramenta de testes Visual Studio test Professional.

D. Robotium

Robotium é uma framework popular para a automação de testes para Android, compatível com aplicações nativas e hibridas. Como tambem com Gradle, Ant e Maven e facilita a criação de testes de Black Box, como se observa na figura 9, não é necessário acesso ao código da aplicação mas sim à sua interface de utilizador (UI).

Esta ferramenta cria *delays* e *timings* automáticos e não é necessário escrever códigos ao navegar de uma atividade para outra [23].

A principal função do Robotium é simular os procedimentos que seriam executados pelo responsável da analise de testes [22].

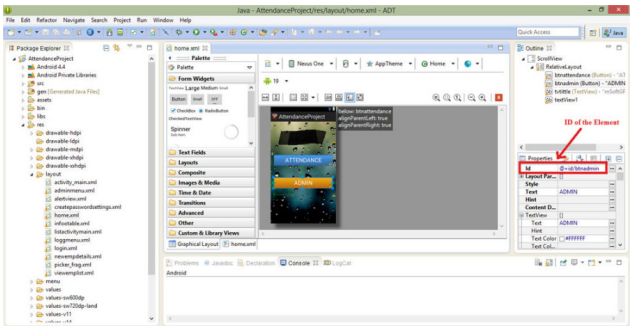


Figure 9. Robotium Android UI test.

E. Cucumber

Cucumber é uma ferramenta de testes open source desenvolvida com o conceito de desenvolvimento orientado por comportamento (BDD secção VIII-B). Anteriormente, era restrito apenas ao Ruby, mas atualmente suporta outras linguagens como Java, NET, Scala e Groovy [23].

É usado para testes automáticos de aceitação de aplicações web, suporta múltiplos sistemas operacionais e o seu código

pode ser executado em *frameworks* como Selenium e Ruby [23].

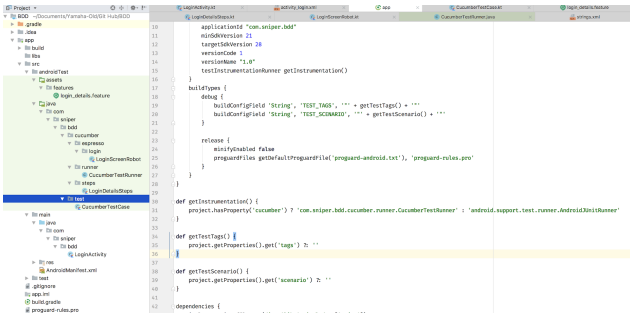


Figure 10. Cucumber.

F. TestComplete

Desenvolvido pela SmartBear Software, o TestComplete é uma plataforma comercial que pode automatizar testes do *desktop*, dispositivos moveis e aplicações web. Esta ferramenta pode ser utilizada com diferentes linguagens como JavaScript, Python, VBScript, JScript, DelphiScript, C++Script e C#Script, além de ter funções de teste orientado por dados, teste por palavras-chave, testes de regressão e teste distribuído.

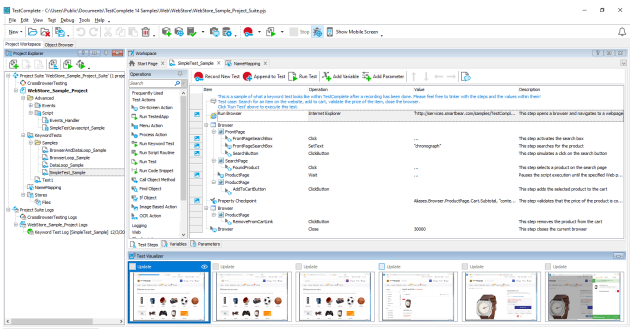


Figure 11. TestComplete.

O TestComplete possibilita a criação de testes automatizados para aplicativos iOS, Web Microsoft Windows e Android. Sejam estes com *scripts* ou de GUI é possível visualizar os testes, gravar e reproduzir os mesmos.

G. UFT - Unified Functional Testing

O *QuickTest Professional* (QTP) da empresa Hewlett-Packard (HP) agora conhecido como *Unified Functional Testing* (UFT) é a ferramenta mais conhecida quando se fala em teste funcional de software, fornecendo um grande leque de recursos para API, serviços web e testes de GUI. Sendo estes testes direcionados a web, mobile ou aplicações *desktop* o UFT utiliza o Visual Basic Scripting Edition para registar os processos de teste e controle [22].

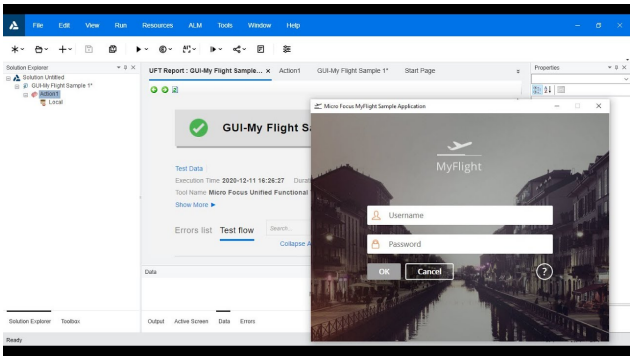


Figure 12. UFT.

Esta ferramenta proporciona testes funcionais automatizados e testes de regressão, podendo automatizar testes em Web, PowerBuilder, Desktop, ActiveX, SAP, Delphi, Net, Flex, Java, Oracle, Siebel, Mobile, PeopleSoft, Stingray e Visual Basic [23].

H. IBM Rational Functional Tester

IBM Rational Functional Tester é uma ferramenta de testes automáticos de regressão que fornece recursos de teste para testes funcionais, regressão, GUI e *data-driven* [25].

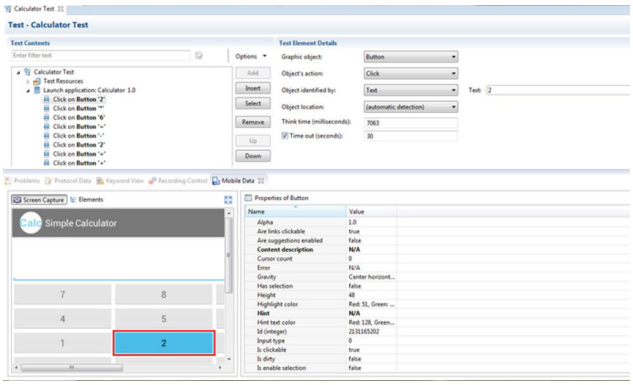


Figure 13. IBM Rational Functional Tester.

Esta ferramenta suporta aplicações como: baseados na Web, .Net, Java, Siebel, SAP, aplicações em terminal de emulador, PowerBuilder, Ajax, Adobe Flex, Dojo Toolkit, GEF, documentos Adobe PDF, zSeries, iSeries e pSeries [25].

XI. M24 - Testes Automatizados

M24 é uma aplicação que é testada ao longo da sua criação e com estes testes é possível averiguar se as suas funcionalidades são executadas como previsto e se os seus serviços estão devidamente incorporados.

A criação dos testes para esta aplicação é feita com a abordagem *outside-in*, sendo criados *user stories* para cada uma das funcionalidades que vão ser testadas utilizando a perspectiva do utilizador final como referencia. Através dessas *user stories* são criados os *scripts* na linguagem Java de forma a testar todas as componentes.

Estes *scripts* servem também para criar testes de regressão, de forma a confirmar que qualquer alteração feita à aplicação não vai entrar em conflito com uma funcionalidade já existente. Quando são feitos testes de regressão começa-se pelos casos de uso de maior risco ou pelos segmentos modificados e mais tarde faz-se um teste de regressão total no qual é testada a aplicação toda.

Quanto a testes de integração, a ITSector utiliza a ferramenta Appium e um mock server, sendo este um servidor de qualidade, para observar requisições e respostas HTTP aos serviços. Testes de sistema são também executados em conjunto com dispositivos reais de forma a ter a perspectiva final do utilizador.

Por fim são feitos testes de aceitação, sendo estes os últimos testes antes de passar a aplicação para produção, servem para confirmar que os requisitos do cliente foram atingidos e que nenhuma das falhas, defeitos ou erros passam para a fase de produção e entrem em contacto com o utilizador final.

É ainda importante de referir que todos os *scripts* feitos são criados para os sistemas Android e IOS de forma a que as diferenças da aplicação entre os dois sistemas sejam mínimas ou não existentes. Isto é possível através de utilização de diferentes variáveis que são alteradas conforme o sistema executado, o *script* do teste vai buscar estas variáveis conforme o sistema.

Todas as componentes sejam elas Android, IOS ou serviços da aplicação devem ser testados afincadamente de forma a obter o melhor produto final possível.

XII. Conclusão

O teste de software é imprescindível para a criação de um bom produto final. Cabe ao cliente, à empresa e ao programador/*tester* utilizar as melhores ferramentas e os testes mais eficientes e necessários para diminuir os riscos e os prejuízos monetários ou temporais.

A automatização de testes é então uma forma eficiente de testar qualquer que seja a aplicação, contudo não deixa de ser necessário a execução de testes finais manuais tendo em vista as vantagens e desvantagens destes.

Existem centenas de testes. Cada um serve para algo específico e segue um determinado padrão, como também existem centenas de ferramentas para os executar, cabe ao programador e ao cliente determinar qual o melhor teste e melhor ferramenta para testar a sua aplicação como também a quantidade de testes feitos.

Quanto à aplicação M24, concluo que os testes feitos e a forma que são executados são indispensáveis para um bom produto final. Sendo este produto uma junção de forças entre o desenvolvimento da aplicação, o teste desta e os requisitos funcionais.

References

[1] Software Testing Help, "What is automation testing (ultimate guide to start test automation)," https://www.softwaretestinghelp.com/automation-testing-tutorial-1/, accessed 21-12-2020.

[2] Richard Ferreira em Geek Blog, "As maiores vantagens de testes automatizados," https://blog.geekhunter.com.br/as-maiores-vantagens-de-testes-automatizados/#O_que_sao_testes_automatizados, accessed 21-12-2020.

[3] Chrystian Costa da Rosa em Medium, "A importância dos testes automatizados no desenvolvimento de software," https://medium.com/beelabacademy/a-importância-dos-testes-automatizados-no-desenvolvimento-de-software-cc8c5079ae7b, accessed 21-12-2020.

[4] Ralph van Roosmalen, "Test automation and return on investment (roi)," https://www.qfs.de/en/solutions/test-automation-basics/test-automation-roi.html, accessed 5-1-2021.

[5] Henrique Moura, Head de QA, na Inmetrics, "Teste manual vs. automação de teste: como e porque usar." https://inmetrics.com.br/teste-manual-vs-automacao-de-teste-como-e-porque-usar/, accessed 21-12-2020.

[6] Bobby Lalvani, "The role of unit tests in test automation," https://www.getzephyr.com/insights/role-unit-tests-test-automation, accessed 22-12-2020.

[7] Cristiano Baumgartner, "Testes de regressão! como, onde e quando utilizar?" https://www.testingcompany.com.br/blog/testes-de-regressao-como-onde-e-quando-utilizar/, accessed 21-12-2020.

[8] Cristian R. Silva, Software Engineer, "Testes de regressao automatizados," https://pt.slideshare.net/ocristian/testes-de-regressao-automatizados-40574077, accessed 21-12-2020.

[9] Guru99, "What is black box testing? techniques, example types," https://www.guru99.com/black-box-testing.html, accessed 21-12-2020.

[10] DevMedia, "Testes de desempenho, carga e stress," https://www.devmedia.com.br/testes-de-desempenho-carga-e-stress/26546, accessed 22-12-2020.

[11] Try Q A, "What is load testing in software testing? examples,how to do,importance, differences," http://tryqa.com/what-is-load-testing-in-software/#Why_is_load_testing_important, accessed 22-12-2020.

[12] Guru99, "Performance testing tutorial: What is, types, metrics example," https://www.guru99.com/performance-testing.html#3, accessed 22-12-2020.

[13] DevMedia, "Teste de integração na prática," https://www.devmedia.com.br/teste-de-integracao-na-pratica/31877, accessed 22-12-2020.

[14] Mateus Fernandes, "Teste de unidade e teste de integração: O que são?" https://medium.com/@mateus1198/teste-de-unidade-e-teste-de-integraç~ao-o-que-s~ao-de58d7a3d3d2, accessed 22-12-2020.

[15] Edson Lucas Albano, Eduardo Amaral, Fabiane Barreto Vavassori Benitti, "Byebug - aprendendo teste de software," https://www.inf.ufsc.br/~fabiane.benitti/byebug/objetos/OA31/integracao/presentation_html5.html, accessed 22-12-2020.

[16] Software Testing Help, "What is system testing – a ultimate beginner's guide," https://www.softwaretestinghelp.com/system-testing/, accessed 22-12-2020.

[17] Dayvson Lima, "Entenda de uma vez por todas o que são testes unitários, para que servem e como fazê-los," https://dayvsonlima.medium.com/entenda-de-uma-vez-por-todas-o-que-s~ao-testes-unitários-para-que-servem-e-como-fazê-los-2a6f645bab3, accessed 22-12-2020.

[18] Microsoft, "Noções básicas de teste de unidade," https://docs.microsoft.com/pt-br/visualstudio/test/unit-test-basics?view=vs-2019, accessed 22-12-2020.

[19] IBM Corp., "Conceito: Teste de aceitação," https://www.cin.ufpe.br/~gta/rup-vc/core.base_rup/guidances/concepts/acceptance_testing_12A0F152.html?nodeId=8fe84d80, accessed 22-12-2020.

[20] DevMedia, "Testes de aceitação: conhecendo e aplicando," https://www.devmedia.com.br/testes-de-aceitacao-conhecendo-e-aplicando/26959, accessed 22-12-2020.

[21] ——, "Automação de testes," https://www.devmedia.com.br/automacao-de-testes/10249#4, accessed 23-12-2020.

[22] Redação EVEO, "5 ferramentas que te ajudam na automação de testes," https://www.eveo.com.br/blog/ferramentas-automacao-testes/, accessed 23-12-2020.

[23] Prime Control, "10 ferramentas de automação de testes mais usadas," https://www.primecontrol.com.br/10-ferramentas-de-automacao-de-testes-mais-usadas/, accessed 23-12-2020.

[24] JS Foundation, "Appium," http://appium.io, accessed 27-12-2020.

[25] IBM, “Ibm rational functional tester,” https://www.ibm.com/products/rational-functional-tester, accessed 27-12-2020.