Universidade da Beira Interior
Departamento de Informática

Nº 4 - 2021: *ResTinder*

Elaborado por:

43760 - Manuel João Nogueira de Carvalho
43994 - Bruno Miguel Gonçalves Monteiro
44149 - Alexandre Salcedas Monteiro
44488 - Sara Maria da Silva Martins
44604 - Manuel Ferreira Magalhães

Orientador:

Professor Doutor Paulo André Pais Fazendeiro

19 de dezembro de 2021

Resumo

Tendo em conta que muitas vezes se torna complicado, em contextos de pares ou grupo, escolher um local para comer, surgiu a ideia de realizar uma aplicação que permitisse escolher com ligeireza o restaurante mais apropriado para todos. Através de uma execução ao estilo de uma aplicação de encontros, a `ResTinder` permite que várias pessoas possam escolher sem problemas onde comer.

Preferiu-se fazer uma aplicação bastante simples e direta, com base em *swipes* que permitem, com o código, fazer ligação entre as pessoas (em caso de *blind date*) e escolher um restaurante em comum. Optou-se por usar ferramentas que permitam o uso rápido da aplicação com a maior comodidade para os seus utilizadores.

Todos os objetivos que foram definidos, no enunciado e na preparação do projeto, foram atingidos com sucesso.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Descrição da Proposta

O principal objetivo do trabalho é, de acordo com o enunciado disponibilizado pelo professor, melhorar as *soft skills* de trabalho coletivo dos diversos elementos do grupo e aplicar e consolidar os conhecimentos obtidos nas aulas teóricas e práticas da unidade curricular de Programação de Dispositivos Móveis [1].

Neste caso particular, é pedido que se faça uma aplicação *mobile*, num estilo similar à aplicação `Tinder`®, que permita facilitar aos seus utilizadores a escolha de restaurantes para as suas refeições, em grupos de dois ou mais elementos.

1.2 Constituição do Grupo

O grupo é constituído apenas por alunos de licenciatura em Engenharia Informática, a saber:

- Manuel João Nogueira de Carvalho, número 43760;
- Bruno Miguel Gonçalves Monteiro, número 43994;
- Alexandre Salcedas Monteiro, número 44149;
- Sara Maria da Silva Martins, número 44488;
- Manuel Ferreira Magalhães, número 44604.

1.3 Organização do Documento

Este relatório está estruturado da seguinte forma:

- No capítulo 1, iniciado na página 5, descrevem-se, de modo muito geral, as bases do trabalho e a constituição do grupo;
- No capítulo 2, iniciado na página 7, descrevem-se as ferramentas e tecnologias utilizadas, bem como a engenharia de software inerente ao projeto;
- No capítulo 3, iniciado na página 12, descrevem-se os detalhes de implementação em termos de *design* e de código-fonte, bem como a forma de instalar e utilizar a aplicação;
- No capítulo 4, iniciado na página 16, descrevem-se os objetivos conseguidos e não conseguidos, a distribuição de tarefas pelos diversos elementos do grupo, problemas encontrados e reflexão crítica sobre os mesmos;
- No capítulo 5, iniciado na página 19, descrevem-se as principais conclusões obtidas na realização do trabalho e o trabalho futuro relacionado com o projeto.

Capítulo 2

Engenharia de Software

2.1 Introdução

Para que o trabalho cumpra todas as especificidades definidas no enunciado, é necessário planear *a priori* o modo de realização deste. As ferramentas e tecnologias ótimas para a execução deste projeto são especificadas na secção 2.2. Além disso, tendo em conta as bases teóricas adquiridas nas aulas da unidade curricular de Programação de Dispositivos Móveis [2] e de Engenharia de Software [3], nas secções 2.3, 2.4 e 2.5 encontra-se a definição de requisitos a que o programa deve responder, respetivos casos de uso e outros diagramas.

2.2 Ferramentas e Tecnologias Utilizadas

As ferramentas utilizadas no desenvolvimento deste projeto são:

- **Android Studio:** ambiente de desenvolvimento integrado para desenvolvimento de aplicações para a plataforma Android™ [4];
- **Google Firebase:** plataforma para criar aplicações *mobile* e *web*, no caso, utilizada para criação e manipulação de bases de dados de utilizadores [5] bem como elaboração do *backend*;
- **Google Places API:** *application programming interface* (API) que permite a pesquisa de locais do Google Maps [6];
- **Github:** repositório de código *online* que utiliza o sistema *Git*, um sistema de controlo de versões distribuído utilizado para desenvolvimento de software [7];

- **Microsoft Whiteboard:** quadro virtual que permite *brainstorming* de ideias e colaboração à distância, nomeadamente na criação de *layouts* [8];
- **Visual Paradigm:** ferramenta virtual para criação de diagramas de engenharia de software, no caso, diagramas de casos de uso e de classes [9].

2.3 Requisitos

Os requisitos funcionais de uma aplicação podem ser considerados como declarações detalhadas de serviços que o sistema deve providenciar, como o sistema deve reagir a determinados *inputs* e situações particulares. Opcionalmente, também se consideram declarações do que o que o sistema não deve fazer [3]. Neste projeto, os requisitos funcionais dizem que a aplicação deve:

1. Possuir uma *interface* de *login* para cada utilizador da aplicação `ResTinder`;
2. Ter capacidade para guardar credenciais de *login* de utilizador;
3. Ter uma opção de registo para novos utilizadores;
4. Possuir uma *interface* para alteração de definições pessoais de utilizador;
5. Possuir um sistema de *blind dates* (*matching* de restaurantes) para pessoas solteiras;
6. Possuir um sistema de *matching* de restaurantes para casais;
7. Possuir uma interface de *swipe* para escolha de restaurantes;
8. Ter capacidade para contactar quem deu *match* do mesmo restaurante;
9. Possuir um sistema de notificações para aviso de *matching*;
10. Ter uma opção de *logout* de conta;
11. Possuir uma interface de espera por um *match*;
12. Ter uma opção de cancelamento de *match*;
13. Ter uma opção de conclusão de *match*;

14. Possuir validação de *email* após registo;

15. Possuir um ecrã de *loading*.

Os requisitos não funcionais definem-se como restrições sobre os serviços oferecidos pelo sistema, como restrições de tempo ou no processo de desenvolvimento, aplicados na aplicação em geral [3]. Assim, os requisitos não funcionais deste projeto em particular declaram que a aplicação deve:

1. Não tolerar mais do que quatro falhas por minuto;

2. Ser segura para os seus utilizadores;

3. Ser capaz de encriptar dados dos utilizadores;

4. Ser capaz de guardar informação sobre os seus utilizadores;

5. Ter tempos de resposta rápidos após *inputs* dos utilizadores;

6. Garantir atualização em tempo real de dados.

2.4 Casos de Uso

Nos diagramas de casos de uso, cada caso representa uma tarefa discreta que envolve interação externa com o sistema. Os casos de uso relativos à aplicação `ResTinder` são os seguintes:

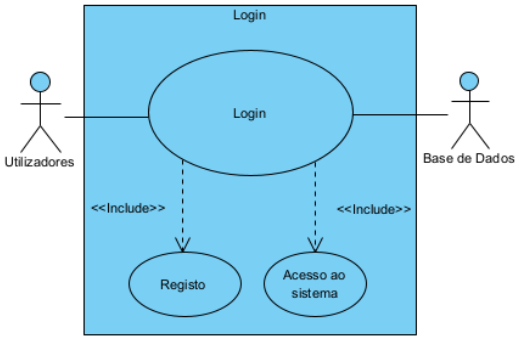

Figura 2.1: Diagrama de caso de uso relativo ao *login* da aplicação `ResTinder`.

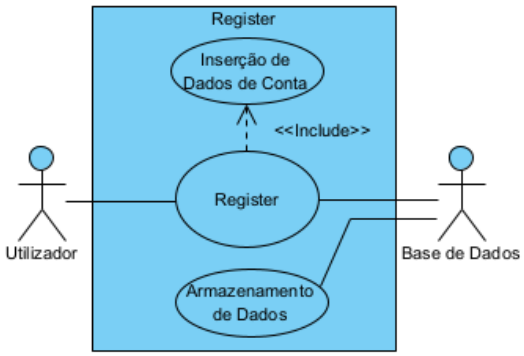

Figura 2.2: Diagrama de caso de uso relativo ao registo da aplicação `ResTinder`.

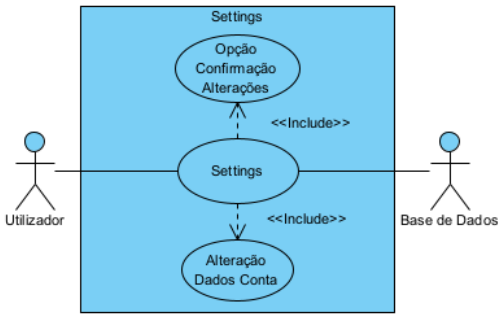

Figura 2.3: Diagrama de caso de uso relativo às definições de utilizador da aplicação `ResTinder`.

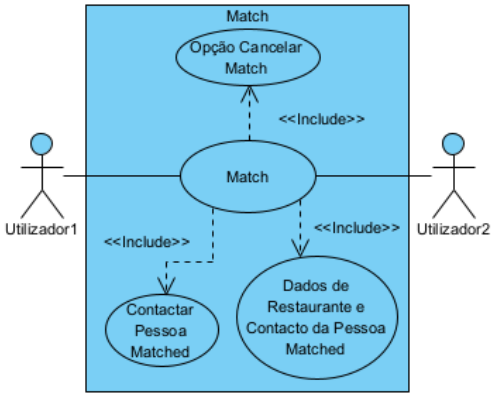

Figura 2.4: Diagrama de caso de uso relativo ao *match* da aplicação `ResTinder`.

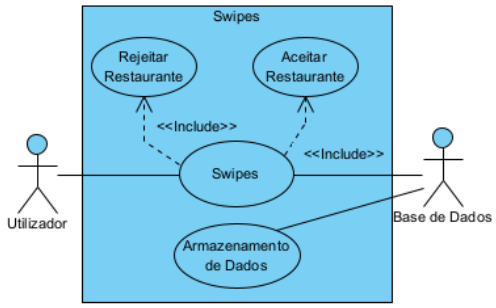

Figura 2.5: Diagrama de caso de uso relativo ao *swipe* da aplicação `ResTinder`.

2.5 Outros Diagramas

Os diagramas de classes mostram as classes de objetos presentes no sistema e as associações entre si. O diagrama de classes relativo à aplicação `ResTinder` é o seguinte:

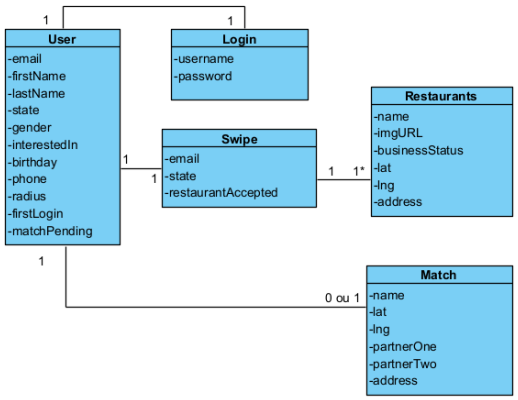

Figura 2.6: Diagrama de classes da aplicação `ResTinder`.

2.6 Conclusão

Os requisitos funcionais e não funcionais permitem perceber com facilidade todas as funcionalidades necessárias para a boa execução do programa. Aliados às ferramentas e tecnologias especificadas, é assim possível implementar a aplicação através de código em linguagem `Java`.

Capítulo 3

Implementação

3.1 Introdução

A implementação do programa é feita em linguagem `Java`, tendo em conta as bases teóricas e práticas obtidas nas aulas da unidade curricular e os requisitos funcionais e não funcionais definidos no capítulo 2. Assim, na secção 3.2 especificam-se as interpretações do enunciado, na secção 3.3 e 3.4 detalha-se a forma de implementação da aplicação, e nas secções 3.5 e 3.6 explica-se a forma de instalação e utilização da aplicação, respetivamente.

3.2 Escolhas de Implementação

Escolheu-se utilizar a *Firebase* para poder ter uma base de dados a atualizar em tempo real, de modo a possuir armazenamento em *cloud*, possibilitando acesso múltiplo aos dados em *queries* concorrentes.

Para a apresentação dos restaurantes, escolheu-se a `Places API` da Google, de modo a fornecer informações específicas sobre os restaurantes de acordo com a localização atual do dispositivo móvel.

Para atualizar a base de dados em tempo real, optou-se por utilizar código em linguagem `JavaScript`.

3.3 *Layout*

Os ficheiros de *layout* estão escritos em `xml`, com o código integrado em `Linear Layout`'s que lhes permite a responsividade independente dos dispositivos em que a aplicação está a ser executada. Foram feitos sete *layouts*, seis dos quais estão especificados nas secções 3.3.1 a 3.3.6.

3.3.1 `activity_main.xml`

Numa primeira instância, elaborou-se o *design* do *layout* do *login* da aplicação. O utilizador será deparado com este *layout* pela primeira vez que inicia a aplicação. Dispõe de campos `EditText` para inserção de credenciais como o *email* e a palavra passe da sua conta. Possui também uma `CheckBox` que permite guardar credenciais do utilizador. Existem duas instâncias de `Button` onde o utilizador pode-se registar ou efetuar o *login* na sua conta. Também é disponibilizado o logótipo da aplicação por via de uma `TextView`.

3.3.2 `home.xml`

No `home.xml` existem diversos componentes, nomeadamente:

- `CardStackView|item_card.xml`: Onde será apresentado os diversos restaurantes para o utilizador escolher. Esta *CardStackView* é importada de uma biblioteca para o *gradle*. O conteúdo do `CardStackView` é obtido através do `item_card.xml` via código;
- Duas `TextView`: Onde estão representadas as direções de swipe: *seta vermelha* para negar o swipe e *seta verde* para aceitar o swipe;
- Quatro `Button` (partilhados também com os *layouts* descritos nas secções 3.3.4 e 3.3.6):
  - `Logout:` Onde o utilizador pode escolher dar logout na sua conta;
  - `Home:` Onde o utilizador é redirecionado para a home da aplicação;
  - `Match:` Onde o utilizador é redirecionado para o match da aplicação;
  - `Settings:` Onde o utilizador é redirecionado para as definições da aplicação.

3.3.3 `loading.xml`

No `loading.xml` existe apenas uma `TextView` com o logótipo da aplicação. Surge quando a aplicação está em espera entre atividades.

3.3.4 `match.xml`

No `match.xml` existem diversos `TextView` e dois `Button`. O primeiro `TextView` alterna entre as mensagens *Pending!*, enquanto o utilizador está à espera de uma resposta do parceiro, e *Matched!*, quando foi encontrado um resultado para ambos. Quando ocorre o *match*, os restantes `TextView` surgem

no ecrã, com indicação do nome do restaurante, localização e o telemóvel da pessoa parceira. O `Button Cancel` está visível desde o início, permitindo cancelar a procura por parceiro ou a escolha final; o `Button Contact` surge quando ocorre o *match* e permite o contacto com a pessoa parceira, definida previamente no *layout* 3.3.6 ou escolhida pela aplicação no caso de *blind date*. Os `Button` da barra inferior estão especificados no *layout* da secção 3.3.2.

3.3.5 `register.xml`

No `register.xml` encontram-se várias instâncias de `EditText` onde o utilizador coloca as suas informações para criar a conta. Estas informações incluem: *email*, *palavra passe*, *primeiro nome*, *último nome*, *idade* e *número de telemóvel*. O `EditText` para *estado civil* encontra-se bloqueado com o valor *single* por defeito. Existem também instâncias de `Spinner` onde o utilizador seleciona o seu *género* e a sua *preferência*. Um `Button Register` encontra-se disponível para o utilizador confirmar os seus dados. Este *layout* dispõe de `ScrollView` que permite uma boa apresentação em dispositivos mais pequenos.

3.3.6 `settings.xml`

O `settings.xml` é composto por diversas `TextView` e `EditText`. As `TextView` indicam o conteúdo dos campos que podem ser editados nas `EditText` - alguns dos quais vêm pré-preenchidos por informações do *layout* `register.xml` (secção 3.3.5), nomeadamente primeiro e último nomes, número de telemóvel e idade. Existe também o `Button Confirm Changes`, que confirma as edições efetuadas. Os `Button` da barra inferior estão especificados no *layout* da secção 3.3.2.

3.4 Detalhes de Implementação

As linguagens utilizadas foram `Java`, para a escrita do código fonte da aplicação, `xml`, para a escrita dos *layouts* especificados na secção 3.3, e `JavaScript`, para a manipulação da *Firebase* no *backend*.

Para que no *swipe* surjam os diferentes restaurantes disponíveis, aplicou-se uma `CardStackView`, que permite mostrar os diferentes locais como cartas.

3.5 Manual de Instalação

Após o *download* do ficheiro de extensão `.apk` relativo à aplicação `ResTinder`, basta abrir a pasta de *downloads* do telemóvel, clicar no ficheiro descarregado e permitir a sua instalação pelo sistema. Depois, basta apenas abrir para utilizar. Para utilizar a aplicação, é necessário ter a localização do telemóvel ativa.

3.6 Manual de Utilização

Ao abrir a aplicação pela primeira vez, serão pedidas permissões de acesso à localização. No ecrã inicial, pode registar-se ao clicar no botão `Register`, ou dar *login* após inserir o *email* e a palavra-passe nos locais corretos, no qual pode dar autorização à aplicação para guardar as credenciais para uma próxima visita. No primeiro *login*, é necessário o utilizador verificar a sua conta a partir do *email* inserido.

Após a entrada, é pedido para inserir o *email* do parceiro (se aplicável) e o raio de quilómetros da localização atual de acordo com a escolha do utilizador para encontrar restaurantes. Logo em seguida, aparece uma lista de restaurantes como um conjunto de cartas, as quais pode arrastar para a direita de modo a aceitar a sugestão ou para a esquerda de modo a rejeitar. Depois, será apresentado um ecrã com a mensagem *Pending* enquanto não houver *matches* na mesma região (caso o utilizador não tenha parceiro definido nas definições) ou enquanto o parceiro não tenha feito a sua escolha. Neste momento, é possível sair da aplicação - uma vez que haverá uma notificação que lhe dirá em que estado está a procura: *Finding match.*

Ao clicar na notificação, quando esta alterar para *You have a match!*, será reencaminhado para a aplicação, que terá as informações relativas ao restaurante escolhido pela aplicação. Se estiver à espera de um *blind date*, então terá também informações sobre o parceiro que a aplicação escolheu para o utilizador.

3.7 Conclusão

A implementação contempla também os testes intermédios feitos ao código para confirmar como este é executado e verificar a existência de erros e *bugs*, que foram sendo corrigidos à medida que eram detetados. Ao final desta fase, estando tudo bem definido e funcional, o programa pode ser utilizado em qualquer sistema `Android`^{TM}, podendo considerar-se o projeto como concluído.

Capítulo 4

Reflexão Crítica e Problemas Encontrados

4.1 Introdução

Qualquer projeto, durante o seu período de implementação, apresenta a quem o implementa diversos problemas que é necessário resolver. A sobreposição entre objetivos alcançados e objetivos propostos é analisada na secção 4.2 e a divisão das tarefas para os atingir é discriminada na secção 4.3. Os problemas encontrados e a reflexão crítica sobre os mesmos e sobre os objetivos são explorados, respetivamente, nas secções 4.4 e 4.5.

4.2 Objetivos Propostos versus Alcançados

Todas as funcionalidades propostas, obrigatórias ou opcionais, foram implementadas na aplicação. É possível escolher entre restaurantes num determinado raio ajustável pelo utilizador, que os pode escolher dando *swipe* para a direita (aceitar a sugestão) ou *swipe* para a esquerda (rejeitar a sugestão). Os utilizadores podem registar-se e os seus dados são guardados numa base de dados *online*, o que permite a correspondência entre utilizadores para os *blind dates*. É possível contactar a outra pessoa após o *match*, através de um botão apresentado na aplicação.

4.3 Divisão de Trabalho pelos Elementos do Grupo

Todo o trabalho foi realizado com o esforço conjunto dos cinco membros do grupo, ou seja, mesmo que um dos elementos se tenha focado mais numa determinada tarefa, os restantes participaram também, direta ou indiretamente, para a concretização da mesma. Assim, cada elemento tem discriminado as tarefas que executou:

- **Alexandre Monteiro:** implementação da *Firebase* e do *backend*;
- **Bruno Monteiro:** *layouts* `xml match`, `home`; implementação dos *swipes* e da obtenção dos restaurantes pelo API;
- **Manuel Carvalho:** *layout* `xml settings`; implementação de serviços e notificações;
- **Manuel Magalhães:** *layout* `xml match`; implementação do *match*;
- **Sara Martins:** *layouts* `xml match`, `register`; implementação de serviços e notificações; relatório.

4.4 Problemas Encontrados

A *Firebase* necessita de um tempo extra, pontualmente, para fornecer os seus serviços com a rapidez desejada pelos utilizadores. Porém, com o maior número de utilizações, a *Firebase* torna-se mais fluida na aquisição de dados.

Para permitir o *match* em tempo real entre utilizadores da aplicação, foi necessário implementar o *backend* em `JavaScript`. A utilização desta linguagem aumentou a complexidade na implementação do código.

Os serviços vinculados não permitiam, pelo menos não de forma simples, guardar as conexões necessárias após o fecho da aplicação pelo método `onDestroy`. Assim, os serviços foram alterados para serviços não vinculados, de modo a permitir que continuassem a escutar os eventos da base de dados mesmo com a atividade destruída. Para impedir a destruição do serviço pelo sistema cinco segundos após a destruição da aplicação, as notificações foram implementadas em *foreground*.

Para criar as notificações, houve o problema de definir os canais de transmissão - obrigatórios a partir da API 26. Foi possível colmatar este problema após uma pesquisa exaustiva pela documentação.

4.5 Reflexão Crítica

O grupo considera que houve um ótimo desempenho na realização deste projeto, tendo em conta a falta de tempo para otimizar o código e o reduzido foco na engenharia de software. Permitiu o aprofundamento de conhecimentos adquiridos nas aulas teóricas, nomeadamente os serviços e notificações, que não foram trabalhados em contexto de aulas práticas, e da linguagem `JavaScript`, que não é lecionada em licenciatura.

Houve uma ótima gestão do trabalho entre os diferentes elementos do grupo, o que facilitou a divisão de tarefas, a comunicação interpessoal e o desenvolvimento do projeto.

Considera-se que os problemas principais encontrados, definidos na secção 4.4, estão fora do controlo do grupo, pelo que não poderiam ser resolvidos com agilidade dentro do prazo de entrega.

4.6 Conclusão

O projeto permitiu a concretização de todos os objetivos propostos, com uma boa agilização de tarefas entre os diversos elementos do grupo de trabalho. Esta boa comunicação permitiu ultrapassar todos os problemas que eram passíveis de resolver durante o período permitido de desenvolvimento.

Capítulo 5

Conclusões e Trabalho Futuro

5.1 Conclusões Principais

O desenvolvimento deste projeto permitiu a todos os elementos do grupo aplicar e consolidar os conhecimentos adquiridos em contexto de aula, quer teórica, quer prática, da unidade curricular de Programação de Dispositivos Móveis. Num aspeto de trabalho mais autónomo, foi possível aprender e apreender conhecimentos de outras áreas, nomeadamente uso de bases de dados remotas e utilização de funcionalidades da plataforma Android Studio não lecionadas. Permitiu também o entendimento da importância das boas práticas de documentação de código a longo prazo e a melhoria das *soft skills* de análise e resolução de problemas e comunicação interpessoal. Acima de tudo, possibilitou a cada elemento deste grupo de trabalho crescer, enquanto pessoa e enquanto futuro engenheiro informático.

5.2 Trabalho Futuro

Para além de todas as funcionalidades conseguidas no desenvolvimento deste projeto, seria interessante também poder expandi-lo para outras opções. Por exemplo, poder-se-ia considerar a possibilidade de tornar a aplicação autossustentável, com a integração de anúncios ou planos de subscrição. Poder-se-á também otimizar o código, tanto na parte em `Java` ou `JavaScript`, para permitir uma melhor *performance*. Além disso, poderia-se arranjar alternativa à API da Google, visto que por vezes tem lentidão na resposta aos pedidos dos utilizadores.

Bibliografia

[1] Pedro R. M. Inácio and Paulo A. P. Fazendeiro. Propostas para Trabalhos de Grupo, 2021. [Offline] Último acesso a 25 de novembro de 2021.

[2] Pedro R. M. Inácio. *Introdução à Programação de Aplicações Android - Apontamentos de Apoio e Guias Laboratoriais de Programação de Dispositivos Móveis*. ISBN: 978-989-654-789-9. UBI - Universidade da Beira Interior: Serviços Gráficos, October 2021.

[3] Nuno G. C. C. Pombo. Engenharia de Software 2021/2022, 2021. [Online] `https://www.di.ubi.pt/~ngpombo/units/se.html`. Último acesso a 25 de novembro de 2021.

[4] Google. Android Studio, 2021. [Online] `https://developer.android.com/studio`. Último acesso a 28 de novembro de 2021.

[5] Google. Firebase, 2021. [Online] `https://firebase.google.com/`. Último acesso a 28 de novembro de 2021.

[6] Google. Places API, 2021. [Online] `https://developers.google.com/maps/documentation/places/web-service/overview`. Último acesso a 28 de novembro de 2021.

[7] Github. Github, 2020. [Online] `https://github.com/`. Último acesso a 28 de novembro de 2021.

[8] Microsoft. Microsoft Whiteboard, 2021. [Online] `https://www.microsoft.com/en-us/microsoft-365/microsoft-whiteboard/digital-whiteboard-app`. Último acesso a 28 de novembro de 2021.

[9] Visual Paradigm. Visual Paradigm Community, 2021. [Online] `https://www.visual-paradigm.com/download/community.jsp`. Último acesso a 19 de dezembro de 2021.