



DDETC – Departamento de Engenharia Eletrónica e Telecomunicações e de Computadores

MEIM - Mestrado Engenharia informática e multimédia

## **Engenharia de software**

### **Trabalho final**

**Turma:**

11D

**Trabalho realizado por:**

Miguel Távora      N°45102

**Docente:**

Luís Morgado

**Data:** 06/02/2022



# Índice

|                                                         |           |
|---------------------------------------------------------|-----------|
| <b>1. INTRODUÇÃO .....</b>                              | <b>1</b>  |
| <b>2. DESENVOLVIMENTO .....</b>                         | <b>4</b>  |
| <b>1. ANÁLISE DE REQUISITOS .....</b>                   | <b>4</b>  |
| <b>1.1 DOCUMENTO DE VISÃO.....</b>                      | <b>4</b>  |
| <b>1.2 ESPECIFICAÇÃO DE REQUISITOS .....</b>            | <b>5</b>  |
| <b>2. PROJETO DE ARQUITETURA DA APLICAÇÃO .....</b>     | <b>9</b>  |
| <b>2.1 ARQUITETURA LÓGICA .....</b>                     | <b>9</b>  |
| 2.1.1 MODELO DE DOMÍNIO .....                           | 10        |
| 2.1.2 DIAGRAMAS DE INTERAÇÃO .....                      | 11        |
| 2.1.3 PADRÕES DE ORGANIZAÇÃO DE SUBSISTEMAS .....       | 12        |
| 2.1.4 REALIZAÇÃO DOS CASOS DE UTILIZAÇÃO.....           | 13        |
| 2.1.5 ARQUITETURA DE MECANISMOS .....                   | 14        |
| 2.1.6 ARQUITETURA GERAL DA SOLUÇÃO .....                | 14        |
| <b>2.2 ARQUITETURA DETALHADA .....</b>                  | <b>15</b> |
| 2.2.1 MODELO DE DINÂMICA.....                           | 15        |
| 2.2.2 ARQUITETURA DE TESTE.....                         | 16        |
| 2.2.3 MODELO DE IMPLANTAÇÃO .....                       | 17        |
| <b>3. IMPLEMENTAÇÃO DO PROTÓTIPO DE TESTE .....</b>     | <b>18</b> |
| <b>4. IMPLEMENTAÇÃO DO PROTÓTIPO APLICACIONAL .....</b> | <b>20</b> |
| <b>5. ANÁLISE CRÍTICA DO PROJETO REALIZADO .....</b>    | <b>23</b> |
| <b>3. CONCLUSÕES .....</b>                              | <b>24</b> |
| <b>4. BIBLIOGRAFIA .....</b>                            | <b>26</b> |

## Índice ilustrações e tabelas

|                                                                                                 |    |
|-------------------------------------------------------------------------------------------------|----|
| Figura 1 - Caso geral dos casos de utilização .....                                             | 5  |
| Figura 2 - Diagramas de casos de utilização do sistema visualização dos utilizadores .....      | 6  |
| Figura 3 - Descrição detalhada do caso utilização atribuir gosto e não gosto .....              | 7  |
| Figura 4 - Realização do caso de utilização Atribuir gosto ou não gosto – Guardar seleção ..... | 13 |
| Figura 5 - Modelo dinâmica de retrocesso de um perfil .....                                     | 16 |
| Figura 6 - Diagrama de implantação .....                                                        | 17 |
| Figura 7 - Arquitetura de teste .....                                                           | 18 |
| Figura 9 - Vista atribuir gosto ou não gosto .....                                              | 21 |
| Figura 8 - Página principal .....                                                               | 21 |
| Figura 11 - Vista atribuir classificação ao utilizador .....                                    | 22 |
| Figura 10 - Vista responder ao questionário .....                                               | 22 |
| <br>                                                                                            |    |
| Tabela 1 - Especificação suplementar reduzido .....                                             | 8  |

# 1. Introdução

Para se entender o que é engenharia de software é importante primeiro entender o que é o software. Software é uma especificação de configurações de um sistema computacional que determina a sua operação. Diferente de um programa que corresponde ao conjunto de instruções para um computador.

Engenharia de software é o processo de desenvolvimento de software como uma atividade de engenharia. Um engenheiro de software é uma pessoa que aplica os princípios de engenharia de software para conceber, desenvolver, manter, testar e avaliar software. O processo de desenvolvimento de software envolve a definição, implementação, avaliação, mediação, gestão, mudança e melhoria do ciclo de vida do software. A crise do software no final da década de 1960 resultou no surgimento do termo numa conferência da NATO, onde foi decidido que os princípios da engenharia deveriam ser aplicados ao desenvolvimento de software.

Esta crise deve-se à explosão combinatória das interações de um sistema, resultando em complexidade desorganizada. Esta complexidade resulta da heterogeneidade das partes do sistema, as partes interatuam entre si, mas a interação é irregular. Isto resulta no crescimento exponencial da entropia. A entropia é uma medida que mede o grau de desordem sobre um sistema fechado num dado instante temporal. A alta entropia resulta em esforço no desenvolvimento, manutenção e evolução do sistema. Para isso utiliza-se a engenharia de software para obter complexidade organizada, onde as interações entre partes obedecem a padrões correlacionáveis no espaço e tempo. Desta forma é possível obter o mínimo de entropia possível através da ordem, organização e coerência funcional e consequentemente o mínimo de esforço.

O trabalho consiste em escolher e desenvolver uma aplicação à escolha do aluno. Neste sentido o trabalho escolhido para desenvolver foi uma aplicação para conhecer pessoas. Esta permite aos utilizadores conhecerem outras pessoas, que neste caso são utilizadores, nas proximidades onde estão a utilizar a aplicação. Desta forma para conseguir que a aplicação funcione é necessário que o dispositivo possua um sensor de localização e por isso será desenvolvido para telemóvel. O que irá diferenciar esta aplicação das restantes aplicações que já existem no mercado atualmente é possuir aprendizagem automática. Desta forma será possível conseguir apresentar utilizadores que podem despertar mais interesse para um determinado utilizador.

Este trabalho enquadrasse no âmbito de engenharia de software na medida que durante todo o processo de desenvolvimento serão utilizadas abordagens sistemáticas, disciplinadas e quantificáveis de desenvolvimento de software. Desta forma será possível obter um sistema com baixa entropia e será preciso o mínimo de esforço para o seu desenvolvimento.

Por isso antes do início do desenvolvimento foram seguidas as práticas que se segue:

- Análise de requisitos – Esta parte insere-se na análise no processo de desenvolvimento, onde é necessário conhecer o problema e definir a solução para o problema. Esta parte é uma parte importante, porque a parte mais difícil de construir software é decidir precisamente o que construir.
- Especificação de requisitos – Esta parte é construída a partir da análise de requisitos, no qual são definidos os casos de utilização e as suas respetivas descrições. Os diagramas de casos de utilização descrevem a funcionalidade do sistema na perspetiva dos atores, ou seja, os objetivos que é suposto a aplicação realizar.
- Arquitetura lógica – Parte onde são especificadas as partes e as relações entre partes através de um modelo de domínio, diagramas de sequência, diagramas de classes dos subsistemas e interação global entre partes. Esta arquitetura por ser especificada em linguagem UML não está presa a uma tecnologia ou linguagem de programação, podendo ser implementada em qualquer linguagem.
- Arquitetura específica – Parte da arquitetura que define como será realizada a implementação da solução. Nesta fase é definida a dinâmica da aplicação, arquitetura

de teste e quais as plataformas aplicacionais e linguagens de programação escolhidas para a implementação da solução.

## 2. Desenvolvimento

### 1. Análise de requisitos

#### 1.1 Documento de visão

A análise de requisitos encontra-se na fase da análise no processo de desenvolvimento. O processo de desenvolvimento é constituído por análise, conceção, construção e verificação. Na fase da análise é preciso conhecer o problema, nomeadamente identificar, analisar e compreender o problema. De seguida é preciso definir a solução, onde são identificados e especificados os requisitos da solução. O problema dos requisitos reside em que estes são ambíguos, incompletos e inconsistentes, sendo também por isso criativos, adaptáveis e versáteis. Por outro lado, um sistema implementado é preciso, consistente e completo. Por isso os requisitos devem ser especificados com a maior precisão possível.

No desenvolvimento da aplicação começou-se por desenvolver o documento de visão. Este documento descreve o sistema em termos gerais, qual o contexto operacional e organizacional, quais as partes envolvidas e características. Define o problema e a solução para o mesmo. Esta parte do desenvolvimento é feito em linguagem natural, mas oferece uma estrutura definida do funcionamento do sistema. Assim as ideias ficam mais claras e precisas sobre o que é suposto a aplicação realizar, não ficando as ideias só na imaginação. Isto deve-se à linguagem possuir regras e restrições inerentes para a compreensão. Este documento geralmente é estabelecido entre a equipa de desenvolvimento e o cliente, contudo neste projeto o desenvolvedor será também cliente do projeto. Por isso foi escrito no documento toda a informação que o desenvolvedor achou pertinente e que ajudaria no desenvolvimento da aplicação. Contudo não será mostrado pois perder-se-ia o foco de todo o processo de desenvolvimento de software, por isso durante toda a descrição das diferentes etapas de desenvolvimento serão utilizados apenas pequenos exemplos ilustrativos do que foi concebido para não se perder o foco com detalhes supérfluos.



Simultaneamente será criado um também um glossário. O glossário é necessário devido a que uma das grandes dificuldades do desenvolvimento de software residir em falhas de comunicação, principalmente entre cliente e fornecedor, porque cada interlocutor utiliza o seu vocabulário próprio. O glossário vem para definir uma linguagem comum entre o cliente e o fornecedor, eliminando assim as ambiguidades.

## 1.2 Especificação de requisitos

Existem diversas técnicas de identificação de requisitos, nomeadamente: através de entrevistas, questionários, sessões de identificação de requisitos entre outros. Neste trabalho iremos seguir o padrão de casos de utilização.

No desenvolvimento da aplicação foi utilizado a abordagem *top-down* na identificação dos casos de utilização. Nesta abordagem o ponto de partida é o documento de visão. Após contruído este documento é definida a fronteira do sistema e a identificação dos atores. Neste sentido foi obtido a seguinte fronteira e atores:

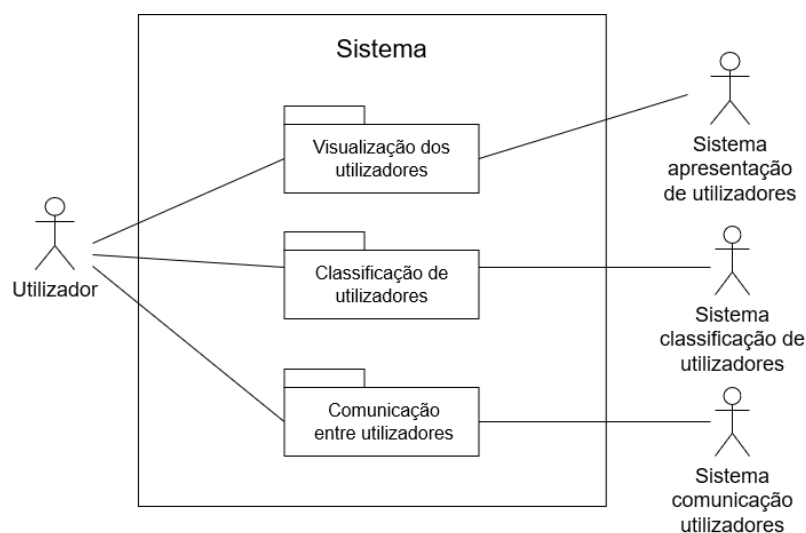
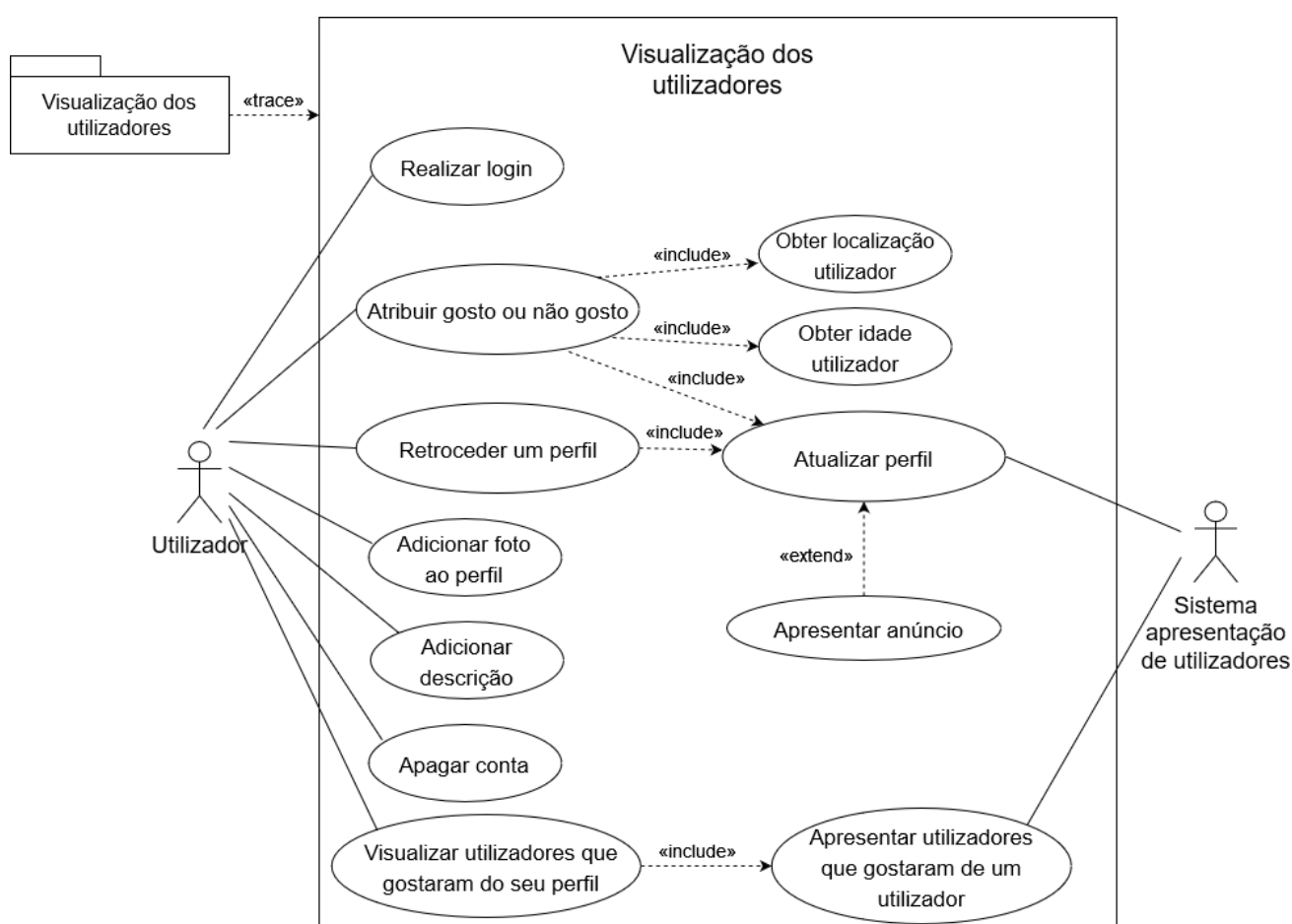


Figura 1 - Caso geral dos casos de utilização

A fronteira do sistema na figura anterior delimita o que é responsabilidade do sistema do que não é. O interior da fronteira é responsabilidade do sistema e o exterior representa os atores que interagem com o sistema. A identificação dos atores é feita através do conhecimento sobre quem interage com o sistema.

Feito isto de seguida é necessário identificar os casos de utilização. Os casos de utilização são na verdade a concretização de objetivos dos atores, por isso é a forma como o sistema lida com eventos relevantes para os atores. No desenvolvimento os casos de utilização foram separados de acordo com as pastas para facilitar a compreensão. Um dos diagramas de casos de utilização desenvolvidos foi:



**Figura 2 - Diagramas de casos de utilização do sistema visualização dos utilizadores**

A partir dos casos de utilização é possível realizar a descrição dos casos de utilização. A descrição dos casos de utilização tem o objetivo de descrever o fluxo de eventos principais e alternativos. Os fluxos principais seguem um fluxo básico e completo da descrição do caso de utilização. Os fluxos alternativos são desvios do fluxo principal, através desta metodologia é possível tornar a descrição precisa e fácil de ler e compreender.

Na especificação dos casos foi escolhido a descrição detalhada do cenário principal e cenários alternativos no formato de uma coluna. Uma alternativa seria utilizar o formato de duas colunas, onde existe numa das colunas os atores e na outra coluna o sistema. No projeto foi realizada a descrição do caso de utilização de atribuir gosto ou não gosto e o resultado foi:

### **Atribuir gosto e não gosto**

**Resumo:** O caso de utilização permite ao utilizador atribui gosto aos utilizadores que tem interesse e pretende conhecer.

**Atores:** Utilizador, Sistema apresentação de utilizadores

**Pré-condições:** O utilizador encontra-se na página principal

#### **Cenário principal:**

1. O caso de utilização começa quando o utilizador pretende conhecer pessoas novas.
2. O utilizador que já se encontra ligado na sua conta, seleciona “procurar pessoas”.
3. O sistema apresentação obtém a localização do utilizador.
4. O sistema recebe a localização do utilizador.
5. O sistema obtém qual a distância máxima que devem ser apresentados os utilizadores.
6. O sistema obtém as idades aceitáveis pelo utilizador.
7. O sistema obtém o(s) género(s) de interesse para o utilizador.
8. O sistema procura utilizadores que se encontrem a uma distância aceitável dentro dos padrões selecionados pelo utilizador.
9. O sistema apresentação muda o *screen*/para apresentar pessoas no qual o utilizador pode estar ou não interessado.
10. O utilizador atribui gosto ou não gosto a um utilizador apresentado.
11. O sistema atualiza para um novo perfil.
12. O sistema apresenta um anúncio ao utilizador arbitrariamente após ser apresentado um determinado número de utilizadores.
13. O ponto 8. , 9. e 10. são repetidos até o utilizador atingir o limite máximo de atribuição de gostos.

**Figura 3 - Descrição detalhada do caso utilização atribuir gosto e não gosto**

Um problema no caso de utilização anterior é este possuir demasiados *includes*, resultando numa complexidade grande inerente ao caso de utilização. Desta forma torna-se mais complicado entender e posteriormente desenvolver o caso de utilização.

De seguida será desenvolvida a especificação suplementar. O modelo de casos de utilização representa os requisitos funcionais, elaborado na perspetiva dos utilizadores e a especificação suplementar representa requisitos não funcionais. Os requisitos não funcionais ditam as restrições a que o sistema deve obedecer. Neste sentido alguns dos requisitos obtidos foram:

| Ref  | Descrição                                                          | Categoria   |
|------|--------------------------------------------------------------------|-------------|
| R1   | A aplicação deve em relação ao hardware:                           | Obrigatório |
| R1.1 | Ajustar-se à resolução de qualquer telemóvel                       | Obrigatório |
| R1.2 | A aplicação deve funcionar para qualquer tipo de sistema operativo | Opcional    |

**Tabela 1 - Especificação suplementar reduzido**

## 2. Projeto de arquitetura da aplicação

### 2.1 Arquitetura lógica

O projeto de arquitetura da aplicação é construído a partir da análise de requisitos. Esta fase pretende analisar, compreender e verificar a forma de realizar a solução proposta através da elaboração da arquitetura da solução. A arquitetura é uma descrição de alto nível do sistema feito através de uma linguagem de modelação designada UML. Um modelo é uma representação abstrata de um sistema e é uma forma de lidar com complexidade. Estes modelos são muito úteis no desenvolvimento de software, visto que realizar software é uma tarefa complexa.

Existem três níveis de modulação de um sistema:

- Independente do modelo computacional – Descreve o modelo de negócio, o contexto de utilização do sistema, o seu comportamento e características esperadas. Esta arquitetura foi implementada através da análise de requisitos.
- Independente da plataforma de execução – Descreve o sistema com tanto detalhe quando possível de forma independente da plataforma de execução. Este nível é conhecido como arquitetura lógica, que será desenvolvido neste capítulo.
- Específico da plataforma de execução – Descreve a concretização do sistema para uma plataforma específica. Este nível é conhecido como arquitetura detalhada e será desenvolvido no próximo capítulo.

Existem diversas métricas de análise à arquitetura, nomeadamente:

- Coesão – Representa uma medida de até que ponto um subsistema/módulo realiza uma única função. Esta métrica deve-se tentar obter o maior valor possível, desta forma na necessidade de alteração o número de módulos afetados é baixo.
- Acoplamento – Grau de interdependência entre subsistemas. Esta métrica deve ser o mais baixa possível, garantindo maior facilidade de desenvolvimento, instalação,

manutenção e expansão.

- Simplicidade – Nível de facilidade de compreensão/comunicação da arquitetura.
- Adaptabilidade – Nível de facilidade de alteração da arquitetura para acomodar novos requisitos.

Existem diversos paradigmas de programação, mas no caso da aplicação será utilizada o paradigma orientado a objetos. Neste paradigma o sistema é modelado como um conjunto de objetos que interagem para produzir o comportamento pretendido.

Nesta fase é necessário definir um aspeto importante para a análise orientada a objetos que é a descrição do problema em termos de objetos relevantes.

### **2.1.1 Modelo de domínio**

O modelo de domínio faz a representação das classes e as relações conceptuais, que descrevem as situações que ocorrem num determinado domínio. O foco é nos conceitos e não no domínio da solução. Será a partir deste modelo que serão criadas as classes do tipo entidade. As entidades representam tipos de objetos do domínio do problema, os atributos representam características das entidades. Existe também as associações que representam relações entre entidades/instâncias e restrições que representam requisitos adicionais. Para a identificação destas informações é necessário primeiro identificar os elementos, onde nomes representam classes, atributos e objetos e os verbos indicam operações e relações. Desta forma é possível eliminar elementos vagos, ambíguos e redundantes.

As vantagens da utilização deste modelo é conseguir ter um enquadramento das entidades do domínio do problema, ajudando a perceber o problema. Com base neste modelo é possível descrever o estado do domínio do problema em qualquer situação.

Em termos de realização deste modelo, este não foi muito bem concebido na perspectiva de encontrar as entidades de domínio. Isto porque este modelo não possui muitas das entidades utilizadas posteriormente na arquitetura. Contudo foi útil para perceber em termos gerais o funcionamento do sistema e para descobrir os serviços existentes e algumas entidades.

### 2.1.2 Diagramas de interação

Os diagramas de interação descrevem a comunicação entre objetos numa interação entre partes do sistema e/ou com o exterior do sistema. Existem dois tipos de diagramas, os diagramas de sequência e os diagramas de comunicação. Os diagramas de sequência possuem a ênfase no tempo e na sequência de interação. Os diagramas de comunicação possuem a ênfase na estrutura. No projeto não foram utilizados diagramas de comunicação por isso não serão desenvolvidos.

Os diagramas de sequência possuem uma organização bidimensional onde na parte vertical representa o tempo e na parte horizontal a estrutura. Neste diagrama existem quatro elementos a *lifeline* que representa a evolução temporal do sistema, a *activation bar* que representa a execução de operações, a mensagem e o operador. As mensagens possuem uma sintaxe própria, que é [attribute=]message-name [(argument)] [:return-value]. Os operadores são fragmentos de interação com semântica específica por exemplo para representar que um elemento é opcional ou está dentro de um *loop*.

### 2.1.3 Padrões de organização de subsistemas

Primeiramente para facilitar a construção foi utilizado a arquitetura lógica de 3 camadas. As camadas são nomeadamente:

- Apresentação – Parte visível do sistema para o utilizador. Parte que obtém os eventos vindos dos atores.
- Domínio – Parte que implementa a lógica de negócio da aplicação.
- Acesso a dados - Diferente dos dados que representam base dados em SQL ou base dados NoSQL. Esta camada representa a forma como são acedidos os dados na base de dados.

Nesta fase iremos focar-nos somente na primeira camada que é a camada de apresentação. Existem diversos padrões de apresentação. O modelo MVC (Model-View-Controller) que separa a modularidade do domínio, a apresentação e as ações baseadas no *input* do utilizador em três classes distintas. Além deste padrão existe também o padrão MVP (Model-View-Presenter) e o MVVM (Model-View-ViewModel). O padrão de apresentação utilizado foi o MVVM, pela razão de que a aplicação será construída para telemóvel e é necessário a atualização automática das vistas. O padrão MVVM oferece um adaptador de apresentação (ViewModel) e realiza a atualização automática das vistas.

Em relação à camada de domínio este pode ter:

- Entidades – Identidade própria, que se mantém ao longo da operação do sistema.
- Agregação – Mantém a consistência de objetos relacionados.
- Fábrica – Gestão de criação de objetos do domínio e encapsula o conhecimento acerca do modo de criação dos objetos.
- Repositório – Realiza a gestão do armazenamento de objetos do domínio, possuindo as funcionalidades de: inserir, obter, atualizar, remover.
- Serviço – Representa o comportamento relevante do domínio, que não é específico de



um elemento de domínio, e coordena a atividade de diferentes objetos.

No caso da implementação do projeto foram utilizadas as entidades, repositórios e os serviços. Os repositórios foram utilizados para realizar o acesso aos dados.

### 2.1.4 Realização dos casos de utilização

As realizações dos casos de utilização foram feitos através de diagramas sequência e do modelo de três camadas. No caso só foram realizados dois de todos os casos. A sua construção foi feita primeiramente com diagramas mais simples, visto que os casos de utilização escolhidos possuíam um tamanho relativamente grande. Os diagramas de sequência finais foram separados em parte de apresentação e parte de domínio para facilitar a compreensão, na medida que cada diagrama não deve possuir mais do que 7 instâncias diferentes. Um exemplo obtido de um diagrama de sequência foi:

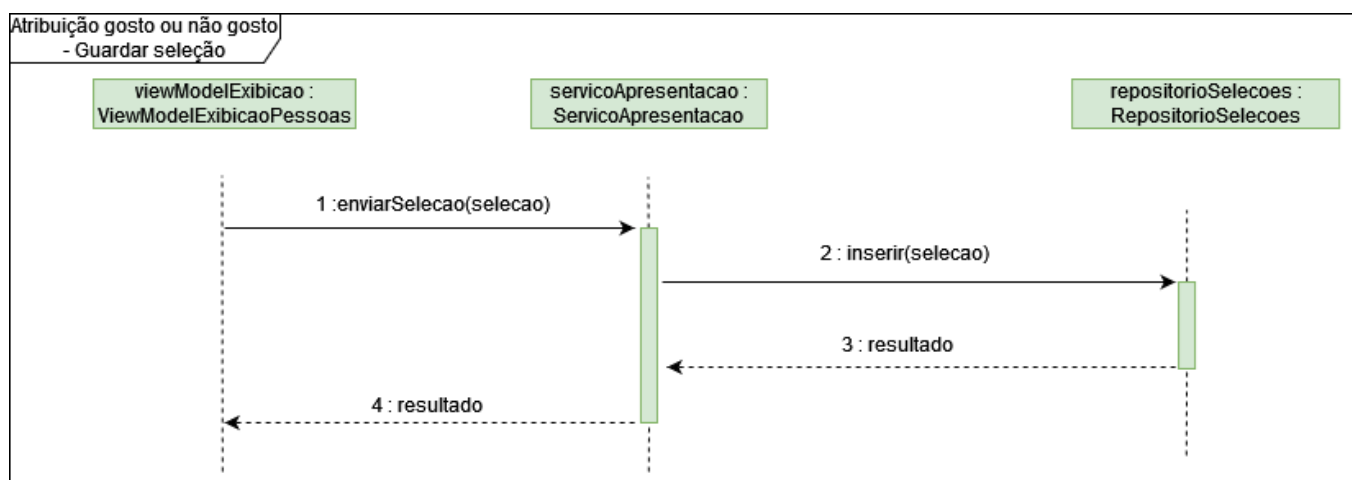


Figura 4 - Realização do caso de utilização Atribuir gosto ou não gosto – Guardar seleção

### **2.1.5 Arquitetura de mecanismos**

A partir dos diagramas de sequência é possível construir a arquitetura de mecanismos. Esta arquitetura utiliza por base os diagramas de classes. Estes diagramas oferecem através de uma linguagem gráfica uma visão geral das classes e das relações entre as classes. Uma vantagem da utilização desta linguagem gráfica é o facto de poder ser implementado em qualquer linguagem de programação.

### **2.1.6 Arquitetura geral da solução**

Esta arquitetura, da mesma forma que a arquitetura de mecanismos, utiliza os diagramas de classes para a sua representação em termos de funcionamento. A principal diferença entre a arquitetura geral da solução e a arquitetura de mecanismos é que a arquitetura geral da solução é construída a partir da arquitetura de mecanismos, consequentemente possui um maior nível de abstração. A arquitetura de mecanismos mostra o funcionamento de um subsistema e a forma como ele interage. A arquitetura geral da solução mostra como o sistema funciona como um todo e quais as relações entre os subsistemas. Nesta solução foi escolhido utilizar o modelo lógico de três camadas para construir a arquitetura geral da solução. Desta forma os serviços e as entidades ficam no domínio, os repositórios no acesso a dados e as vistas e os ViewModels na camada de apresentação.

## 2.2 Arquitetura detalhada

A arquitetura detalhada como referido anteriormente descreve a concretização do sistema para uma plataforma específica. Nesta fase foram realizados os seguintes modelos:

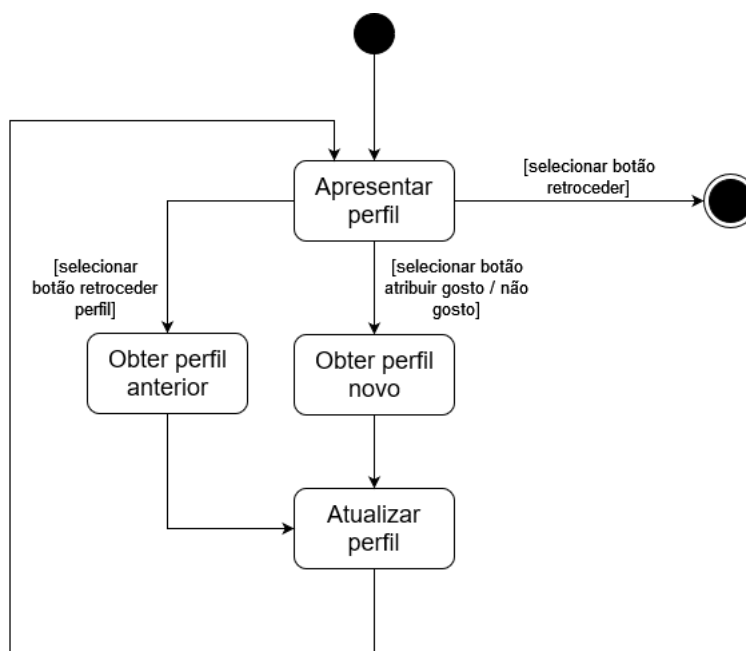
- Modelo de dinâmica
- Arquitetura de teste
- Modelo de implantação

### 2.2.1 Modelo de dinâmica

O modelo de dinâmica oferece uma organização no espaço (estrutura) e no tempo (dinâmica).

- A estrutura mostra as partes e as relações entre partes de um sistema.
- A dinâmica mostra a forma como as partes e as relações de partes evoluem no tempo.
- A estrutura e a dinâmica juntos formam o comportamento que representa a forma como um sistema age perante estímulos do ambiente.

Este modelo foi utilizado em duas situações. A primeira foi para se entender o modo de funcionamento da navegação das diferentes páginas da aplicação. A segunda foi para representar a dinâmica de quando um utilizador está a atribuir gostos e não gostos e pretende retroceder um perfil apresentado. O resultado obtido foi o que se segue:



**Figura 5 - Modelo dinâmica de retrocesso de um perfil**

Não será muito detalhado este modelo, devido à grande variedade de funcionalidades que este apresenta.

### 2.2.2 Arquitetura de teste

A arquitetura de teste tem como objetivo testar camada de domínio na arquitetura lógica de três camadas. Para isso o ambiente gere a interface de utilização de forma que os programas de teste possam ser independentes da interface. O acesso a dados possui alguns dados de teste para verificar o correto funcionamento. Dentro da arquitetura vão ser realizados diversos testes um por cada caso de utilização. Contudo um dos casos de utilização possui diversos casos de utilização mais simples no seu interior e por isso foram testados também esses casos individualmente.

### 2.2.3 Modelo de implantação

Este diagrama representa as tecnologias que serão utilizadas na implementação do sistema. Os artefactos representam recursos físicos que o sistema utiliza ou executa. Ligações representam canais de comunicação e os nós que são recursos físicos capazes de executar artefactos. Neste caso para a implantação foi escolhido o android como sistema operativo, onde vai correr a apresentação. Na comunicação entre o servidor e a apresentação será utilizado o *middleware* gRPC. Para linguagem de programação será utilizado o Java para ambos o domínio e apresentação. Como base de dados será utilizado o Firestore.

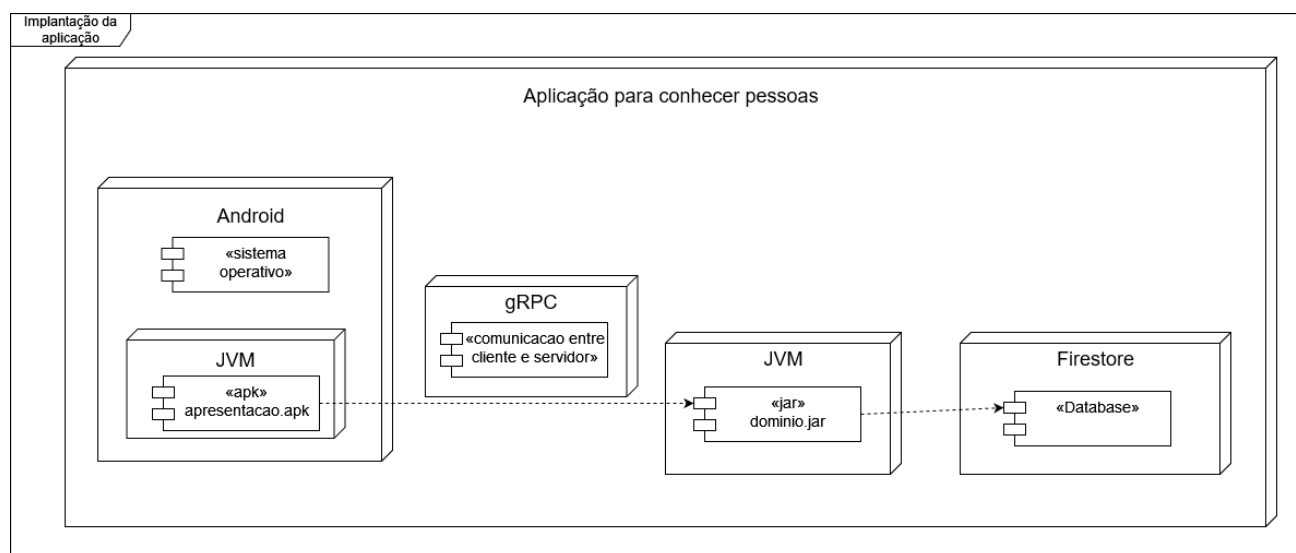


Figura 6 - Diagrama de implantação

### 3. Implementação do protótipo de teste

A arquitetura de teste tem por objetivo testar a camada de domínio. Para isso foi feito o seguinte diagrama:

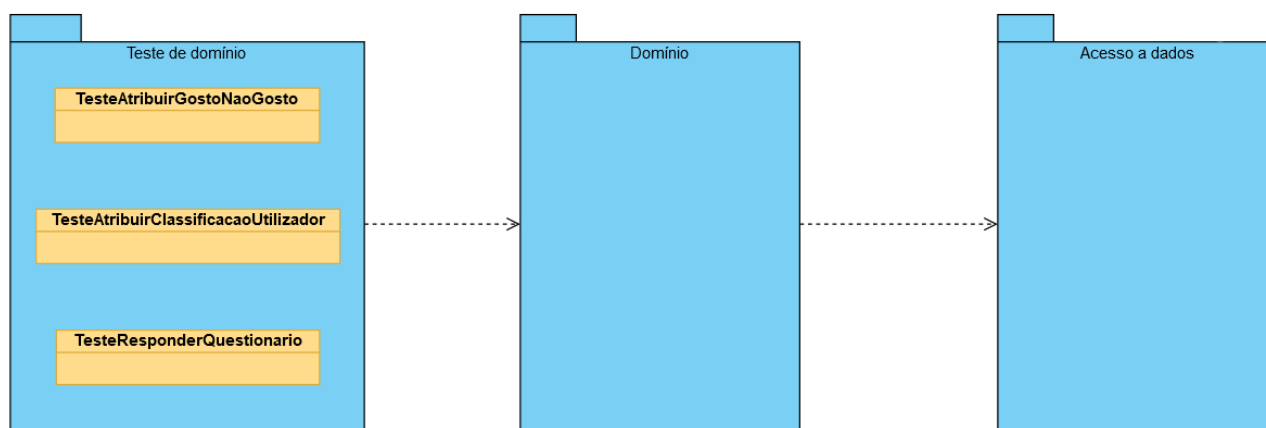


Figura 7 - Arquitetura de teste

As classes no interior da pasta do teste do domínio incluem também os ViewModels da camada de apresentação. Isto deve-se a que como a comunicação é realizada por gRPC para testar o bom funcionamento do domínio como servidor é necessário enviar objetos através deste *middleware* de comunicação. Em termos do acesso a dados foram implementadas as classes dos repositórios, mas com acesso a dados através de listas e mapas. No momento do arranque são inseridos dados por uma classe designada `InserirDados` de forma a conseguir testar efetivamente o bom funcionamento do domínio.

No acesso a dados, para testar o domínio devia ter sido utilizada uma interface no qual se criava uma classe de acesso a dados que implementava a interface. Desta forma tanto a implementação de teste como a implementação real do acesso a dados seguiriam o mesmo padrão em vez de alterar uma classe já implementada.

Um detalhe importante foi como existe a necessidade de implementação dos contratos para o gRPC, as operações dos contratos seguem o enquadramento do nome dos métodos no diagrama de mecanismos e os argumentos seguem o mesmo nome das entidades.

Como os casos de utilização possuem incorporados outros casos de utilização, têm de ser testados todos os casos. Desta forma existem testes para isso, como um dos testes ocorre dentro do domínio e do acesso a dados será chamada uma função no serviço de classificação para testar esse caso de utilização.

O caso de utilização de calcular os melhores utilizadores era pretendido utilizar aprendizagem automática para realizar esta tarefa. Contudo esta implementação seria muito demorada de implementar e testar. Uma outra limitação era a falta de dados, porque para obter um bom classificador por norma é preciso muitos dados, algo que não existe. Por isso foi optado por este método obter as classificações dadas aos utilizadores e criar uma recomendação a partir deste.

No caso de utilização de atribuir gosto ou não gosto não foi implementada a parte onde é obtida a localização geográfica do utilizador. Isto deve-se ao facto de não se saber como são estruturados os dados de localização. Uma outra limitação é esta parte estar dependente da apresentação e por isso só seria possível saber após implementar a apresentação. Um outro fator é a falta de tempo e também fugir do âmbito da unidade curricular.

Para realizar os testes é necessário correr primeiramente o domínio e só depois as diferentes classes na apresentação. Para realizar o teste sobre o cálculo dos melhores basta correr o domínio.

## 4. Implementação do protótipo aplicativo

Na implementação do protótipo aplicativo foram implementados os casos de utilização realizados anteriormente e também o caso de utilização de retroceder perfil.

Para implementar a apresentação foram utilizados na íntegra os diagramas da arquitetura de mecanismos, os diagramas de sequência e da arquitetura geral da solução. Através destes diagramas a implementação tornou-se essencialmente uma tradução dos diagramas para código Java. Ficando assim código fácil de produzir. Para construir a apresentação foi utilizado o AndroidStudio, com a versão 11 do Java. Foi utilizado o AndroidStudio porque é a única tecnologia conhecida, onde utilize por base o telemóvel como sistema computacional. Como foi dito anteriormente foi necessário construir os ViewModels para a arquitetura de teste, um detalhe importante é o facto de não ter sido preciso alterar o código nos ViewModels para incorporar com as Vistas criadas. Um problema que não foi conseguido resolver foi o facto de no gRPC foi utilizado o Maven para gerar os artefactos. O AndroidStudio utiliza o Gradle como repositório e a única maneira encontrada de incorporar os artefactos na aplicação foi adicioná-los localmente na pasta lib. Contudo quando é executado o programa num emulador os artefactos do gRPC não são adicionados à aplicação e por isso não é possível testar o funcionamento da apresentação a conectar-se com os serviços do domínio.

Para implementar o acesso a dados foi utilizado o Firestore e os diagramas UML dos repositórios, que possuem a mesma estrutura que as classes utilizadas para testar o domínio. Por isso foi alterado o conteúdo dos métodos dos repositórios para acederem ao Firestore. Esta base de dados é designada NoSQL, significando que não precisa da definição das tabelas e dos tipos de dados. Desta forma é possível adicionar qualquer tipo de dados que seja pretendido. Foi escolhido este tipo de base de dados por ser possível adicionar diretamente classes, fazendo um bom complemento pois assim é possível guardar as instâncias das entidades diretamente na base de dados. Para utilizar esta tecnologia foi necessário adicionar uma dependência nas dependências do pom.xml do Maven e possuir uma conta de serviço na Google. Como foi referido anteriormente não foi possível conectar a apresentação aos serviços e por isso o teste

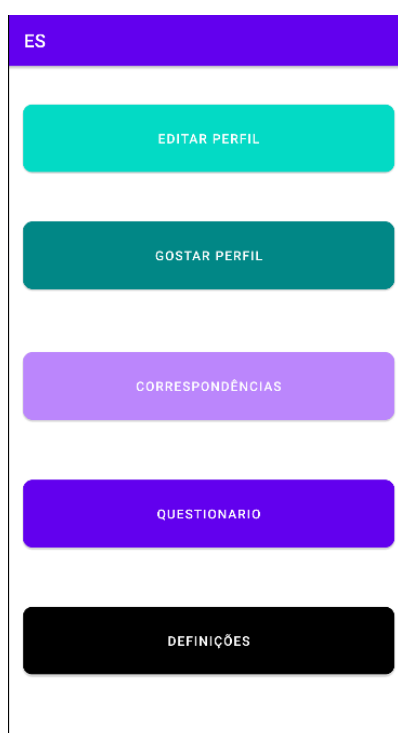


do acesso a dados no Firestore foi feito através do protótipo de teste ao domínio.

Em relação ao domínio este não foi alterado do teste para o modelo aplicacional.

Em termos de avaliação da utilização desta aplicação, imaginando que seria possível conectar a apresentação ao domínio esta teria uma avaliação média. A interface gráfica está simples, mas ainda falta implementar muitas partes da aplicação para que este fosse possível de utilizar por um utilizador comum, nomeadamente a constituição do perfil.

As interfaces gráficas obtidas foram:



**Figura 9 - Página principal**



**Figura 8 - Vista atribuir gosto ou não gosto**

ES

Tu costumavas fazer amigos regularmente?


MUITO POUCO

NEM MUITO NEM POUCO

MUITO FREQUENTEMENTE

**Figura 11 - Vista responder ao questionário**

ES



Maria, 23, 60km

Um exemplo de texto so para ver como fica com alguns caracteres, por isso ainda tem de ser um texto relativamente grande

Classificação

SUBMITER

**Figura 10 - Vista atribuir classificação ao utilizador**

## 5. Análise crítica do projeto realizado

Os pontos fortes da aplicação foi a constituição de uma boa arquitetura para ser implementada. Isto observou-se na altura da implementação, onde o código das diferentes classes existentes foi fácil de implementar. Isto surgiu tanto na implementação do protótipo de teste como no protótipo aplicacional. Na verdade, o que se fez foi traduzir os diagramas UML para código Java. Na altura da implementação quase não foram precisos ajustes à arquitetura. Um outro ponto forte foi o facto de não ter sido preciso alterar nada na camada de domínio para implementar os protótipos aplicacionais. Sendo por isso um bom indicador que foi feita uma boa arquitetura.

Um dos pontos fracos começou logo no modelo de casos de utilização, onde existiram casos de utilização com demasiados *includes* e *extends*. Isto complicou muito a essência do caso de utilização. De seguida foi a escolha de precisamente os casos de utilização mais complexos para desenvolver, resultando no aumento exponencial de tempo preciso para desenvolver estes casos. Isto resultou também no aumento de tempo necessário para implementar todos os pontos especificados. Colmatando depois na falta de tempo para conseguir conectar a apresentação ao domínio. Um outro ponto fraco foi o modelo de domínio ter ficado pouco rico em termos de conhecimento das entidades.

Uma forma de resolver o problema dos casos de utilização muito extensos seria isolar o caso de utilização das restantes componentes do qual faz *includes* e *extends*. Ficando assim um caso de utilização mais simples e fácil de desenvolver e compreender.

A resolução para o problema do modelo de domínio seria refactorizar este até ser possível observar de facto as entidades de domínio e a partir destas constituir os diagramas de sequência.

### 3. Conclusões

Em suma, com a realização deste projeto foi possível adquirir os seguintes conhecimentos:

- Conhecer as diversas formas de fazer a análise de requisitos.
- Conhecimento sobre o que é um documento de visão e de que forma este ajuda na análise de requisitos.
- Utilização do modelo *top-down* dos casos de utilização para obter os objetivos pretendidos na aplicação.
- Formação de uma ideia concreta sobre o caso de utilização através da descrição detalhada.
- De que forma a especificação suplementar ajuda a estabelecer as restrições da aplicação.
- Utilidade de um glossário para definir termos comuns a todos os envolvidos no projeto, para retirar ambiguidade nos objetivos pretendidos.
- Conhecer o que é a arquitetura lógica e de que forma esta é útil para o posterior desenvolvimento de software.
- Utilidade do UML no desenvolvimento de software, nomeadamente para desenvolver software independente da plataforma onde é executada.
- De que forma o modelo de domínio na arquitetura lógica ajuda a perceber o problema e a constituir as entidades do domínio.
- A utilidade dos diagramas de sequência para ver como cada caso de utilização interage ao longo do tempo.
- Os diagramas de classes e a sua utilidade para ver as interações entre as classes e a sua constituição.
- A arquitetura geral da solução para observar como o sistema se comporta como um todo.
- O conhecimento sobre o que é a arquitetura detalhada e como esta define as plataformas aplicacionais utilizadas.

- O que é um modelo de dinâmica e de que forma este ajuda a implementar código que varia ao longo do tempo e do estado de forma mais simples e eficiente.
- O que é a arquitetura de teste e de que forma esta ajuda a testar a aplicação na camada de domínio da aplicação.
- O que é o modelo de implantação e para que serve.
- O mais importante, como desenvolver código simples e fácil de perceber.
- De que forma as boas práticas de engenharia de software ajudam a produzir, testar e evoluir o software com o mínimo de complexidade possível.

## 4. Bibliografia

[1] Textos de apoio, Engenharia de Software, Luís Morgado, 2021

[https://en.wikipedia.org/wiki/Software\\_engineering](https://en.wikipedia.org/wiki/Software_engineering)