INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores

CROSSWORKING

DAVID PEREIRA LOURENÇO

LOURENÇO GONÇALVES VALA

Relatório realizado no contexto de Projeto e Seminário,

Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2021-2022

Orientador: João Humberto Holbeche Trindade

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia Eletrónica e Telecomunicações e de Computadores

CROSSWORKING

DAVID PEREIRA LOURENÇO

_____

LOURENÇO GONÇALVES VALA

_____

Relatório realizado no contexto de Projeto e Seminário,

Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2021-2022

Orientador: João Humberto Holbeche Trindade

_____

Resumo

O desenvolvimento da espécie humana até à atualidade deve-se em grande parte, a ideias que algumas das pessoas mais importantes da nossa história, conseguiram colocar em prática devido aos conhecimentos que possuíam.

No contexto atual, em que cada individuo se especializa numa área de estudos em detrimento de tantas outras, a prospeção para que uma pessoa com ideias concretizáveis, rentáveis, e que muitas vezes poderiam mudar o rumo da evolução da sociedade, não consiga colocar a sua ideia em prática devido a falta de conhecimentos nessa área em específico é elevada.

Como forma de mitigar o problema descrito, foi desenvolvida uma plataforma, que tem como objetivo proporcionar a troca de ideias entre utilizadores. Os utilizadores têm oportunidade de registar as suas ideias bem como os seus conhecimentos, conhecimentos estes que ficaram associados ao seu perfil, por forma a proporcionar ao utilizador uma lista de ideias que se enquadrem com as suas aptidões. Posteriormente é oferecida a possibilidade de um utilizador se candidatar à realização de uma ideia, candidatura esta que deverá ser aceite pelo proponente, para que a ideia fique atribuída e os seus contactos sejam trocados.

A concretização desta plataforma foi desenvolvida uma *Web API* (*Application Programming Interface)*, alojada num servidor com interface *HTTP*, a qual realiza a manipulação dos dados necessários ao correto funcionamento da plataforma. Para disponibilização de interface gráfica ao utilizador foi decidida a utilização da framework *Android*. A persistência dos dados foi garantida com recurso a uma base de dados relacional, e a um serviço de armazenamento de objetos.

Apesar da escolha da framework *Android*, a construção da *Web API* teve em consideração a possível expansão da plataforma para outras frameworks *front-end*.

Abstract

Up until now the development of humanity was largely due to the ideas of some of the most important people in our history and their ability to put these ideas to practice with their knowledge.

In today's age, most people choose to specialize in a particular field whilst neglecting the rest. This makes it particularly difficult for them to put into practice easy and rentable ideas, which could change society.

To fight this issue a platform was designed for people to be able to share their ideas, their knowledge, and their skills. The users of this platform can save their ideas and skills in their profile, so the ideas shared to them are related to their expertise. Afterwards they can apply to one of these ideas, and should they be accepted by the creator, they will enter in contact to commence development.

For this platform a *Web API* (*Application Programming Interface*) was developed and deployed in a *HTTP* server, to manipulate the data required for this service. As for the user interface (UI), the *Android* Framework was chosen. For the data to be persistent, a relational database was created, as well as an object storage service.

Although the UI chosen for this platform was an *Android* application, the *Web API* was developed with consideration for future expansion into other types of *front-end*.

Índice

Listagem de Figuras

Lista de Listagens

1 Introdução

No início da espécie humana, cada individuo existia de forma singular, sem existência de uma comunidade. Neste paradigma, era essencial possuir o maior número de conhecimentos para garantir a sua sobrevivência [1]. Com o nascimento das primeiras comunidades, começou a constatar-se uma divisão de conhecimentos, onde cada individuo continuava a apresentar capacidade de viver isolado, mas, conseguia, em comunidade adquirir conhecimentos mais específicos sobre a sua tarefa, fosse esta a coleta, a caça ou a domesticação entre outras atividades.

Com o avançar dos séculos e da evolução, a necessidade de viver em comunidade, ser parte integrante desta, e garantir o seu contributo para a mesma aumentou de forma exponencial. A especialização de cada individuo em determinada área foi elevada ao seu máximo expoente, através das áreas do ensino secundário e as licenciaturas, mestrados e doutoramentos nos Politécnicos e Universidades.

1.1 Problema

A elevada especialização de toda a sociedade numa área especifica levou a elevadas discrepâncias de conhecimentos entre a área de estudos e todas as restantes. Existem hoje indivíduos extremamente cultos em determinadas áreas, mas que apresentam deficiência no conhecimento em tantas outras.

No entanto, a falta de conhecimento sobre múltiplas áreas, não se traduz num desinteresse geral sobre as mesmas, o que leva a que muitas vezes seja possível teorizar uma ideia que poderia levar a uma melhoria efetiva nessa mesma área, mas devido a falta de conhecimento específico que essa ideia viria a necessitar, a mesma é descartada.

Constata-se de igual forma que, existem indivíduos extremamente especializados que não conseguem ter ideias relativas à sua área de estudos suficientemente sustentadas, para que possam empregar o seu tempo e conhecimento.

Por forma a colmatar este cenário, foi idealizada uma plataforma onde é possível trocar ideias e conhecimentos, permitindo juntar no mesmo ambiente pessoas com ideias inovadoras e pessoas com conhecimentos técnicos para as desenvolver. Surge assim a plataforma *CrossWorking*.

1.2 Objetivos

Precedentemente ao desenvolvimento da plataforma, e uma vez detetado e avaliado o problema, foram traçados os objetivos essenciais ao funcionamento da plataforma, e que se esperava cumprir para que o problema identificado pudesse ser mitigado. Pretendia-se assim cumprir os seguintes objetivos com o desenvolvimento da plataforma:

- Partilha de ideias.
- Partilha de aptidões e conhecimentos.
- Incentivo à inovação e a cooperação.
- Instigação à criação de equipas multidisciplinares
- Melhoria nas capacidades descritivas dos utilizadores, bem como na capacidade de comunicação com outros utilizadores de áreas de estudos díspares.

1.3 Enquadramento

Uma vez identificado o problema, foi realizada uma pesquisa por soluções existentes e alternativas à que se pretendia implementar. Desta pesquisa destacam-se duas soluções onde poderíamos encontrar alguns pontos em comum com a plataforma *CrossWorking*, sendo estas o *LinkedIn*, e a *Toptal*. No entanto, ambas as plataformas têm o seu foco na contratação de pessoas para determinada tarefa, o que não se coaduna com o objetivo da plataforma *CrossWorking*, uma vez que o seu objetivo não reside na contratação, mas sim na partilha e na entreajuda dos utilizadores.

Ainda durante a execução do projeto, os autores tomaram conhecimento de uma terceira plataforma denominada de *Fixando* que, mais uma vez, apresenta alguns pontos em comum com a solução *CrossWorking*, embora esteja presente o mesmo ponto de divergência face às anteriormente mencionadas, isto é, a existência de um contrato ou prestação de serviço. Um utilizador, na plataforma *Fixando,* procura profissionais para resolução de problemas, algo que a distancia da plataforma *CrossWorking* que proporciona um espaço para ideias e para pessoas que desejem realizar ideias que poderão ser revolucionárias.

Analisadas as soluções já existentes, que os autores tenham tido conhecimento, e uma vez comparadas com a implementação que viria a ser levado a cabo, foram reconhecidas as diferenças e considerou-se que as mesmas seriam motivação suficiente para avançar com o desenvolvimento da solução *CrossWorking*, uma vez que existe necessidade e espaço no mercado para a existência desta mesma plataforma.

2 Análise do Problema

Neste capítulo são apresentados os conceitos fundamentais para a correta compreensão da solução realizada, assim como a descrição dos elementos que a compõem. São detalhados os intervenientes na plataforma, a informação que os descreve e os identifica, e as ações que cada interveniente tem ao seu dispor.

2.1 Conceitos fundamentais

Uma ideia é o imaginar de um conceito, por um utilizador, que poderá ter uma concretização quando aplicado, mas que, pelos mais variados motivos, não é possível, ao criador da ideia concretizá-la. Para a plataforma *CrossWorking* representa o elemento central do seu funcionamento; os utilizadores registados têm a possibilidade de propor ideias, ou candidatar-se a ideias já propostas, por outros utilizadores.

Um utilizador é uma qualquer pessoa que se encontre registada na plataforma e utiliza a mesma, podendo assumir o papel de proponente ou candidato de uma ideia. Um candidato é um qualquer utilizador que se propõe a implementar, ou a ser parte integrante da solução de uma ideia; e os proponentes, utilizadores que possuem ideias, partilhadas na plataforma, e que aguardam que outros utilizadores se candidatem às mesmas.

Uma candidatura representa esta proposta por parte do candidato ao proponente. Os proponentes ao receberem uma proposta, poderão responder e estabelecer contacto com o candidato.

Uma competência (por vezes denominada pelos autores de *skill*, importado do idioma inglês) é um conhecimento que determinada pessoa possui e que lhe permite realizar uma atividade ou um conhecimento que o proponente da ideia considera necessário para a concretização da mesma.

2.2 Ideias

Para a plataforma CrossWorking, a ideia é o elemento central do seu funcionamento. As ideias são compostas por:

- um título, que deve ser sugestivo do conteúdo da mesma e composto por não mais de cinco palavras;
- uma pequena descrição, que apresenta algum detalhe sobre o conteúdo da ideia sem entrar em grande elaboração sobre a mesma;
- uma descrição, onde, dentro dos conhecimentos do proponente, se apresenta o detalhe extensivo do conceito da ideia, das suas motivações, objetivos, entre outros pontos relevantes à concretização desta;
- uma lista de competências associadas, com vista a indicar aos utilizadores que se queiram candidatar à ideia, quais as competências que estes devem dominar.

Quando listadas, as ideias apresentam uma opção para que um utilizador se possa candidatar.

2.3 Competências

As competências são conhecimentos, habilidades ou aptidões que cada individuo possui, e se sente capaz de utilizar para concretizar ideias. Devido ao tipo de conhecimento que se espera de um utilizador, quando este assume possuir uma determinada competência, existe uma limitação no número de competências possíveis para um único utilizador. No contexto de plataforma, assume-se que um utilizador não poderá possuir conhecimento especializado em mais de dez competências. Nas ideias existe também uma limitação no número de competências necessárias à concretização da mesma, não permitindo assim a associação de mais de vinte competências a uma ideia.

Uma competência é agrupada numa categoria geral, que representa a área de conhecimento onde se insere; e é caracterizada por:

- um título, que representa o conhecimento que se pretende oferecer aos utilizadores;
- uma pequena descrição, que servirá para informar a forma como o conhecimento foi adquirido, bem como a sua experiência na utilização do mesmo.

2.4 Utilizadores

Um utilizador é qualquer individuo que se encontre registado na plataforma e possua uma conta ativa.

Os utilizadores caracterizam-se por:

- um endereço de correio eletrónico, essencial para a autenticação e o contacto entre utilizadores;
- um nome;
- uma imagem de perfil;
- uma descrição, onde o utilizador deve introduzir um texto que o descreva tanto ao nível profissional como também pessoal, uma vez que poderá servir de critério de seleção na sua candidatura às ideias;
- uma lista de competências que o utilizador domina, e se predispõem a utilizar para a concretização de ideias.

A cada utilizador existe associada uma lista de ideias que este propôs, onde é possível, para além de editar cada ideia, verificar os candidatos às suas ideias e entrar em contacto com os mesmos após aceite a candidatura. Existe ainda uma lista de resultados onde é possível, ao utilizador, verificar o estado das candidaturas que enviou.

2.4.1 Proponente

Proponentes são utilizadores que partilham ideias na plataforma CrossWorking, com o intuito de que algum outro utilizador, candidato, possua as competências necessárias para as concretizar.

Um proponente em nada difere de um utilizador na sua caracterização concreta, existindo apenas uma distinção na terminologia, por forma a especificar o tipo de intervenção de um utilizador, num determinado momento.

2.4.2 Candidato

Candidatos são utilizadores que se dispõem a concretizar uma ideia, por possuírem as competências necessárias para a sua concretização, e por essa ideia lhes despertar interesse.

2.5 Candidaturas

As candidaturas representam a proposta de um candidato para o proponente de uma ideia específica.

Uma candidatura possui a informação do utilizador candidato, tal como a ideia a que este se propôs, mas também um atributo que representa o seu estado. Uma candidatura pode estar pendente, aceite ou recusado, dependendo da escolha do proponente da ideia. A qualquer momento um candidato que se encontre no estado pendente poderá anular a sua candidatura e deixar assim de ser considerado candidato a uma ideia.

Quando um candidato é aceite, existe uma troca de contactos entre este e o proponente.

Um utilizador pode ser um proponente de uma ideia, e não pode candidatar-se a esta; no entanto pode candidatar-se a uma outra ideia não sua.

3 Descrição da Solução

Como resposta ao problema encontrado, e tendo por base a análise do problema, foi implementada a plataforma *CrossWorking* e, consequentemente, os módulos que compõem esta mesma solução. Foi também delineada a interação entre os módulos.

3.1 Estrutura Geral

A plataforma *CrossWorking* é composta por uma aplicação *back-end* e uma aplicação front-end.

O *back-end* da plataforma *CrossWorking* encontra-se dividido nas seguintes partes:

- *Web API*.
- Serviço de autenticação.
- Base de dados relacional.

Existindo a possibilidade de crescimento desta plataforma, partiu-se do princípio de que o servidor poderia vir a servir múltiplas frameworks de *front-end*, como tal foi adotada uma arquitetura que viabiliza a existência de múltiplas implementações com diferentes opções de front-end. Como seria inviável realizar a implementação com múltiplas frameworks, foi escolhida pelos autores a framework *Android* como opção a desenvolver, tendo sido tomado como critério de escolha a familiarização dos respetivos criadores com a referida framework. Os autores consideraram ainda que esta framework seria a indicada para mostrarem os seus conhecimentos. Inerente a todos estes fatores é também necessário ter em consideração o universo *Android* que segundo *Counterpoint Research, Gartner, IDC* [2] atingiu em 2021 o marco de 2,5 mil milhões

de utilizadores, representando 71.3% da quota de mercado dos sistemas operativos para dispositivos móveis, segundo dados da *statcounter GlobalStats* [3].

Foi ainda ponderada a possibilidade da criação de uma aplicação web, no entanto, não se procedeu ao desenvolvimento da mesma, pois não existia o tempo necessário para a sua correta implementação no âmbito da unidade curricular Projeto e Seminário.

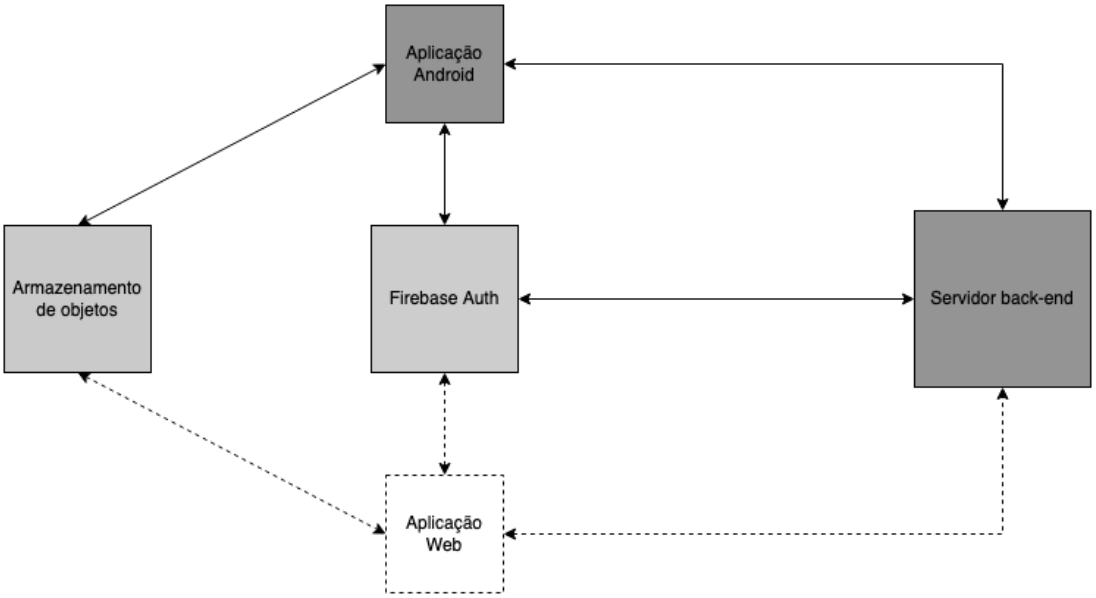

Figura 1 - Diagrama de blocos da plataforma

3.2 Back-end

O servidor da plataforma *CrossWorking* foi desenhado por forma a sustentar o funcionamento de uma aplicação de *front-end*, independentemente da tecnologia utilizada por esta. Por forma a garantir este desacoplamento, o servidor expõe uma *API* (*Application Programming Interface*), através do protocolo *HTTP*.

A *API* exposta pelo servidor permite pedidos e respostas em formato application/json, o que garante que qualquer aplicação *front-end* que seja capaz de interpretar dados neste formato possa usufruir dos recursos disponibilizados pela *API*. O acesso aos recursos expostos pelo servidor, é possível mediante a apresentação de um token de acesso, o qual também permite implementar uma política de controlo de acessos.

A implementação do servidor recorre à framework *Spring* [4] para:

- Gestão de dependências.
- Implementação da arquitetura *MVC*.
- Utilização dos componentes, existente na framework *Spring*, para a implementação de aplicações *web*.

Para a persistência dos dados tratados, pelo servidor, optou-se por uma solução apoiada em três vertentes: a flexibilidade dos tipos de dados tratados, sem que se perdesse a consistência implícita de um modelo relacional; a segurança dos utilizadores da plataforma; e a disponibilidade e custo das soluções. Com base nestes três princípios, desenhou-se uma solução tripartida. Foi desenhado um modelo de dados relacional, o qual foi aplicado numa base de dados relacional

PostgreSQL, e onde se pretende guardar os dados dos utilizadores, das suas ideias, *skills*, entre outros. É utilizada a plataforma externa *Firebase* para proceder à autenticação dos utilizadores, o que permite garantir uma maior segurança dos mesmos. Também se prossupõe a existência de um serviço de armazenamento de imagens, no entanto a escolha do serviço a utilizar, assim como a sua implementação fica a cargo de quem implementa a aplicação *front-end*. Esta opção foi tomada com o objetivo de reduzir os custos associados à hospedagem da plataforma e do serviço de armazenamento de objetos. Também permite que não exista qualquer dependência da plataforma para com o serviço que armazena as imagens. Apenas deve ser garantido que o serviço utilizado permite o acesso às imagens através de um *URI* (*unique resource identifier*).

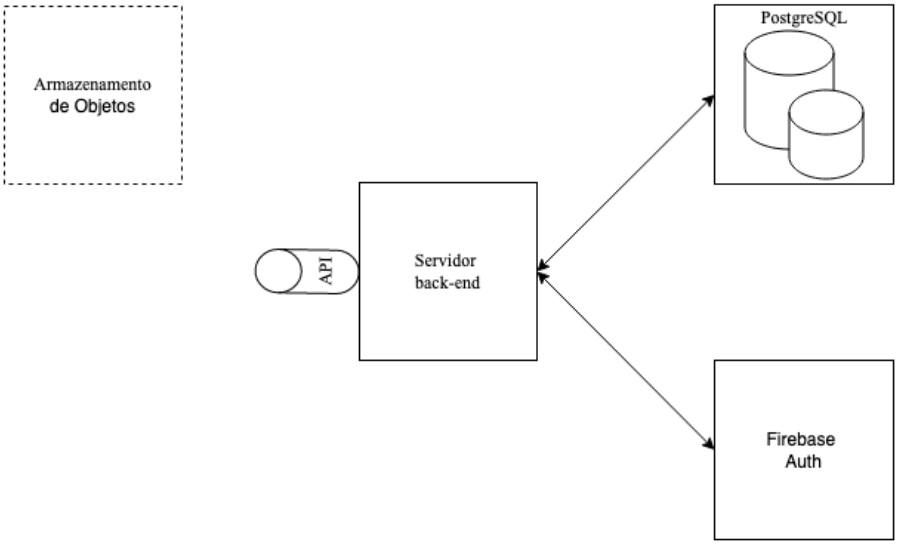

Figura 2 - Diagrama de blocos back-end

3.3 Front-End

A aplicação de *front-end* foi desenhada por forma a disponibilizar todas as funcionalidades implementadas no servidor, e expostas pela *API*, aos utilizadores através de uma interface gráfica, e numa framework que abrangesse um grande número de possíveis interessados.

Fez-se assim uma opção por desenvolver uma aplicação para a framework *Android*, não só pelo número de utilizadores que esta framework possui, mas também por ser uma tecnologia que agrada bastante os criadores da plataforma *CrossWorking*, e com a qual estes se encontram já bastante familiarizados devido ao seu estudo no contexto da licenciatura.

Por forma a seguir todas as indicações da *Google* [5], desenvolvedora da framework Android, para novas aplicações móveis, toda a aplicação encontra-se implementada com recurso ao mais recente toolkit desenvolvido para a criação de interfaces gráficas nativas em *Android*, o *Jetpack Compose*. Para poder desenvolver esta aplicação, foi necessário antes de qualquer implementação, debater e selecionar o padrão de desenho que permitiria criar uma aplicação estável, escalável e que cumprisse todos os requisitos de uma aplicação para a framework *Android*. Assim, foi selecionado o padrão *Model-View-ViewModel* (*MVVM*) que permite um desacoplamento das partes constituintes de uma aplicação (os dados, o comportamento e as vistas); além desse desacoplamento, é também garantida na documentação da framework um correto funcionamento deste padrão de desenho com o desenho da própria framework, e com o seu lifecycle [6]; sobre este padrão *MVVM* foi ainda adotada a arquitetura *Clean*, que pretende

criar uma maior separação, em camadas, do código implementado, aumentando a reutilização dos componentes da solução. Associado a este padrão encontra-se ainda presente um outro padrão de desenho, o repositório, que foi utilizado para encapsular o acesso a dados, permitindo uma interface uniforme para o exterior, mesmo que na sua implementação sejam acedidos múltiplos fornecedores de dados.

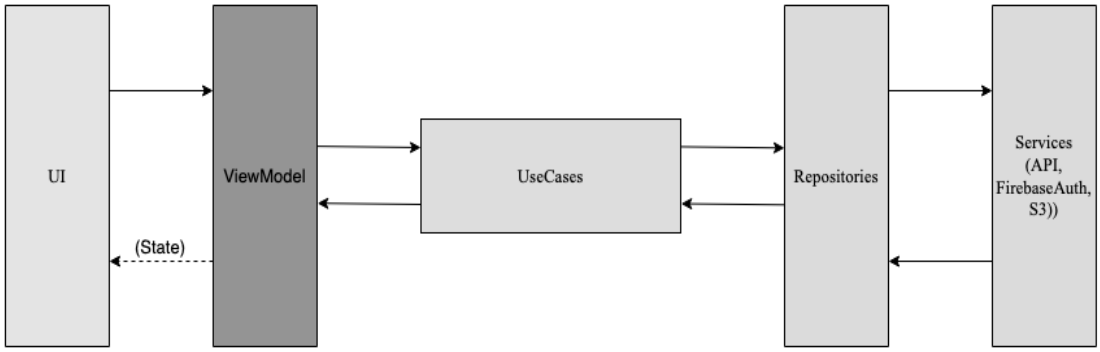

Figura 3 - Diagrama arquitetura aplicação android

Tal como referido anteriormente foram implementados os mecanismos de autenticação com recurso ao serviço *Firebase Authentication*. Este serviço permite delegar ao *front-end* as operações de criação de conta e início de sessão, e posteriormente gerar um token utilizado no processo de autenticação no servidor *back-end*. A escolha deste serviço permitiu tornar as operações anteriormente mencionadas mais imersivas, adotando o mesmo estilo e arquitetura visual da restante aplicação; ao contrário de outras soluções que realizam a autenticação em páginas de terceiros, onde não existe opção de customização estética. Por outro lado, é acrescida uma dependência a um utilizador da *API CrossWorking*, uma vez que este necessita de implementar a autenticação, e esta se encontra restrita ao fornecedor de serviço *Firebase Authentication*.

Por fim, foi também anteriormente mencionado o serviço de armazenamento de objetos, e a implementação do acesso ao mesmo no *front-end*. Esta opção de desenho permite diminuir o número de transferência de dados entre a totalidade da plataforma, o que, no cenário atual, onde são utilizados serviços cloud, representa uma diminuição de custos. Assim é responsabilidade de quem implementa o serviço front-end selecionar o fornecedor de serviço indicado para si, e implementar o carregamento de imagens para esse serviço, gerando um *URI* para a imagem carregada no serviço. O serviço de carregamento de imagens apenas deve aceitar carregamento de objetos para aplicações autenticadas; no entanto a leitura dos objetos deve ser publica. Uma vez na posse do *URI* o mesmo deve para a *API CrossWorking*, para que fique associado ao utilizador que realizou o carregamento da imagem.

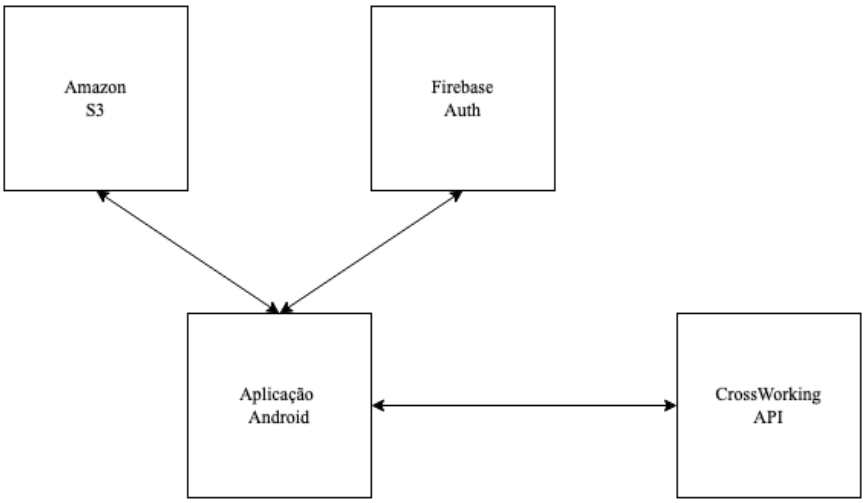

Figura 4 - Diagrama de blocos aplicação android

4 Infraestrutura

Neste capítulo é descrita a infraestrutura da plataforma *CrossWorking*, os serviços de sustentam o seu funcionamento e a interoperabilidade entre estes serviços.

É analisada a escolha do fornecedor de serviço, e as soluções oferecidas por este das quais se tira partido para a plataforma *CrossWorking*

4.1 Hospedagem de serviços

Como desenvolvedores de uma plataforma sem infraestrutura física pré-existente, os autores depararam-se com a necessidade de recorrer a serviços cloud, para hospedar a base de dados relacional *PostgreSQL* e o servidor com a *Web API*. Para que um serviço *cloud* pudesse ser considerado como viável, necessitava de oferecer um serviço para hospedagem do servidor com a *Web API*, compatível com as tecnologias da mesma; e uma base de dados relacional *PostgreSQL*. Como critério de seleção foi também preciso ter em consideração o fator monetário, uma vez que, os autores não possuíam qualquer tipo de fundo monetário para o desenvolvimento desta plataforma.

Tendo todos estes fatores em consideração, optou-se pelos serviços oferecidos pela *AWS*, uma vez que permitia acesso gratuito a um serviço de hospedagem (denominado pela *AWS* de *Beanstalk*) e a uma base de dados relacional na qual existia um máximo no armazenamento de 20GB (serviço que a *AWS* denomina de *RDS*).

4.2 Firebase Authentication

Tal como a grande maioria das aplicações existentes no dia de hoje, também na plataforma *CrossWorking* existe a necessidade de identificar o utilizador. Esta identificação irá permitir não só uma estruturação dos dados, mas também aplicar uma política de acesso aos mesmos baseada no utilizador.

Como critérios de seleção de um serviço de autenticação foi tido em conta a reputação desse serviço de autenticação junto da comunidade; a compatibilidade com um grande número de plataformas; os tipos de autenticação suportados (*Google*, *Facebook*, entre outros) e também a possibilidade de customização estética da vista de criação de conta e início de sessão.

Tidos em conta os critérios, os autores concluíram que o serviço que melhor se enquadraria na plataforma seria o *Firebase Authentication*, isto porque, não só assume uma boa reputação junto da comunidade e suporta os mais diversos tipos de autenticação, como também, de entre todos, é aquele que permite a customização total da vista onde é criada a conta e realizado o inicio de sessão, uma vez que este serviço espera que seja o utilizador a implementar a vista, e a realizar a chamada às funções de autenticação.

Para além destes critérios, este serviço ainda se destaca por ser gratuito até ao limite de 10 000 utilizadores, pela sua consola de gestão de contas, e por possibilitar um serviço de notificações associado às contas dos utilizadores.

4.3 Amazon S3

A identificação de um utilizador não fica garantida apenas com a sua informação escrita, é essencial para os criadores da plataforma a possibilidade de um utilizador apresentar uma imagem de perfil à sua escolha. A utilização de imagens juntamente com a restante informação de um utilizador permite aumentar a confiança de todos os utilizadores da plataforma, fator essencial quando se trata de uma plataforma inovadora.

Os critérios para seleção do serviço são:

- um serviço gratuito, pelos mesmos motivos apresentados no ponto 4.1 Hospedagem;
- interface de acesso compatível com a framework *Android*;
- disponibilização um caminho para a localização da imagem, uma vez feito o carregamento;

Uma vez que já estão em uso alguns serviços da *AWS*, e existindo um serviço denominado de *S3*, que cumpria na totalidade os critérios anteriores, foi tomada a opção de utilizar esse serviço na plataforma *CrossWorking*. O serviço da *AWS* na modalidade gratuita tem disponível 5GB de armazenamento, valor que não sendo muito elevado, é suficiente para a fase de desenvolvimento.

5 Implementação do back-end

No capítulo que se segue é descrita a estruturação da base de dados relacional, a implementação do servidor *back-end* da plataforma *CrossWorking*, bem como a utilização do serviço de autenticação para validação do identificador do utilizador. É apresentada com detalhe a arquitetura utilizada no desenvolvimento do servidor, bem como na gestão do acesso à base de dados, onde se incluem tópicos como o isolamento das operações.

É também neste capítulo apresentada a interação esperada com o servidor, isto é, os recursos que o servidor disponibiliza, o formato dos pedidos ao servidor, o formato das respostas fornecidas pelo mesmo e o tipo de erros que poderão ocorrer.

5.1 Persistência de dados

A base de dados tem como responsabilidade o armazenamento de todos os dados da *Web API*. Este domínio engloba os dados pessoais dos utilizadores, das ideias criadas na plataforma e das interações existentes entre estes.

Para persistência dos dados foi escolhido o sistema *PostgreSQL*. O *PostgreSQL* é um sistema de base de dados relacional por objetos que estende a linguagem *SQL*. Como fatores que levaram à escolha deste foram tidos em conta os seguintes pontos: o *PostgreSQL* é uma tecnologia *open-source*, apresenta uma maior compatibilidade com os serviços de hospedagem que foram considerados durante a conceção inicial do projeto, possui um grande número de tipos nativos ao sistema que não o só tonam mais consistente, como também diminuem a complexidade no mapeamento da informação da base de dados em objetos de domínio.

O modelo relacional que foi desenhado para esta base de dados está representado na figura abaixo.

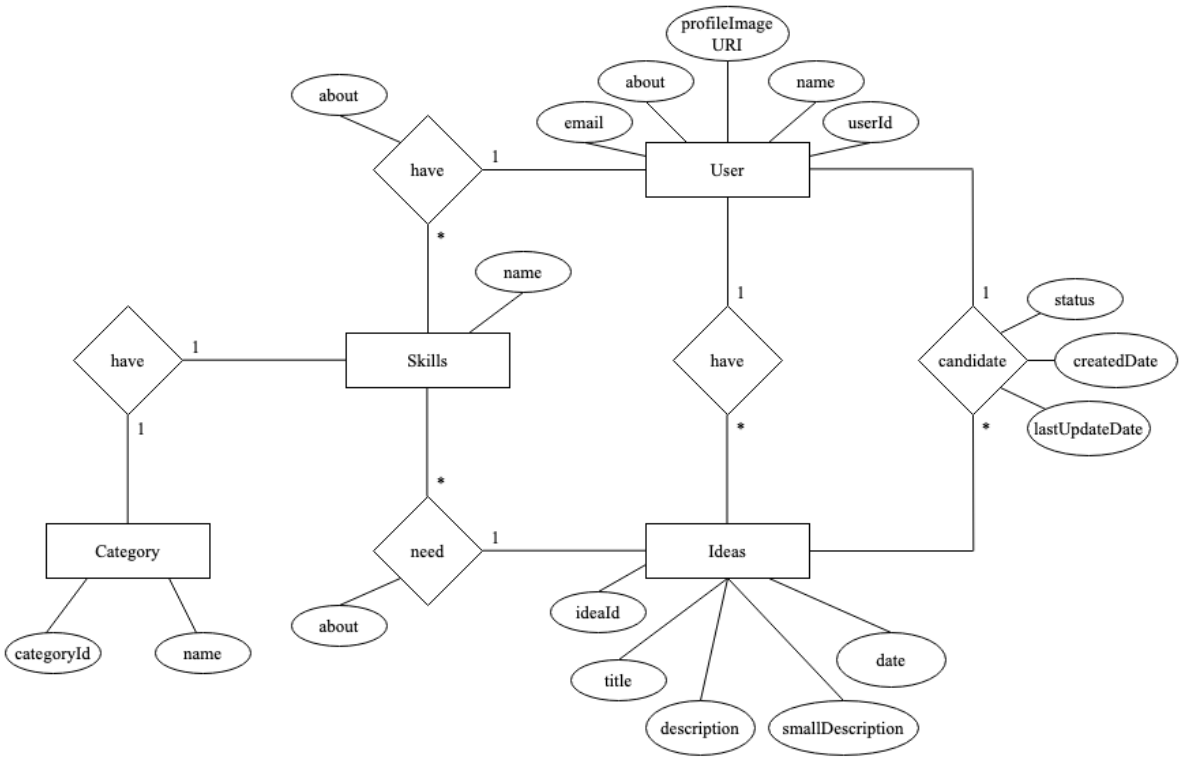

Figura 5 - Modelo Relacional Plataforma CrossWorking

Inicialmente foi definida a entidade *USER*. Esta entidade representa um utilizador, o qual possui:

- um identificador, estabelecido pela *Google Firebase Authentication* quando a conta é criada;
- um nome;
- um email;
- uma descrição do perfil;
- um *URL* para o *blob* com a imagem do utilizador, guardado na *AWS S3*.

Devido à utilização da *Google Firebase Authentication*, não existe necessidade de guardar dados sensíveis do utilizador para além do seu email, que é usado na troca de contactos após um utilizador ser aceite para uma ideia.

Definidos os utilizadores, procedeu-se ao desenvolvimento da entidade *IDEA*, que representa as ideias propostas pelos mesmo utilizadores, composta por:

- um identificador único, gerado pela própria base de dados;
- o identificador do utilizador que a criou;
- um título;
- uma descrição pequena;
- uma descrição detalhada;
- a data de criação.

Com estas entidades concluídas, foi desenhada a entidade *CATEGORY*, que corresponde a uma categoria, na qual as *skills* serão inseridas de forma a agrupá-las. Uma categoria é composta por um identificador único e um nome.

Seguiu-se a criação da entidade *SKILL*, que representa as competências que os utilizadores têm, e também competências que são pedidas nas ideias. Esta entidade é composta por um identificador único, um nome e o id da categoria correspondente.

Tendo as *skills* desenhadas, avançou-se para as, associações *USERSKILL* e *IDEASKILL*, que associam skills a utilizadores e ideias, respetivamente, através dos seus identificadores. Ambas as relações têm ainda uma descrição, que possibilita a apresentação de mais algum detalhe referente à respetiva *skill*.

Por fim existe ainda a associação *CANDIDATE*, que representa a candidatura de um utilizador a uma ideia, composta pelos identificadores destes e um estado da candidatura.

5.2 Web API

Uma *API*, *Application Programming Interface* é um intermediário de software que permite que dois módulos de software distintos comuniquem entre si. No caso específico de uma *Web API*, trata-se de um intermediário disponível via protocolo *HTTP* e que está a ser executado num sistema que atende a pedidos *HTTP* com uma disponibilidade constante.

O servidor da plataforma *CrossWorking*, expõe através do protocolo *HTTP* uma *Web API* que permite o acesso à informação e comportamento do servidor, de forma desacoplada. A *API* da plataforma *CrossWorking* suporta pedidos em formato *application/json*, e responde a esses mesmos pedidos também em formato application/json, com a exceção de situações de erro ou falha onde a resposta é fornecida no formato *application/problem+json.* Não existe preservação de estado ao realizar um pedido, não existindo também nenhuma funcionalidade que requeira cookies para manutenção de estado. Os recursos são identificados pelo seu *URI* e no caso dos recursos com paginação existem disponíveis dois *query* parameters; *pageIndex*, que permite identificar o número da página que se pretende aceder; e *pageSize*, que indica o número de valores que cada página deve apresentar. A autorização dos utilizadores, bem como o controlo de acesso aos recursos é inteiramente realizado através do cabeçalho *Authorization*, o qual deve conter um *bearer token* válido no serviço *Firestore Authentication*.

O conteúdo dos pedidos realizados à *API*, assim como o conteúdo da resposta enviada por esta, são objetos *JSON*, com significado na plataforma *CrossWorking*. A totalidade dos objetos aceites na realização de pedidos, assim como de objetos de resposta, encontra-se detalhada na documentação da *API* [7], para que seja possível a utilização da mesma por qualquer utilizador.

Era objetivo dos criadores da plataforma *CrossWorking* a disponibilização de respostas onde o formato do corpo fosse *application/vnd.siren+json* [8], para alem do formato *application/json*. No entanto, após discussão entre os autores, foi tomada a decisão de apenas implementar respostas em formato *application/json*, uma vez que este formato permitiria a utilização de bibliotecas já existentes para a realização do mapeamento do objeto *JSON* para objetos de domínio; e também, devido ao contexto em que o projeto se inseria, e ao intervalo disponível para a sua implementação.

Apesar de apenas disponibilizar respostas *HTTP* com o corpo no formato *application/json*, foi acautelada a possibilidade de uma posterior implementação de respostas com

outro formato. Para tal o servidor foi construído de acordo com o padrão de desenho *Model-View-Controller* (*MVC*). Este padrão separa a aplicação em 3 componentes, o modelo, responsável por toda a lógica da aplicação, a apresentação, que trata da disponibilização dos componentes ao utilizador e o controlador que liga estas duas partes. O servidor ficou composto por cinco partes:

- *Data Access Objects* (*DAOs*) – Interfaces que disponibilizam os contratos de acesso à base de dados relacional.
- *Models* – Objetos do domínio do servidor portadores da informação para todas as partes do sistema à exceção dos *controllers*.
- *Services* – Responsáveis pela lógica de negócio, são chamados pelos *controllers* e chamam os métodos das *DAOs* para obterem dados da base de dados.
- *Controllers* – Responsáveis por atender os pedidos feitos à API e chamar o método conversor dos *DTOs* em objetos de domínio do servidor (*models*), para que todas as camadas inferiores apenas tenham conhecimento dos objetos de domínio.
- *Data Transfer Object* (*DTOs*) – Objetos usados na comunicação entre o cliente e o servidor, através da *API.* Estes objetos apresentam uma estrutura fixa, e permitem garantir que, mesmo que ocorra alteração no modelo de dados, que implique uma alteração da estrutura dos objetos do domínio, a mesma não se propague até ao utilizador da *API.*

Os *DTOs* permitem garantir que a interface exposta se mantém uniforme a partir do momento em que a *API* é publicada. Este facto é de grande relevância pois qualquer tipo de alteração à interface exposta publicamente, poderia levar a que os utilizadores da *API* experienciassem erros.

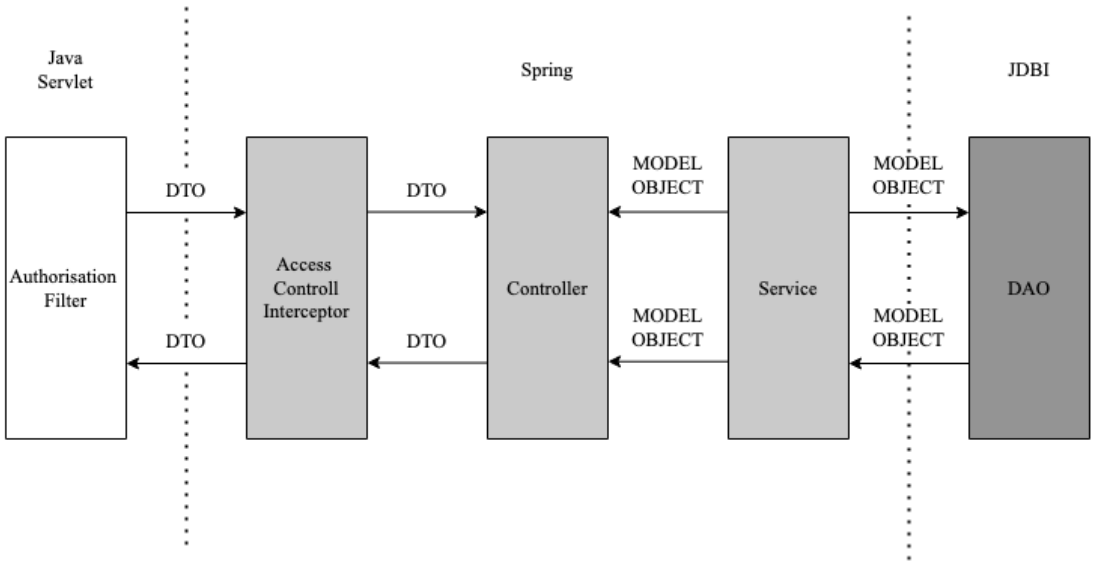

Figura 6 - Arquitetura aplicação front-end

A *API* foi desenvolvida usando a framework *Spring Boot* e a dependência *Spring Web MVC*, juntamente com a linguagem de programação *Kotlin* desenvolvida pela *JetBrains*. A escolha desta linguagem deteve-se na sua versatilidade, ao elevado grau de familiarização dos autores com a mesma, a possibilidade de conjugação dos princípios da programação funcional com programação orientada a objetos, e por fazer uso da *Java Virtual Machine* (abreviada para a sigla *JVM*), tornando a implementação compatível com um grande número de serviços de *cloud*. Uma vez eleita a linguagem, tornou-se evidente a escolha da *Framework Spring Boot*, uma vez que oferece um elevado número de serviços uteis. Para esta eleição foram também tidas em conta as seguintes vantagens ao utilizar *Spring Boot* com a dependência *Spring Web MVC*:

- promoção da construção de uma *API Restful*, seguindo o modelo *MVC*;
- injeção de dependências;
- anotações capazes de configurar objetos com papéis importantes, como um *@RestController*;
- servidor *HTTP* pré-contruído com atendimento de pedidos e interpretação do protocolo.

5.2.1 Acesso à base de dados relacional

O acesso do servidor aos dados necessários ao seu funcionamento é um ponto relevante da sua implementação. Este acesso deve cumprir um grande número de requisitos, tais como, eficiência no mapeamento, possibilidade de aplicação de transações, gestão de conexões, entre outros. Após analisadas múltiplas bibliotecas de acesso a dados, tomou-se como opção a biblioteca *JDBI*, uma vez que esta permite a gestão de conexões e o mapeamento dos *resultSet* em objetos de forma automática, assim como, juntamente com o package *org.springframework.jdbc.datasource*, permite a gestão de transações através da gestão da conexão à base de dados [9].

Para que o servidor pudesse efetivamente realizar o acesso à base de dados, foram então criadas as *DAOs*. Nestas interfaces foram declaradas as funções de acesso aos dados, onde cada função apresenta uma anotação, local onde é especificada a query *SQL* que deverá ser executada na chamada à função. São possíveis dois tipos de anotações, *@SqlQuery*, quando se pretende realizar uma interrogação a uma tabela da base de dados; ou *@SqlUpdate*, quando a ação pretendida envolve uma modificação dos dados (ex. *update*, *delete*, *insert*). Neste último caso, onde se pretende realizar uma modificação do estado da base de dados, é ainda possível anotar com *@GetGeneratedKeys*, levando a que a informação modificada seja retornada.

A biblioteca *JDBI* realiza o mapeamento dos valores obtidos na execução da *query SQL*, para objetos do tipo do retorno das funções declaradas na *DAO* que originaram a execução da *query*.

5.2.2 Rotas da API

A *Web API* da plataforma *CrossWorking* disponibiliza um conjunto de recursos, disponíveis através de rotas especificas, as quais permitem aos clientes comunicar com o servidor. Estas rotas encontram-se devidamente detalhadas na documentação da *API CrossWorking* [7]. Seguem-se os recursos disponibilizados pela *API*:

- Criar um utilizador.
- Obter lista de utilizadores.
- Obter um utilizador através do seu identificador.
- Editar um utilizador.
- Apagar um utilizador.
- Criar uma ideia.
- Obter lista das ideias mais recentes.
- Obter ideia específica.
- Obter lista de ideias de um utilizador.
- Editar uma ideia.
- Apagar uma ideia.

- Adicionar candidato a uma ideia.
- Obter lista de candidatos a uma ideia.
- Obter resultados das candidaturas de um utilizador.
- Obter uma candidatura pelo lado do proponente.
- Obter uma candidatura pelo lado do candidato.
- Aceitar ou rejeitar uma candidatura (ação do proponente).
- Cancelar uma candidatura (ação do candidato).
- Obter lista de categorias.
- Adicionar *skill* a um utilizador.
- Obter lista de *skills* de um utilizador.
- Obter *skill* específica de um utilizador.
- Apagar uma *skill* de um utilizador.
- Adicionar *skill* a uma ideia.
- Obter lista de *skills* de uma ideia.
- Obter *skill* específica de uma ideia.
- Apagar uma *skill* de uma ideia.

No caso de existir um erro o corpo das respostas apresenta o formato *application/problem+json.* Desta forma é possível apresentar informação sobre o erro de forma estruturada, e contendo mais detalhe sobre o mesmo. Assim, os corpos das respostas *HTTP*, em caso de erro, têm por exemplo o seguinte formato:

```
{
    "type": "https://github.com/lourencovala/crossworking-
project/blob/main/docs/api/errors/UnauthorizedException.md",
    "title": "Authorization token Missing",
    "detail": "To access this resource you need to provide an valid
authorization token",
    "status": 401
}
```

Listagem 1 - Conteúdo corpo de resposta em situação de erro

5.2.3 Autorização

O serviço *Firebase* foi selecionado pela sua fácil adaptação à *Framework Android* e pelos seus serviços extra que permitem criação de contas de utilizador e início de sessão através de contas já existentes noutras plataformas como o *Facebook* e o *Gmail*.

A autenticação de um utilizador ocorre na aplicação de front-end, a qual gera para um utilizador autenticado, um identificador do mesmo. Para além do identificador do utilizador o serviço de autenticação é responsável por gerar também um id token. Um id token é um JSON Web Token que contem declarações sobre a identidade do utilizador. O id token é assinado quando gerado, podendo ser, a qualquer momento verificado, através de uma função disponibilizada pela biblioteca Firebase Admin para o efeito [10].

Quando o servidor *CrossWorking* recebe um pedido de criação de utilizador, guarda a instância de utilizador que recebe, a qual contem, para além da informação providenciada pelo próprio utilizador, um identificador gerado pelo servidor de autenticação. Uma vez guardada esta instância o utilizador passa a estar efetivamente registado na plataforma.

Todos os pedidos ao servidor devem conter no *Authorization Header* o token gerado pelo serviço *Firebase Authentication*. Recebido o pedido no servidor *CrossWorking* o mesmo é intercetado num *filter*, este irá obter o *header* de autenticação, extrair o *token* do mesmo e realizar a validação deste através do método *verifyIdToken,* disponibilizado pela biblioteca *Firebase Admin*. Caso seja possível validar o token, é retornado pelo método *verifyIdToken* o identificador do utilizador, correspondente a esse mesmo token, que será definido como atributo do pedido. Se a validação não for possível, é respondido de imediato com o código de erro *401 Unauthorized*.

Posteriormente, existem dois interceptors, que validam se o identificador, extraído dos atributos do pedido, tem autorização para aceder ao recurso que pretende. Se o utilizador não possuir essa autorização é então enviada uma resposta com o código de erro *403 Forbidden*.

6 Implementação do front-end

No presente capítulo é descrita a organização do *front-end* da plataforma *CrossWorking*, a sua estrutura, o padrão de desenho adotado e as bibliotecas presentes. É feita uma análise à implementação do serviço de autenticação do lado do cliente, bem como à implementação do acesso ao serviço de armazenamento de imagens.

A par da estrutura técnica da aplicação Android será também descrita a interação existente entre o utilizador e a aplicação, as funcionalidades disponíveis, os prossupostos para a correta utilização da aplicação e ainda os requisitos mínimos para poder executar a aplicação sem esperar comportamento anómalo.

6.1 Organização

Em 2017 a *Google,* durante o evento *Google I/O,* introduziu o padrão *MVVM* como a arquitetura a seguir durante o desenvolvimento de uma aplicação Android, a introdução deste padrão trouxe consigo a existência de classes *ViewModel*, e a possibilidade de manter instâncias de *ViewModel* mesmo durante reconfigurações; resolvendo assim um problema estrutural da *framework* Android, que ao destruir as atividades destruía também consigo os dados existente. Um ano após a recomendação da *Google*, surgiu, junto da comunidade de desenvolvedores *Android,* a aplicação de uma variante do *MVVM* denominada de *MVVM Clean*.

O padrão *MVVM Clean* prossupõe a existência dos seguintes elementos:

- *Fragment/UI* (User Interface): Elementos visuais com função de apresentar os dados ao utilizador final.

- *ViewModel*: Responsável por preservar os dados de forma consistente com o ciclo de vida da *UI.* Essencial para garantir a sobrevivência dos dados as mudanças de configuração que levam à destruição dos elementos visuais.
- *UseCases*: Os *UseCases* implementam a lógica do domínio, necessária ao funcionamento da aplicação.

Juntamente com o padrão *MVVM* foi também adotado o padrão repositório, que permite expor uma interface uniforme independentemente de realizar o acesso a múltiplos fornecedores de dados, contribuindo assim para o total desacoplamento das partes da aplicação.

Por forma a diminuir ao máximo a dependência entre as partes, foram também criados serviços, os quais apresentam a lógica de acesso aos dados. Cada um destes serviços deve implementar uma interface correspondente.

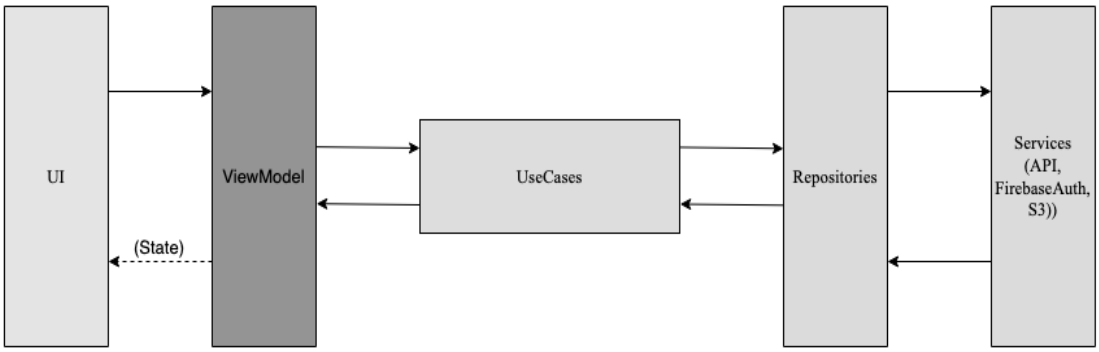

Figura 7 - Diagrama da arquitetura da aplicação android

O acesso à *API* do servidor *CrossWorking* é realizado através da biblioteca *Retrofit*; esta biblioteca permite realizar pedidos *HTTP*, mapear objetos de domínio em objetos *JSON* ao realizar pedidos, e mapear objetos *JSON* em objetos de domínio ao receber as respostas. Esta biblioteca prossupõe a existência de interfaces, onde são declaradas funções, anotadas com o pedido *HTTP* a realizar; estas interfaces são os serviços para o acesso à *API*.

Além dos serviços de acesso à *API*, foram também criadas as seguintes classes de serviços:

- *S3Service*: Este serviço implementa a lógica de acesso ao serviço de armazenamento de imagens.
- *AuthService*: Implementa toda a lógica do serviço de autenticação.

Cada uma destas classes implementa uma interface correspondente e encapsula a dependência do fornecedor de serviço. Assim na eventual necessidade de alteração do fornecedor de serviço, apenas é necessário implementar uma nova classe que cumpra a respetiva interface.

6.2 Jetpack Compose

A aplicação para a *framework* Android foi desenvolvida, na sua totalidade, com recurso ao toolkit *Jetpack Compose*. Por forma a seguir todas as indicações da documentação *Google Developers*, adotou-se um modelo no qual existe apenas uma *activity* durante todo o tempo de vida da aplicação.

Existe no *Android* um conjunto de diretrizes visuais e de navegação que pretendem criar um design semelhante em todas as aplicações, oferecendo uma experiência unificada e contínua ao utilizador. Estas diretrizes estão agrupadas num tema denominado de *Material Theme* [11]. Este tema sugere a existência de uma *top bar* e de uma *bottom bar* para questões de navegação, sugestões que foram aceites e implementadas na aplicação *CrossWorking*.

Para simplificar a aplicação deste tema o *toolkit Jetpack Compose* oferece uma *composable function* denominada de *Scaffold*, que providencia um layout indicado para aplicar o *Material Theme*. Assim na raiz de componentes da aplicação está a *composable function Material Theme*, seguida da função *Scaffold*. Uma vez definido o *Scaffold* foi necessário gerir os ecrãs a apresentar no interior do mesmo.

Nesta arquitetura *single activity,* cada ecrã é uma *composable function*, constituída por outras *composable functions*. Para gerir qual a função a ser apresentada num determinado momento, é utilizado o componente *Navigation*; este componente permite gerir qual a composable function a ser chamada durante a composição, mediante uma rota associada à mesma [12].

Ao utilizar a arquitetura single *activity* deixa de existir a possibilidade de utilizar os *lifecycle callbacks* para realizar ações. É também interdito chamar diretamente nas *compose functions* funções que alterem o estado, uma vez que poderão ocorrer múltiplas recomposições. Por forma a oferecer um substituto a estes *callbacks*, o *toolkit Jetpack Compose*, disponibiliza um conjunto de funções denominadas de *side-effects*. Os *side-effects* oferecem um local controlado onde podem ser realizadas chamadas ao viewmodel que modifiquem o estado do viewmodel.

No desenvolvimento da aplicação *CrossWorking* foram utilizados os seguintes *side-effects*:

- *LaunchedEffect*: Sempre que necessário obter o estado de um ecrã sem que seja necessário garantir a reposição do estado inicial é utilizado um *LaunchedEffect* [13].
- *DisposableEffect*: Sempre que necessário obter o estrado de um ecrã, e garantir que quando o mesmo saia da composição o estado é reposto é utilizado um *DisposableEffect* [14].

Cada *composable function* correspondente a um ecrã possui o seu *ViewModel*; os ecrãs definem os elementos visuais a apresentar mediante o estado do seu *ViewModel*. A opção de existir um *ViewModel* por ecrã permitiu garantir o isolamento entre ecrãs. Este isolamento, numa arquitetura *single activity*, assume uma maior relevância pois não existe garantia de a função em composição não volte a ser chamada antes do componente de navegação alterar a função em composição para a que se pretende navegar, podendo criar um ciclo de chamadas indesejadas [15].

6.3 ViewModels

Os *ViewModel*s são classes desenhadas para preservar o estado a ser apresentado pela *UI*, considerando o *lifecycle*, para que esse mesmo estado sobreviva a mudanças de configuração.

Na aplicação CrossWorking os *ViewModel*s expõem funções para realizar um determinado comportamento, e propriedades do tipo *State*, onde é mantido o estado atual do *ViewModel*. Os estados possíveis de determinado *ViewModel* são definidos através de uma *sealed class*, e dos seus filhos. Os filhos de uma *sealed class* podem ser do tipo *Object*, se for um estado simples, ou do tipo *data class*, se para além do estado existir outra informação associada a esse mesmo estado.

Para preservação do estado os *ViewModel*s possuem uma propriedade privada do tipo *MutableState*, que permite modificar o estado atual do *ViewModel*, e uma propriedade pública do tipo *State* que permite consultar o estado atual do *ViewModel*.

Os *MutableStates* assim como os *States* são integrados no ambiente de execução do *Compose*, fazendo com que qualquer alteração, que ocorra no valor de um estado que esteja a ser avaliado numa função que esteja em composição, levará a uma recomposição dessa mesma função.

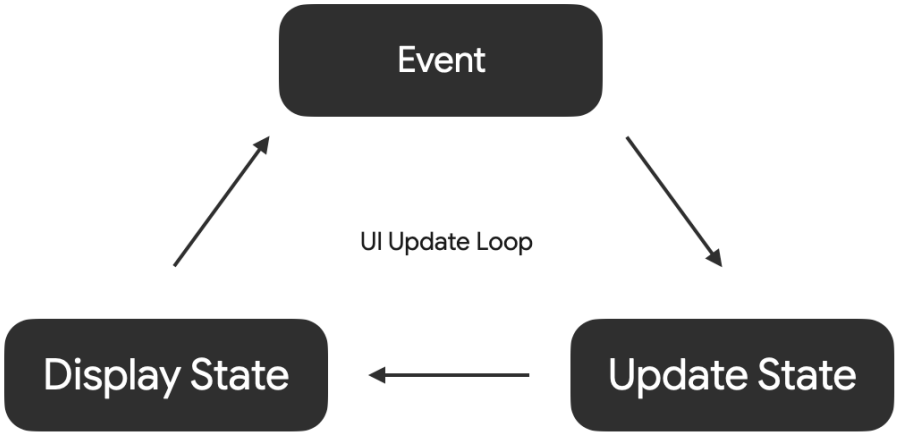

Figura 8 - Diagrama ciclo de atualização da UI

Para além das funções afetas a cada *ViewModel*, e ao comportamento que o mesmo implementa, os *ViewModel*s possuem ainda duas outras funções essenciais para a gestão do estado em arquitetura single *activity*, que são:

- *restoreInitialState*: Repõe o estado do *ViewModel* no seu estado inicial quando existe a destruição do efeito que originou a primeira transição de estado.
- *makeComplete*: Coloca o *ViewModel* no estado de completo, e aguarda que a navegação altere a função em composição para ser chamado o *dispose* do efeito que levou à primeira transição de estado.

As funções do *ViewModel* que envolvam chamadas assíncronas são executadas no contexto de uma corrotina com o *scope ViewModel*. O *viewmodelscope* cancela automaticamente a corrotina quando o *viewmodel* onde esta se insere é destruído, o que é ideal para este tipo de

utilização uma vez que apenas se pretende obter dados para apresentação enquanto o ecrã e o seu respetivo *viewmodel* existirem.

6.4 Gestão de dependências

A injeção de dependências em desenvolvimento android é muito utilizada e recomendada pela documentação da própria plataforma, uma vez que permite uma maior reutilização do código, aumentar a facilidade de alteração de um determinado elemento, e facilitar a testagem.

No entanto, à medida que o tamanho da aplicação cresce, o número de dependências a injetar também tem tendência a aumentar. Porém, a importância do uso de dependências não se está na criação das classes a serem injetadas. Este processo de criação, na verdade, torna difícil gerir todas as dependências manualmente.

Para continuar a tirar partido da injeção de dependências, sem necessidade de gerir manualmente as mesmas, tomou-se por opção a utilização de uma biblioteca de gestão de dependências. Na mesma secção onde se recomendava a injeção de dependências, também era apresentada uma sugestão da biblioteca a utilizar, sendo esta a biblioteca Hilt.

A biblioteca *Hilt* é baseada na biblioteca *Dagger*, e pretende trazer, face ao *Dagger* uma estrutura simplificada direcionada a aplicações *Android* [16].

Para declarar uma dependência como injetável foi criada uma classe denominada de *AppModule*, a qual foi anotada com *@Module* e *@InstallIn*. Estas anotações permitem definir essa classe como um módulo onde o *Hilt* deve pesquisar as dependências injetáveis, e onde instalar esse mesmo módulo. Nesta classe, para cada dependência, é criada uma função anotada com a anotação *@Bind*; o tipo de retorno dessa função indica o tipo que deve ser construído pela biblioteca Hilt e injetado por esta sempre que solicitado uma instância desse tipo num construtor ou campo anotado com *@Inject*.

Assim declararam-se na classe *AppModule* como injetáveis as seguintes classes:

- serviços;
- repositórios;
- *useCases;*
- *retrofit;*
- e o *viewmodel HelperViewModel* (posteriormente será abordada com mais detalhe esta opção).

Uma vez declaradas as dependências que se pretendiam injetar, foi ainda necessário criar uma classe denominada de *CrossWorkingApplication*, classe esta que foi anotada com a anotação *@HiltAndroidApp*, por forma a desencadear a geração do código e a fornecer à biblioteca *Hilt* uma referência para a instância de *Application* e instância de *Resources*.

Uma das principais vantagens da utilização da biblioteca *Hilt* é a possibilidade de injetar classes pertencentes a bibliotecas do *jetpack*, tais como *ViewModel* e *WorkManager*, sem necessidade de as declarar explicitamente, e delegar a gestão do lifecycle dessas classes à biblioteca *Hilt*.

Para tirar partido da integração entre a biblioteca *Hilt* e a biblioteca *ViewModel* foi necessário anotar os ViewModels que se pretendia injetar com a anotação *@HiltViewModel*; uma

vez anotado com esta anotação o ViewModel fica imediatamente pronto a ser injetado através da chamada à função *hiltViewModel()*.

O *HelperViewModel* foi declarado como injetável na classe *AppModule*, ao contrário dos restantes, uma vez que, a função *hiltViewModel* apenas pode ser chamada no contexto de uma *composable function*, e se pretendia obter o *ViewModel* antes de ocorrer a primeira composição. Assim o *HelperViewModel* é injetado diretamente num campo através da anotação *@Inject*.

6.5 Autenticação

Para realizar a autenticação na aplicação Android foi utilizado o serviço Firebase Authentication, o que permitiu criar os ecrãs de criação de conta e início de sessão integrados no tema da restante a aplicação.

As chamadas à biblioteca *FirebaseAuth* foram encapsuladas na classe *AuthService*, que cumpre o contrato *IAuthService*, e apenas nesta classe é possível aceder aos métodos da biblioteca de autenticação. Esta opção de desenho permite o desacoplamento da aplicação do serviço de autenticação e assim garantir que, na eventualidade de ser necessário alterar o serviço de autenticação apenas é necessário implementar uma nova classe que implemente a interface *IAuthService*, utilizando o novo serviço de autenticação.

Existem dois métodos de autenticação disponíveis para os utilizadores, autenticação através de email e palavra-passe, e autenticação via conta *Google*.

6.5.1 Email e Password

A autenticação através de email e palavra-passe possui dois ecrãs distintos, para criação de conta e início de sessão, assim como lógicas distintas para estes dois ecrãs.

No ecrã de criação de conta o utilizador introduz o seu email a sua palavra-passe, e aciona o botão Registar. O acionar do botão Registar provoca uma chamada assíncrona ao método *createUserWithEmailAndPassword* exposto pela biblioteca *FirebaseAuth*.

No ecrã de início de sessão o utilizador introduz também o seu email e a sua palavra-passe, e aciona o botão Iniciar Sessão. O acionar do botão Iniciar Sessão provoca uma chamada assíncrona ao método *signInWithEmailAndPassword* exposto também pela biblioteca *FirebaseAuth*.

Tanto no caso de criação de conta como no caso de início de sessão, obtém-se o objeto *FirebaseUser* que contém o *userId* do utilizador. Criada a conta no serviço de autenticação é necessário realizar um pedido *POST* ao servidor *CrossWorking* para que a conta seja também criada no contexto da *API*; em resposta a este pedido é recebido um objeto *User*, que é guardado num *State* global a toda a aplicação permitindo apresentar a imagem do utilizador na *bottom bar*, assim como identificar, em qualquer momento, o utilizador da aplicação com sessão iniciada.

6.5.2 Conta Google

A autenticação da conta *Google* existe no ecrã de criação de conta assim como no ecrã de início de sessão, no entanto, e uma vez que a função exposta pela biblioteca *FirebaseAuth* para acesso com conta *Google* não distingue criação de conta e início de sessão, é utilizada a mesma função para as duas ações.

Para obtenção das credenciais *Google* é lançada uma *activity* através da função *rememberLauncherForActivityResult*. Esta função permite chamar uma *activity* e obter como parâmetro de uma função callback o resultado dessa chamada. O resultado obtido é um objeto *Intent* que possui as credenciais google necessárias para iniciar sessão no serviço *FirebaseAuth*.

Assim na posse das credenciais da *Google* é realizada uma chamada assíncrona ao método *enterWithGoogle* exposto pela biblioteca *FirebaseAuth*.

Este método de autenticação retorna um objeto *FirebaseUser*, o qual possui o *userId* do utilizador autenticado. É então realizado um pedido *HTTP* ao servidor *CrossWorking* para criação de conta do utilizador recém autenticado. Se o utilizador ainda não possuir conta na *API*, a mesma é criada, no entanto, caso já exista um utilizador com esse email o servidor responde com o erro *409 Conflict*, e como efeito dessa resposta é feito um pedido pela informação desse utilizador. Em ambos os casos, existirá no fim do processo de autenticação, uma instância de *User* que será guardada num *State* global a toda a aplicação permitindo apresentar a imagem do utilizador na *bottom bar*, assim como identificar, em qualquer momento, o utilizador da aplicação com sessão iniciada.

6.6 Armazenamento de Imagens

O serviço S3 é um serviço de armazenamento de objetos, este serviço oferece a possibilidade de replicação dos dados por múltiplas regiões, e possibilita o armazenamento de ficheiros de grandes dimensões.

No contexto da plataforma CrossWorking este serviço é utilizado para o armazenamento das imagens de perfil.

Para garantir o isolamento total desta framework bem como do serviço de armazenamento de imagens, do restante código da aplicação foi criada a classe *S3Service* que implementa a interface *IImageStorage.* Assim como no serviço de autenticação, a substituição deste serviço por qualquer outro apenas implica a implementação de uma nova classe que cumpra a interface *IImageStorage*.

A classe *S3Service* apenas implementa um método que permite realizar o upload de uma imagem, e devolve a localização do recurso, uma vez criado.

Para poder utilizar a framework *Amplify* é necessário iniciar a mesma, processo que ocorre da primeira vez que o *UserInfoViewModel* é construído. Esta opção garante que o serviço apenas é iniciado quando é estritamente necessário, não ocupando memória até esse momento.

Quando um utilizador aciona a opção para selecionar uma imagem a ser utilizada como imagem de perfil, é lançada uma *activity* através da função *rememberLauncherForActivityResult*, a qual permite obter o resultado da *activity* lançada, neste caso, o uri local da imagem a ser carregada.

Uma vez iniciado o serviço, a classe *Amplify* expõe um *companion object* denominado de *Storage*, o qual expõem a função *uploadInputStream*. Foi instanciado um *input stream* para a imagem identificada pelo *URI* local, e passado esse mesmo *input stream* à função *uploadInputStream*, juntamente com um *id* gerado através da função *randomUUID*.

Após o carregamento da imagem é montado o *URI* remoto da imagem, e retornado o mesmo. Este *URI* é armazenado na *API CrossWorking* associado ao perfil do utilizador que realizou o carregamento da imagem.

7 Conclusões

A plataforma *CrossWorking* apresenta neste momento todas as funcionalidades necessárias para o seu funcionamento e para cumprir os objetivos delineados. Permite a troca de ideias entre utilizadores, a realização de candidaturas às ideias, e troca de contactos entre o proponente de uma ideia e o utilizador que se propôs a concretizá-la.

O servidor back-end, assim como a aplicação front-end, encontram-se num estado estável, pelo que o servidor já se encontra hospedado e disponível através do *URI* https://crossworking.link, e a aplicação Android está pronta a ser submetida para avaliação da *Google* e distribuída através da sua loja de aplicações, *Google Play Store*.

Apesar do estado atual dos componentes da plataforma, foram identificados diversos pontos de melhoria na implementação atual, assim como funcionalidades que seriam muito interessantes adicionar para que a aplicação se tornasse mais intuitiva.

Destacaram-se assim como pontos de melhoria os seguintes:

- Nomes de funções, classes e propriedades implementadas tanto no servidor back-end como também na aplicação *front-end*.
- Implementação de testes de integração e testes de *UI* para aferir o correto comportamento dos componentes.
- Gestão das funções de navegação recebidas como parâmetro pelas *composable functions*.
- Organização do código em packages.
- Testes existentes (unitários na aplicação Android e de integração no servidor).
- Aumentar a cobertura dos mesmos.
- Melhorias em certos aspetos visuais na aplicação Android, como animações, *hold-to-select* elementos de listas e esconder a *bottom bar* enquanto o teclado estiver ativo.
- Processo de autenticação via *Google* necessitava de mais atenção.

Como funcionalidades extra que se consideram importantes de implementar podem ser apontadas as seguintes:

- Recuperação de palavra-passe para utilizadores já autenticados na plataforma.
- Serviço de notificações para aviso de aceitação de candidatura.
- Campo específico para adição de imagens a uma ideia juntamente com as suas descrições.
- Sistema de recomendação de *skills* de um utilizador, permitindo que a comunidade valorize determinada *skill* de um utilizador.
- Filtros no ecrã de *feed* para a apresentação de ideias.
- Associação de perfis de outras redes sociais para possibilitar partilha de mais informação acerca do utilizador.

Embora todos estes pontos anteriormente mencionados tenham a sua devida importância, os autores consideram que o estado atual é o necessário para conclusão deste projeto no contexto da unidade curricular de Projeto e Seminário.

Referências

[1] Yuval Harari. (2015). Sapiens: A Brief History of Humankind. 2ª Edição, RANDOM HOUSE UK. Hebrew

[2] BusinessofApps. (). "Android Stats", https://www.businessofapps.com/data/android-statistics. (Consulta em: 04/06/ 2022).

[3] GlobalStats. (). "Mobile Operating System Market Share Worldwide" https://gs.statcounter.com/os-market-share/mobile/worldwide. (Consulta em: 04/06/ 2022)

[4] Spring. (). "Spring Framework Documentation", https://docs.spring.io/spring-framework/docs/current/reference/html/. (Consulta em: 22/06/ 2022)

[5] Developers Android. (). "Build better apps faster with Jetpack Compose", https://developer.android.com/jetpack/compose. (Consulta em: 11/07/ 2022).

[6] Developers Android. (). "Guide to app architecture", https://developer.android.com/topic/architecture. (Consulta em: 11/07/ 2022).

[7] Documentação API CrossWorking. (). "OpenAPI", https://github.com/lourencovala/crossworking-project/blob/main/docs/api/OpenAPI.yaml. (Consulta em: 14/07/2022)

[8] Siren. (). "Siren: a hypermedia specification for representing entities", https://github.com/kevinswiber/siren. (Consulta em: 14/07/2022)

[9] JDBI Docs. (). "Advanced Topics", https://jdbi.org/#_advanced_topics_2. (Consulta em: 14/07/2022)

[10] Firebase Docs. (). "Verify ID Tokens", https://firebase.google.com/docs/auth/admin/verify-id-tokens. (Consulta em: 14/07/2022)

[11] Developers Android. (). "Material Design for Android", https://developer.android.com/guide/topics/ui/look-and-feel. (Consulta em: 19/07/2022)

[12] Developers Android. (). "Navigation with Compose", https://developer.android.com/jetpack/compose/navigation. (Consulta em: 19/07/2022)

[13] Developers Android. (). "LaunchedEffect", https://developer.android.com/jetpack/compose/side-effects#launchedeffect. (Consulta em: 19/07/2022)

[14] Developers Android. (). "DisposableEffect", https://developer.android.com/jetpack/compose/side-effects#disposableeffect. (Consulta em: 19/07/2022)

[15] droidcon. (). "Jetpack compose navigation architecture with viewmodels", https://www.droidcon.com/2021/10/25/jetpack-compose-navigation-architecture-with-viewmodels/. (Consulta em: 19/07/2022)

[16] Developers Android. (). "Dependency injection with Hilt", https://developer.android.com/training/dependency-injection/hilt-android#hilt-and-dagger. (Consulta em: 19/07/2022)