IPG Politécnico da Guarda
Escola Superior de Tecnologia e Gestão

RELATÓRIO DE ESTÁGIO

Curso Técnico Superior Profissional
em Desenvolvimento de Aplicações Informáticas

João Pereira de Almeida Ferreira

dezembro | 2012

Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

RELATÓRIO DE ESTÁGIO

JOÃO PEREIRA DE ALMEIDA FERREIRA

RELATÓRIO PARA A OBTENÇÃO DO DIPLOMA DE TÉCNICO SUPERIOR PROFISSIONAL

EM DESENVOLVIMENTO DE APLICAÇÕES INFORMÁTICAS

dezembro de 2021

Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

RELATÓRIO DE ESTÁGIO

Aluno: João Ferreira

Supervisor: José Gouveia

Orientador: Paulo Nunes

dezembro de 2021

Ficha de identificação

Estudante

Nome: João Pereira de Almeida Ferreira

Nº de Aluno: 1702944

Curso: TeSP de Desenvolvimento de Aplicações Informáticas

Telemóvel: 969 206 756

E-mail: jpaferreira93@gmail.com

Local de estágio

Empresa: Capgemini Engineering Portugal

Localidade: Fundão

Morada: Centro de Negócios e Serviços, Praça Amália Rodrigues, 6230-350

Site: www.capgemini-engineering.com

Estágio curricular

Duração: 750 horas

Data de início: 19/03/2021

Data de fim: 03/08/2021

Orientador: Paulo Jorge Costa Nunes

Supervisor na empresa: José Gouveia

Agradecimentos

Quero começar por agradecer à minha família por todo o apoio e motivação que me têm dado nestes tempos difíceis.

Agradeço também ao IPG por me ter dado uma oportunidade de mudar de carreira com este curso, ao meu orientador pela sua disponibilidade para tirar dúvidas e aos meus colegas de curso que me ajudaram a manter a boa disposição durante este período.

De seguida, queria agradecer à Capgemini Engineering e em particular, ao José Gouveia e ao Luís Alves, por me terem ajudado na integração à empresa e guiado durante toda a execução do projeto.

Por último, tenho de agradecer à equipa onde fui inserido, pelos laços de amizade que rapidamente criámos, pelo seu humor e excelente espírito de equipa.

Plano de Estágio

O plano de estágio consiste em:

- Integrar o estagiário na empresa e em projeto interno
- Aprender as boas práticas de programação em Android praticadas na empresa
- Aprender a metodologia Scrum e as respetivas cerimónias e participar nas mesmas
- Utilizar a ferramenta de controlo de versão de software Git
- Utilizar o sistema operativo Android Automotive
- Investigar o conceito de Hyper Agile Backend
- Aprender a linguagem de programação Kotlin
- Desenvolver software em Java e Kotlin
- Investigar algoritmos e modelos de reconhecimento facial que utilizam TensorFlow

Resumo

O estágio que este relatório descreve, teve uma duração de 750 horas, tendo iniciado no dia 19 de março de 2021 e acabado no dia 3 de agosto do mesmo ano. Foi realizado na empresa *Capgemini Engineering Portugal* (anteriormente conhecida por *Altran*) no Fundão.

O projeto no qual fui integrado foi uma prova de conceito com o nome *A2C2F - Android Automotive Car Connectivity Framework*, tendo o objetivo de criar um sistema de reconhecimento facial no sistema operativo *Android Automotive*, para ser utilizada em automóveis. Após as primeiras user stories terem sido definidas, a equipa foi divida em *Frontend* e *Backend*, tendo eu ficado no grupo responsável pela parte *Frontend* do projeto. Como tal, trabalhei em *Android* durante quase todo o período de estágio. Infelizmente, acabei por não trabalhar em *Android Automotive* por motivos que irei expor noutro capítulo.

Fui, juntamente com o resto da equipa, responsável por criar grande parte das user stories relativas à parte de *Frontend*, tendo-as apresentado ao *Project Manager* para serem revistas e aceites.

Participei também em todas as cerimónias de *Scrum* que foram feitas ao longo do projeto.

Índice

Índice de figuras

1. Projeto

De tempos a tempos, a *Capgemini Engineering* gere uma iniciativa onde os colaboradores podem dar as suas ideias para novos projetos. A ideia deste projeto foi uma das premiadas pela empresa, pelo que pôde avançar e tornar-se num projeto real com um pequeno orçamento. É de sublinhar que, sendo um projeto criado dentro da empresa, é apenas uma prova de conceito, pelo que não há quaisquer clientes envolvidos no seu desenvolvimento.

O projeto em si consiste na criação de um sistema de reconhecimento facial no sistema operativo *Android Automotive* da *Google*. O objetivo principal consiste em examinar a face do condutor e compará-la a faces já guardadas no sistema, para saber se o condutor tem permissão para conduzir o veículo. Um segundo objetivo, opcional, seria o de avaliar a fadiga ou embriaguez do condutor para determinar se o mesmo se encontra capaz de conduzir. Este segundo objetivo não foi possível cumprir, pelo menos até à data final do estágio.

A aplicação funciona da seguinte forma:
Quando é iniciada pela primeira vez, o utilizador só tem a opção de criar um novo perfil, onde introduz os seus dados e regista a sua face. Tendo então o perfil criado, esse utilizador pode fazer login na aplicação de duas maneiras diferentes: através do seu username e password, ou através de reconhecimento facial. Se uma pessoa diferente que já tenha um perfil criado se puser no lugar do condutor, a aplicação irá reconhecê-la e fazer login do seu perfil. Caso esse utilizador ainda não tenha um perfil criado, a aplicação irá perguntar-lhe se quer criar um novo perfil. Na criação desse perfil irá pedir a password do *Primary User* (utilizador primário) para poder prosseguir com o processo.

1.1. Fase inicial de pesquisa

Durante as primeiras semanas, foi me pedido apenas para pesquisar sobre microserviços, contentores e orquestração. Conceitos que apresentarei noutro capítulo deste relatório.
No final da terceira semana de estágio tive de reunir e traduzir toda esta informação num documento *State of the Art*, que entreguei ao *Project Manager*.
Nesta primeira fase, também instalei o *Android Studio* e comecei a fazer pequenas aplicações para me ambientar à linguagem. Como estudei Java durante o curso, foi essa a linguagem com que

trabalhei durante este tempo.
Também li alguma documentação da empresa. Em particular, estudei como a metodologia *Scrum* é utilizada na empresa e quais as boas práticas de programação em Android.

Foi durante esta fase, que a minha equipa se apercebeu de um problema. A ferramenta com que iríamos trabalhar, um emulador do *Android Automotive*, não tinha ligação com câmara, pelo que não se podia testar a aplicação nesse emulador. Por este motivo, o projeto acabou por ser feito em Android para dispositivos móveis, com a intenção de mudarmos o projeto para Android Automotive assim que o Product Owner arranjasse uma solução. Até à data final do estágio nenhuma solução foi encontrada, pelo que o projeto foi sempre trabalhado simplesmente em Android.

1.2. Modelo de reconhecimento facial

Após terminar a fase de pesquisa inicial comecei a procurar aplicações já existentes nas quais nos pudéssemos basear, visto que não tínhamos qualquer experiência em criar modelos de *Machine Learning*. Depois de várias tentativas sem sucesso, a minha equipa deparou-se com um projeto em Java no *GitHub* que tinha um modelo de reconhecimento facial já construído e treinado.
Durante esta fase comecei também a aprender autonomamente as bases da linguagem Kotlin, que viria a ter de utilizar mais tarde.

1.3. User stories

Tendo como base esta aplicação e como já tínhamos o modelo de reconhecimento facial, começámos então a fazer as primeiras user stories.
As primeiras user stories que fiz foram a construção de um interface para o registo de utilizadores e a implementação de um sistema de segurança através de password para fazer esse registo.

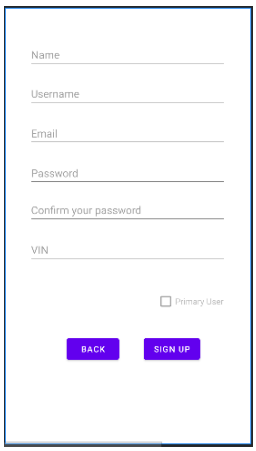

Figura 1: Esboço do ecrã de registo de utilizadores

Para efeitos de teste, começámos por usar temporariamente o json-server (apresentado em: https://www.npmjs.com/package/json-server). O json-server é simplesmente uma falsa *API REST* que imita métodos *HTTP*, sendo os dados guardados num ficheiro *JSON* que servia de base de dados. Para conseguir aceder a esta base de dados local, foi necessário instalar o *node.js*. Para comunicar com o *json-server* basta escrever o comando: json-server --watch db.json. Sendo db.json o nome do ficheiro *JSON*.

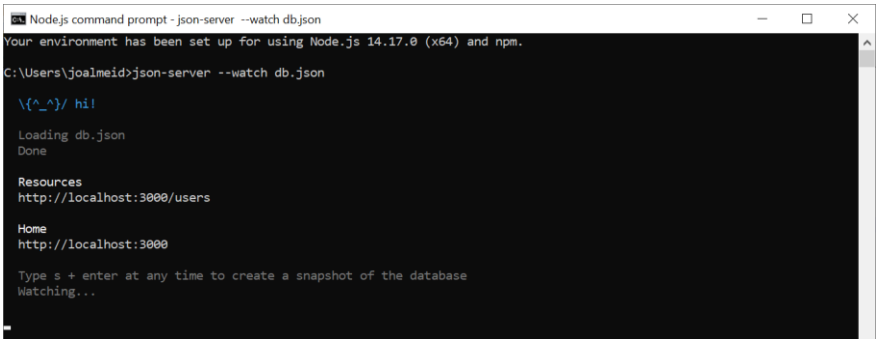

Figura 2: Linha de comandos Node.js

Assim que a equipa teve acesso a um servidor próprio, foi então feita a ligação entre o *Frontend* e o *Backend*. Para comunicar com o servidor, usámos *Retrofit*. O *Retrofit* é uma biblioteca desenvolvida pela *Square* que é utilizada como um *REST Client* no Android e Java. Utiliza a biblioteca *OkHttp* para fazer Http Requests. O *Retrofit* torna mais fácil recuperar e fazer upload de *JSON* através de uma web service *REST*.

A figura 3 mostra o código da função createUsers, que faz a comunicação com o servidor quando é registado um novo utilizador. A função onResponse é executada quando é recebida uma resposta

HTTP, mesmo que a resposta seja um erro 404. Caso contrário, é executada a função onFailure.

A função cria um objeto da classe User com os dados que são introduzidos pelo utilizador nas respetivas áreas (figura 1). É depois feita uma *call* para o endpoint que faz o registo de utilizadores (figura 4).

São feitas condições que comparam o corpo da resposta do servidor à resposta do *Backend* para ver se já o username ou e-mail introduzidos já estão registados na base de dados.

Caso estas condições se verifiquem, é feito um *Toast* (pequeno pop-up em Android) a avisar o utilizador de que esses dados já estão registados no sistema.

Se nenhuma dessas condições se verificar, são guardados alguns dados em SharedPreferences e é feito um Toast a avisar o utilizador de que o registo foi bem-sucedido. SharedPreferences é uma funcionalidade de Android que permite guardar dados para poderem ser usados em diferentes Activities.

Por fim, é feito o login do utilizador que acabou de se registar e é feito um Intent para mudar a Activity para continuar para o processo do registo facial.

```
//registers new user in the database
@Throws(NoSuchAlgorithmException::class)
fun createUsers(etName: String, etUsername: String, etEmail: String, etPassword: String,
    etVin: String, sp: SharedPreferences, loggedInState: SharedPreferences) {
    val uniqueString = toHexString(getSHA(etUsername))
    val user = User(etName, etUsername, etEmail, toHexString(getSHA(etPassword)), uniqueString, etVin)
    Log.d( tag: "user - sent to server", user.toString())
    val call = jsonPlaceHolderApi.createUser(user)
    call.enqueue(object : Callback<Any> {
        override fun onResponse(call: Call<Any>, response: Response<Any>) {
            Log.d( tag: "response", response.body().toString())
            if (response.body()
                    .toString() == "{message=This username is already registered, errorCode=ERROR_USERNAME_EXISTS}"
            ) {
                Log.d( tag: "Code isn't successful: ", response.code().toString())
                Log.d( tag: "response username", response.body().toString())

                Toast.makeText(
                    mainActivity,
                    "Username already registered",
                    Toast.LENGTH_LONG
                ).show()
                return
            }
            if (response.body()
                    .toString() == "{message=This email is already registered, errorCode=ERROR_EMAIL_EXISTS}") {
                Log.d( tag: "Code isn't successful: ", response.code().toString())
                Log.d( tag: "response email", response.body().toString())
                Toast.makeText(
                    mainActivity,
                    "Email already registered",
                    Toast.LENGTH_LONG
                ).show()
                return }
            //use of SharedPreferences to save name and unique string
            val editor = sp.edit()
            editor.putString("name", etName)
            editor.putString("uString", uniqueString)
            editor.commit()
            Toast.makeText(
                mainActivity,
                "The register of user" + " " +
                        etName + " " + "was successful! Welcome.",
                Toast.LENGTH_LONG
            ).show()
            try { AltLogin().loginUserAlt(etUsername, toHexString(getSHA(etPassword)), reg: true, sp, loggedInState)
            } catch (e: NoSuchAlgorithmException) { e.printStackTrace() }
            val i = Intent(mainActivity, FaceRecTut::class.java)
            i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
            mainActivity.startActivity(i)
        }
        override fun onFailure(callPost: Call<Any>, t: Throwable) {
            Log.d( tag: "Create User Failure", t.message!!)
        }
    })
}
```

Figura 3: Função de Registo de Utilizadores

```
interface JsonPlaceHolderApi {
    //login system with username and password
    @POST( value: "account/login")
    fun loginUserAlt(@HeaderMap headers: Map<String, String>): Call<String>

    //login system with uniqueString
    @POST( value: "account/loginByUniqString")
    fun loginUserUnique(@Header( value: "uniqString") uniqString: String): Call<String>

    //logout system with token
    @POST( value: "account/logout")
    fun logoutUser(@Header( value: "x-access-token") token: String): Call<String>

    //register users
    @POST( value: "users/register")
    fun createUser(@Body postResponse: Any): Call<Any>
```

Figura 4: Endpoints do Backend

De seguida, fui encarregado de várias user stories, de entre as quais: um login alternativo ao reconhecimento facial que funciona por username e password, a validação dos dados durante o registo de um novo utilizador e a criação de um ficheiro xml de todas as strings das mensagens que são apresentadas ao utilizador.

Para a validação dos dados de registo, utilizei *if's* encadeados (figura 5). Considerei que os campos de preenchimento não poderiam estar vazios e que o nome e o username não poderiam ter caracteres especiais e tinham de ter um número mínimo de caracteres. Também validei a password de forma semelhante, usando as funções representadas na figura 6.

```
//register form checks
if (etName.text.toString().isNotEmpty() && etUsername.text.toString().isNotEmpty() &&
    etEmail.text.toString().isNotEmpty() && etPassword.text.toString().isNotEmpty() &&
    etPassword2.text.toString().isNotEmpty()
) {
    if (!hasSpecialChars(etName.text.toString())) {
        if (!hasSpecialChars(etUsername.text.toString())) {
            if (has3OrMoreChars(etName.text.toString())) {
                if (has3OrMoreChars(etUsername.text.toString())) {
                    if (etEmail.text.toString().matches(emailPattern.toRegex())) {
                        if (isPassValid(etPassword.text.toString())) {
                            if (etPassword.text.toString() == etPassword2.text.toString()) {
                                try {
                                    retro.createUsers(
                                        etName.text.toString(),
                                        etUsername.text.toString(),
                                        etEmail.text.toString(),
                                        etPassword.text.toString(),
                                        etVin.text.toString(),
                                        getSharedPreferences( name: "userData", MODE_PRIVATE),
                                        getSharedPreferences( name: "loggedInState",MODE_PRIVATE)
                                    )
                                } catch (e: NoSuchAlgorithmException) {
                                    e.printStackTrace()
                                }
```

Figura 5: Validações no formulário de Registo de Utilizadores

```kotlin
//checks if password is valid
private fun isPassValid(pass: String): Boolean {
    return hasLength(pass) && hasDigit(pass) && hasUppercase(pass)
}
//verifies length of password
private fun hasLength(value: String): Boolean {
    return value.length >= 7
}
//verifies if password contains digits(numbers)
private fun hasDigit(value: String): Boolean {
    return value.matches("(.*\\d.*)".toRegex())
}
//verifies if password contains uppercase
private fun hasUppercase(value: String): Boolean {
    return value != value.lowercase(Locale.getDefault())
}
//verifies if name and username contain special characters
private fun hasSpecialChars(value: String): Boolean {
    val special: Pattern = Pattern.compile( regex: "[^a-z0-9 ]", Pattern.CASE_INSENSITIVE)
    val matcher: Matcher = special.matcher(value)
    return matcher.find()
}
//verifies if name and username have at least 3 characters
private fun has3OrMoreChars(value: String): Boolean {
    return value.length >= 3
}
```

Figura 6: Funções de validação

Os ficheiros xml das strings são usados para colocar todas as strings de texto que são apresentadas ao utilizador, tal como em botões ou caixas de texto, num só ficheiro. Para dar um exemplo, existem vários botões com o texto “Cancelar” na aplicação, para isso podemos vamos buscar a string específica a este ficheiro e associá-la a todos esses botões. A aplicação tem a funcionalidade de mudar o idioma entre Ingês e Português. Como tal, existe um ficheiro string.xml para cada língua.

```xml
<string name="register_user">Register User</string>
<string name="ok">OK</string>
<string name="cancel">Cancel</string>
<string name="register_title">Please enter the primary user\'s password</string>
<string name="continue_bt">Continue</string>
<string name="wrong_pass">Incorrect password</string>
<string name="del_all">Do you want to delete all Recognitions?</string>
<string name="del_all_bt">Delete All</string>
<string name="toast_rec_cleared">Recognitions Cleared</string>
<string name="toast_rec_up">Recognitions Updated</string>
<string name="no_face_list">No Faces Added</string>
<string name="sel_recog_delete">Select Recognition to delete:</string>
```

Figura 7: Excerto do ficheiro xml de strings em Inglês

```xml
<string name="register_user">Registar Utilizador</string>
<string name="ok">OK</string>
<string name="cancel">Cancelar</string>
<string name="register_title">Por favor introduza a palavra-passe do utilizador principal</string>
<string name="continue_bt">Continuar</string>
<string name="wrong_pass">Palavra-passe incorreta</string>
<string name="del_all">Pretende apagar todos os reconhecimentos?</string>
<string name="del_all_bt">Apagar tudo</string>
<string name="toast_rec_cleared">Reconhecimentos limpos</string>
<string name="toast_rec_up">Reconhecimentos atualizados</string>
<string name="no_face_list">Nenhum reconhecimento adicionado</string>
<string name="sel_recog_delete">Selecione reconhecimentos para apagar:</string>
```

Figura 8: Excerto do ficheiro xml de strings em Português

Após insistência do *Product Owner*, mudou-se a linguagem do projeto de Java para Kotlin. Para não termos de reescrever todo o código do zero, usámos a funcionalidade do *Android Studio* para converter ficheiros Java em Kotlin. Depois, foi apenas uma questão de limpar o código para evitar os erros provocados pela conversão.

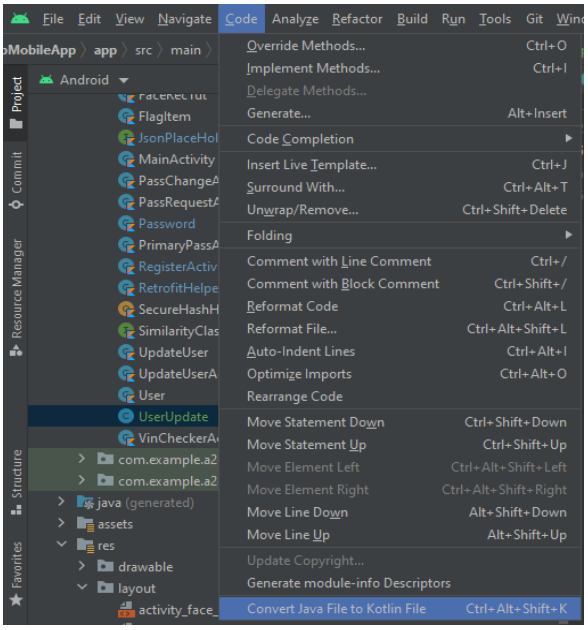

Figura 9: Conversão de ficheiros Java para Kotlin

1.4. Testes unitários

Perto do fim do estágio, foi-nos pedido para começarmos a fazer testes unitários. No entanto, nenhum de nós tinha experiência com isso, e como não era uma tarefa prioritária, acabámos por não conseguir completar esta tarefa. De seguida, estão duas figuras que mostram testes unitários feitos a duas Activities diferentes.

A figura 10 mostra testes feitos a uma função que passa uma String por um algoritmo hash para a encriptar. A função toHexString_and_getSHA_True_Test introduz um exemplo de uma String que é transformada e compara-a com o resultado expectável. A função toHexString_and _getSHA_False_Test compara a String com um resultado falso.

A figura 11 contém testes a funções de validação de dados.

```
package com.example.a2c2f_facerecognition

import ...

class PrimaryPassActivityTests {
    var  sha: SecureHashHelper = SecureHashHelper
    @Test
    @Throws(NoSuchAlgorithmException::class)
    fun toHexString_and_getSHA_True_Test() {
        val actual = sha.toHexString(sha.getSHA("A123456"))
        val expected =
            "d13079915c02da7c244eb43452062993a818c1fd782c1d7df04
        Assert.assertEquals(expected, actual)
    }

    @Test
    @Throws(NoSuchAlgorithmException::class)
    fun toHexString_and_getSHA_False_Test() {
        val actual = sha.toHexString(sha.getSHA("A123456"))
        val expected =
            "13079915c02da7c244eb43452062993a818c1fd782c1d7df04d
        Assert.assertNotEquals(expected, actual)
    }
}
```

Figura 10: Testes unitários

```
class PassChangeActivityTest {
    var passChangeActivity = PassChangeActivity()

    @Test
    fun isPassValid_True_Test() { Assert.assertTrue(passChangeActivity.isPassValid("A123456")) }

    @Test
    fun isPassValid_False_Test() { Assert.assertFalse(passChangeActivity.isPassValid("A12345")) }

    @Test
    fun hasLength_isMoreOrEqualsSeven_Test() { Assert.assertTrue(passChangeActivity.hasLength("A123456")) }

    @Test
    fun hasLength_isLessOrEqualsSeven_Test() { Assert.assertFalse(passChangeActivity.hasLength("A12345")) }

    @Test
    fun hasDigit_True_Test() { Assert.assertTrue(passChangeActivity.hasDigit("A123456")) }

    @Test
    fun hasDigit_False_Test() { Assert.assertFalse(passChangeActivity.hasDigit("ABCDEFG")) }

    @Test
    fun hasUppercase_True_Test() { Assert.assertTrue(passChangeActivity.hasUppercase("A123456")) }

    @Test
    fun hasUppercase_False_Test() { Assert.assertFalse(passChangeActivity.hasUppercase("a123456")) }
}
```

Figura 11: Teste unitários (continuação)

2. Conceitos de Hyper Agile Backend

2.1. Microserviços

Para perceber o que são microserviços e como estes são cada vez mais importantes, tenho primeiro de apresentar o conceito de arquitetura monolítica. Arquitetura monolítica é considerada uma maneira tradicional de desenvolver aplicações. Uma aplicação monolítica é construída como sendo única e indivisível. É um modelo unificado, onde todas as funções da aplicação se encontram num só bloco. Os diversos módulos do sistema são executados numa mesma máquina, compartilhando recursos de processamento, memória, bases de dados e ficheiros.

Este tipo de arquitetura torna o desenvolvimento e a implementação de aplicações mais fácil, visto que os programadores só têm de trabalhar com um único sistema. No entanto, se o sistema for crescendo e ficando mais complexo, consome cada vez mais recursos e torna-se mais difícil de trabalhar, o que acaba por gerar alguns desafios substanciais para a sua manutenção.

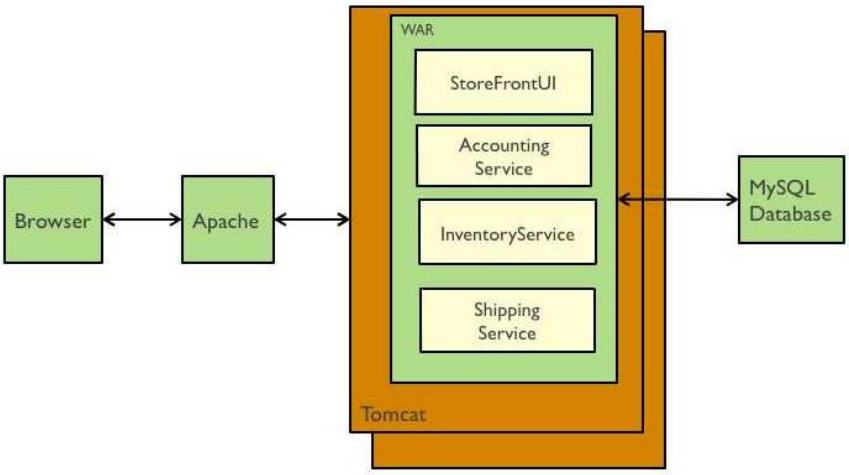

Figura 12: Representação de uma arquitetura monolítica

Por outro lado, os microserviços são uma abordagem de arquitetura para a criação de aplicações. O que diferencia a arquitetura de microserviços das abordagens monolíticas tradicionais é como esta decompõe a aplicação por funções básicas. Cada função é denominada um serviço e pode ser criada e implementada de maneira independente. De forma simples, os microserviços ajudam a construir uma aplicação como um conjunto de pequenos serviços, em que cada um executa o seu próprio processo e pode ser implementado de forma independente.

Estes serviços podem ser escritos em diferentes linguagens de programação e usar diferentes técnicas de armazenamento de dados. A manutenção deste tipo de sistema é muito mais eficaz,

visto que não é necessário implementar todo o sistema cada vez que se fazem modificações num só serviço. Também é mais fácil de se testar, no entanto, a complexidade de criar um sistema com este tipo de arquitetura torna os microserviços uma solução ineficaz para aplicações mais simples.

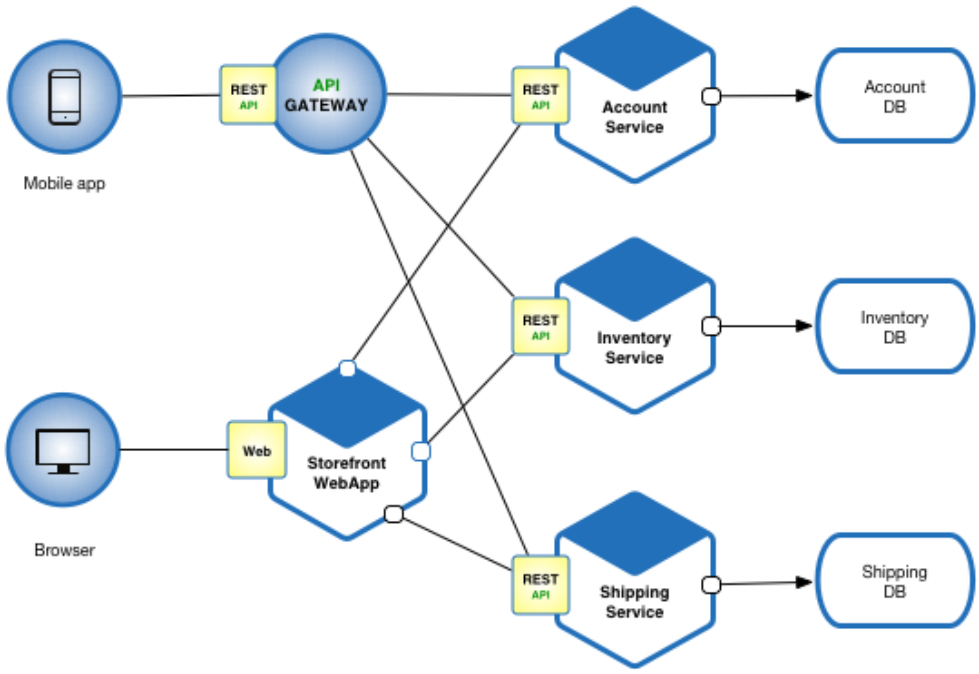

Figura 13: Representação de uma arquitetura de microserviços

2.2. Contentores

No ramo da Informática, um contentor não é mais do que um pacote standard de software que agrupa o código de uma aplicação juntamente com os ficheiros de configuração, as bibliotecas e as dependências associadas necessárias à execução da aplicação. Isto permite aos programadores implementar aplicações facilmente em todos os ambientes.

O problema de uma aplicação não ser executada corretamente quando é movida de um ambiente para outro é tão antigo como o próprio desenvolvimento de software. Estes problemas surgem normalmente devido a diferenças nos requisitos de biblioteca subjacentes da configuração e outras dependências.

Os contentores abordam este problema ao fornecer uma infraestrutura leve e imutável para o empacotamento e implementação de aplicações. Uma aplicação ou serviço, as respetivas dependências e a sua configuração são empacotadas como uma imagem de contentor. A aplicação contentorizada pode ser testada como unidade e implementada como uma instância de imagem de contentor para o sistema operativo anfitrião.

Desta forma, os contentores permitem aos programadores implementar aplicações em ambientes com pouca ou nenhuma modificação.

2.3. Orquestração

A orquestração de contentores é a automação de grande parte do esforço operacional necessário para correr serviços em contentores. Isto inclui implementação, escalabilidade, rede e balanço de cargas.

Devido aos contentores serem leves e efémeros por natureza, corrê-los em produção pode-se rapidamente tornar num grande esforço, particularmente quando se lida com microserviços, pois cada um corre no seu próprio contentor. Isto leva a que, numa aplicação com centenas ou milhares de serviços, a complexidade do sistema seja demasiado grande para este ser gerido manualmente.

Os benefícios da orquestração de contentores são então uma maior simplicidade de operações, maior resiliência do sistema e maior segurança, ao reduzir a probabilidade de erro humano.

3. Cerimónias Scrum

Durante o período de estágio tive imensas reuniões com a equipa. Fazíamos principalmente quatro tipos diferentes de reuniões:

Dailies: Todos os dias às 10h fazíamos uma reunião curta de cerca de 10 a 15 minutos onde cada membro da equipa dizia o que tinha feito no dia anterior acerca do projeto, o que iria fazer nesse dia, e quais eram os seus bloqueios, caso existissem.

Sprint Planning: Reuniões feitas antes de se iniciar cada sprint onde se planeavam as user stories que entravam nesse sprint.

Groomings: Reuniões só entre os programadores, onde se atribuíam os story points a cada story. Cada um votava no número de story points que achava sensato para cada story e no final chegávamos a acordo para um número definitivo. Usámos o site https://www.planitpoker.com para fazermos a votação (figura 14).

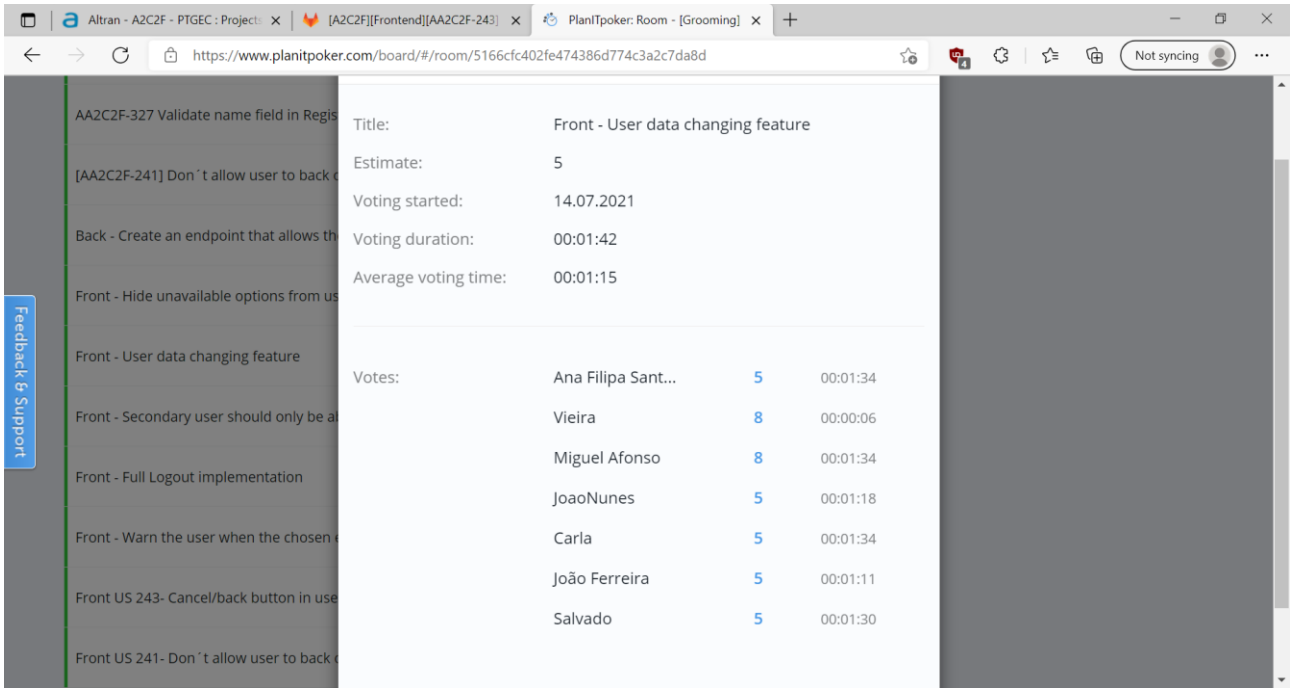

Figura 14: Votação dos Story Points para uma User Story

Demos: No final de cada sprint, as equipas de *Frontend* e de *Backend* apresentavam o que tinham feito nesse sprint ao *Product Owner* e ao *Project Manager*.

Para além destas reuniões, fizemos ainda muitas outras para esclarecimento de dúvidas entre os diferentes membros da equipa do projeto.

4. Tecnologias utilizadas

Android Studio

Android Studio é o ambiente de desenvolvimento integrado (IDE) oficial da *Google* para o sistema operativo Android. É baseado no software *IntelliJ IDEA* da *JetBrains* e foi feito especificamente para o desenvolvimento para Android.

O *Android Studio* foi a aplicação em que mais trabalhei durante o estágio, tendo escrito todo o código para o projeto neste software.

Jira

Jira é um software comercial desenvolvido pela *Atlassian*. É uma ferramenta que permite a monitorização de tarefas e acompanhamento de projetos garantindo a gestão de todas as suas atividades num único lugar.

No caso do projeto em que trabalhei, havia três quadros de tarefas diferentes. Cada um dos quadros tem a função de organizar as user stories de acordo com o seu estado. O primeiro é o *Definition Board*, utilizado no início do Sprint, onde as user stories são postas assim que criadas, até estas serem aceites pelo *Project Manager*.

O segundo, o *Development Board*, funciona durante o Sprint. Ao começar-se a trabalhar na user story, desloca-se de To Do para Working e quando se acaba, passa para Verification. No final, depois da Demo desse Sprint, se for aceite pelo *Product Owner* e *Project Manager*, é passada para Resolved.

O último quadro, o *Delivery Board* é utilizado no final do Sprint. Se as user stories que têm o estado de Resolved não precisarem de ser reabertas, são passadas para o estado de Closed, terminando assim o ciclo de vida da user story.

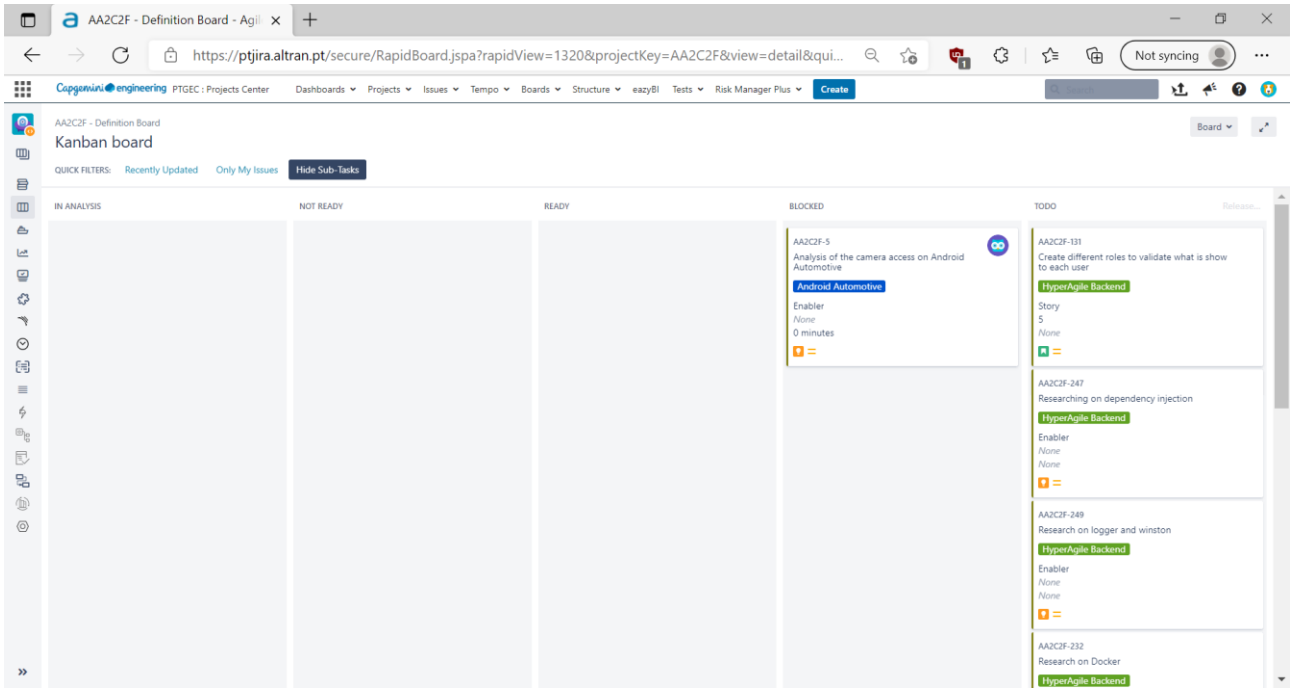

Figura 15: Definition Board

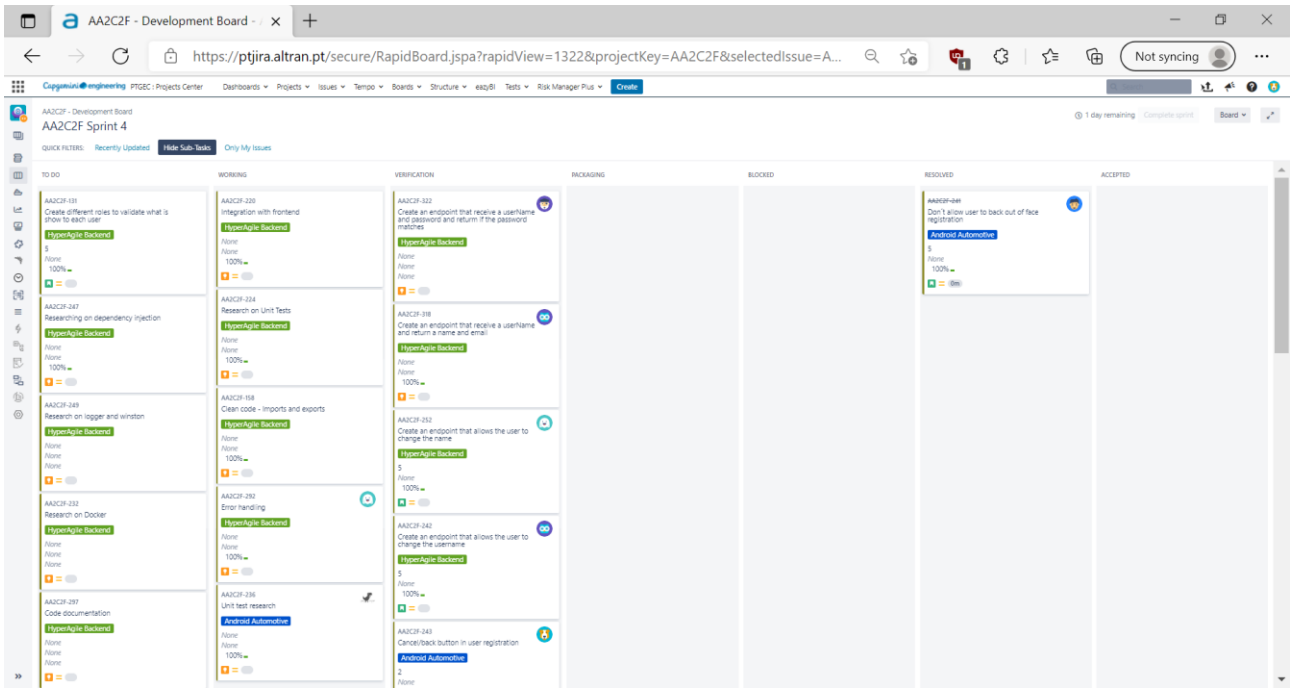

Figura 86: Development Board

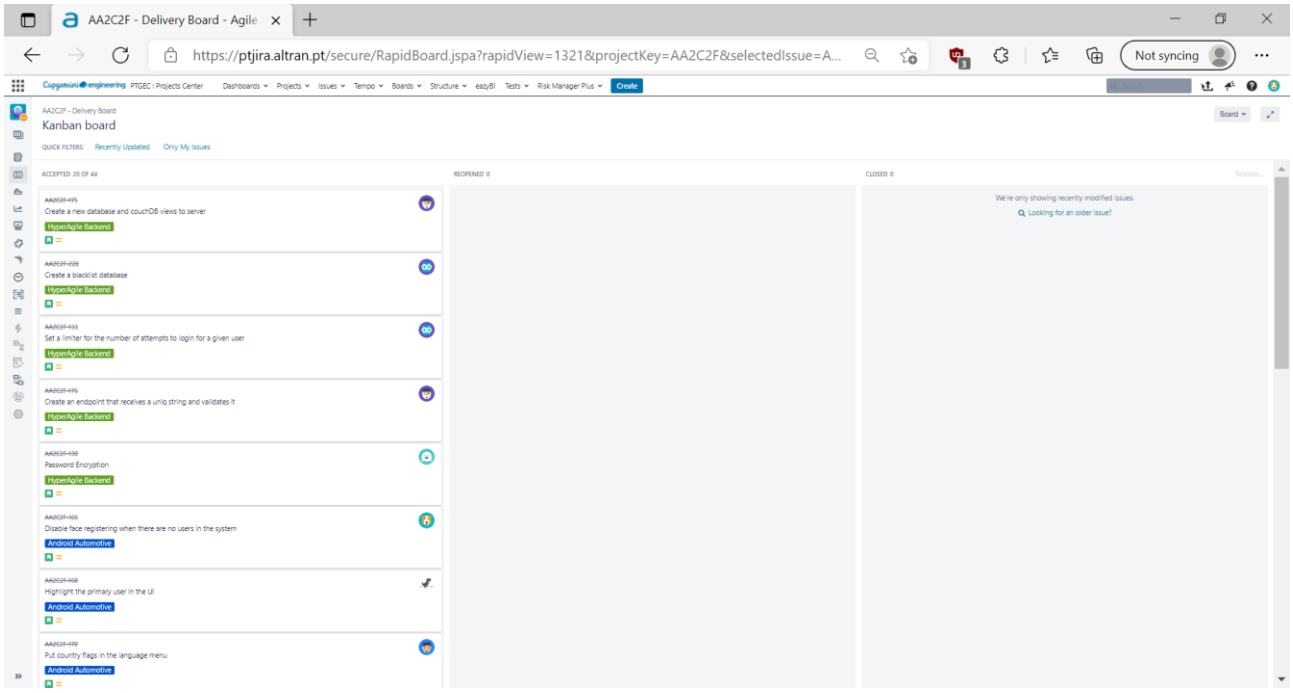

Figura 17: Delivery Board

O Jira tem também uma funcionalidade para inserir as horas trabalhadas no projeto.

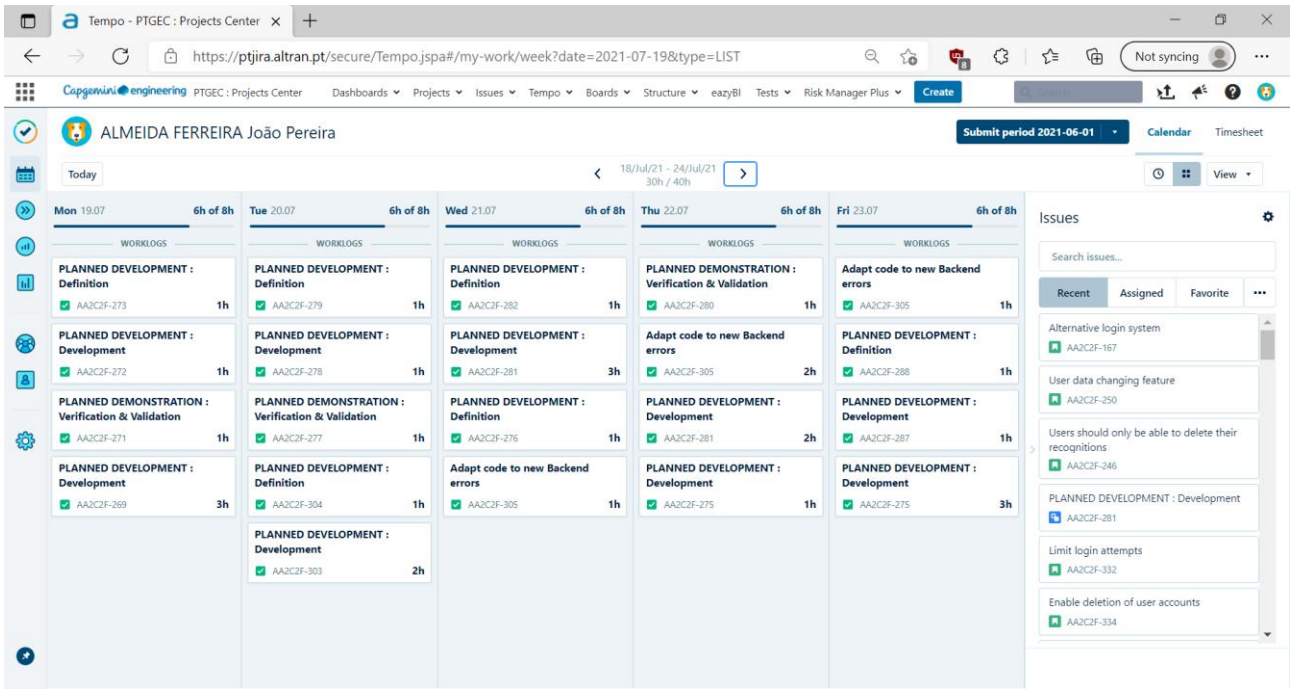

Figura 18: Horas do projeto

Gitlab

O *GitLab* é um gestor de repositório de software baseado em git, com suporte a Wiki e gestão de tarefas. O *GitLab* é semelhante ao *GitHub*, mas permite que *developers* armazenem o código nos seus próprios servidores, ao invés de servidores de terceiros.

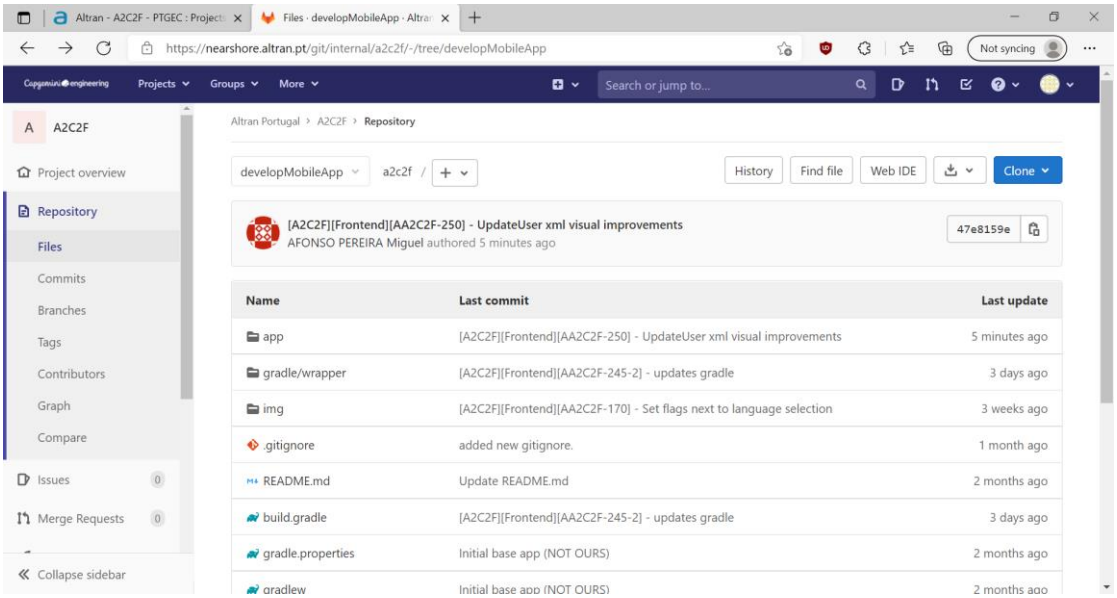

Figura 12: Um branch do projeto visto no GitLab

Durante o estágio, utilizei o *GitLab* principalmente para fazer *Merge Requests*, com o intuito de passar as mudanças que fiz ao projeto para outro colega da equipa rever e aprovar, antes de se fazer o *merge* para o *branch* onde estávamos a desenvolver o projeto.

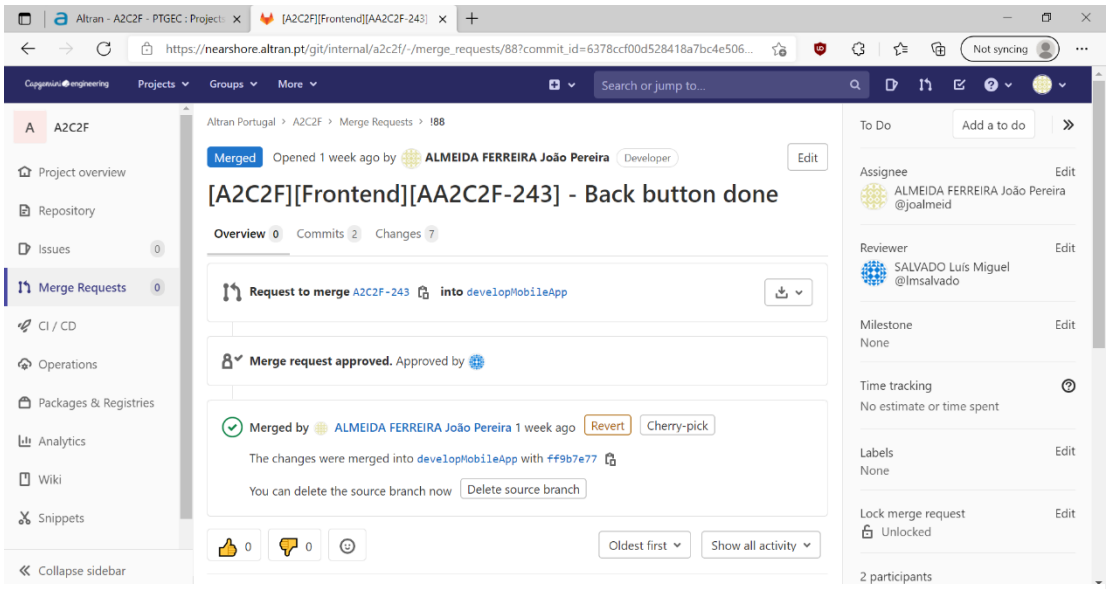

Figura 20: Merge Request no GitLab

Conclusão

Este estágio ajudou-me a colocar os conhecimentos que obtive durante o curso de Desenvolvimento de Aplicações em prática, num ambiente profissional. Foi um estágio rigoroso e desafiante, que me fez evoluir enquanto programador.
Permitiu-me aprofundar os meus conhecimentos em Android e Git e pôr em prática a metodologia Ágil. Também me deu a conhecer novas ferramentas e softwares que poderei ter de vir a usar durante a minha vida profissional.

Em geral, o estágio foi muito recompensador, pois no final fui chamado para continuar na empresa para fazer um estágio profissional. Estou, por isso, grato à Altran e ao IPG por esta oportunidade que me foi dada.

Referências Bibliográficas

As seguintes referências foram usadas para a pesquisa dos conceitos que expus no capítulo 2.

Microservices, "Pattern: Monolithic Architecture". Acedido a 12 de setembro de 2021 em: https://microservices.io/patterns/monolithic.html

Microservices, "Pattern: Microservice Architecture". Acedido a 12 de setembro de 2021 em: https://microservices.io/patterns/microservices.html

Red Hat, "O que é a arquitetura de microsserviços?". Acedido a 13 de setembro de 2021 em: https://www.redhat.com/pt-br/topics/microservices/what-are-microservices

OPUS Software, "Micro Serviços: Qual a diferença para a Arquitetura Monolítica?". Acedido a 13 de setembro de 2021 em: https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/

Docker, "What is a Container?". Acedido a 17 de novembro de 2021 em: https://www.docker.com/resources/what-container

Microsoft Azure, "What is a Container?". Acedido a 17 de novembro de 2021 em: https://azure.microsoft.com/pt-br/overview/what-is-a-container/#overview

VMWare, "What is container orchestration?". Acedido a 18 de novembro de 2021 em: https://www.vmware.com/topics/glossary/content/container-orchestration