

Instituto Politécnico do Cávado e do Ave

Projeto Aplicado

Relatório de Projeto Aplicado

Licenciatura em Engenharia de Sistemas
Informáticos

Diogo Oliveira
Nº 21111

Joshua Jones
Nº 21116

Leandro Matos
Nº 21124

Conteúdo

Lista de Figuras	4
1. Introdução	7
1.1. Enquadramento	7
1.2. Objetivos	7
1.3. Estrutura do documento	7
2. Contextualização	8
3. Estado de Arte	9
3.1. <i>Mapsted</i>	9
3.2. <i>Nearmotion</i>	9
3.3. <i>indoo.rs</i>	10
3.4. <i>what3words</i>	10
4. Proposta do Sistema	11
5. Prototipagem	12
6. Diagramas	14
6.1. Diagrama Casos de Uso	14
6.2. Diagrama de Atividades	16
6.3. Diagrama de Sequência por Ecrã	17
6.4. Diagrama de Entidade Relação	21
6.5. Diagrama de Atividades	22
7. Mockups	23
8. Implementação API	25
8.1. Source	25
8.1.1. Config	25
8.1.2. Errors	27
8.1.3. Routes	27
8.1.4. Services	28
8.2. Services/utilizador.js	29
8.3. Routes/utilizadores.js	30

8.4.	Middlewares.js.....	31
8.5.	Passport.js.....	31
8.6.	Router.js.....	32
8.7.	Config/routes.js.....	33
9.	Aplicação Movel	34
10.	Conclusão.....	45
11.	Bibliografia:.....	46

Lista de Figuras

Figura 1 - Mapsted.....	9
Figura 2 - Narmotion	9
Figura 3 - indoo.rs.....	10
Figura 4 - what3words.....	10
Figura 5 - Ecrã Mapa.....	12
Figura 6 - Ecra Login.....	12
Figura 7 - Ecrã Horário.....	13
Figura 8 - Ecrã Rota.....	13
Figura 9 - Diagrama Casos de Uso	14
Figura 10 - Comportamento Normal Diagrama Casos de Uso	14
Figura 11 - Diagrama Caso de Uso Login	15
Figura 12 - Diagrama de Atividades.....	16
Figura 13 - Diagrama de Sequência Ecrã Registo	17
Figura 14 - Diagrama de Sequência Ecrã Login	18
Figura 15 - Diagrama de Sequência Reset Password.....	18
Figura 16 - Diagrama de Sequência Ecrã Perfil.....	19
Figura 17 - Diagrama de Sequência Ecrã Horário.....	19
Figura 18 - Diagrama de Sequência Ecrã Pesquisa	20
Figura 19 - Diagrama de Entidade Relação.....	21
Figura 20 - Diagrama de Atividades Login	22
Figura 21 - Diagrama de Atividades Percurso	22
Figura 22 - Mockup Reset Password	23
Figura 23 - Mockup Login	23
Figura 24 - Mockup Register.....	23
Figura 25 - Mockup Encontrar Sala	24
Figura 26 - Mockup Ecrã Inicial.....	24
Figura 27 - Mockup Perfil	24
Figura 28 - Mockup Horário.....	24
Figura 29- source -config	25
Figura 30 - source - errors	27
Figura 31 - source - routes.....	27
Figura 32 - source - services	28
Figura 33 - Services/utilizador	29
Figura 34 - Routes/utilizadores	30
Figura 35 - Middlewares.....	31
Figura 36 - Passport	31
Figura 37 - Router	32

Figura 38 - Config/routes.....	33
Figura 39 - Código database.kt	35
Figura 40 - Código Sala Dao	36
Figura 41- Código LoginResponse e LoginRequest.....	37
Figura 42 - Código Classe Dados Sala	37
Figura 43 - Código LoginActivity PT1	38
Figura 44 - Código LoginActivity PT2	38
Figura 45 - Código ClassroomFinderActivity PT 1.....	39
Figura 46 - Código ClassroomFinderActivity PT 2.....	40
Figura 47 – Códigg Classe RoutingActivity.....	41
Figura 48 - Código PerfilActivity	42
Figura 49 - Login XML PT 1	43
Figura 50 - Login XML PT 2	44
Figura 51 - Activity Login	44

Lista de Acrónimos:

- API: Application Programming Interface;
- UI: User Interface;
- IPCA: Instituto Politécnico do Cávado e do Ave;
- IPS: Indoor Positioning System;
- SiGES: Sistema de Gestão de Ensino Superior;
- SIG: Sistema de Informação Geográfica.

1. Introdução

Este projeto consiste num IPS com localização exata dos utilizadores dentro das infraestruturas do Instituto Politécnico do Cávado e do Ave. Esta aplicação permitirá, com acesso a dados facultados pelos Serviços Administrativos, saber o horário do utilizador e fazer a rota otimizada de modo que este chegue ao seu destino.

1.1. Enquadramento

A ideia deste projeto surgiu no primeiro dia de aulas onde foi proposto a dois elementos deste grupo guiar os alunos novos dentro do IPCA. Tendo as cadeiras de Projeto aplicado e também Programação de Dispositivos Móveis decidimos que seria uma mais-valia, tanto para o nosso percurso académico, como também para o IPCA criarmos uma aplicação que conseguisse certificar que nenhum aluno ou docente teria algum problema a encontrar o seu destino no IPCA.

1.2. Objetivos

O objetivo deste projeto é conseguirmos ter as capacidades para criarmos uma aplicação que satisfaça os objetivos que temos para a mesma. Ao concretizarmos esses objetivos iremos ganhar vários conhecimentos, tanto a nível da programação, como competências para o nosso futuro profissional. Estas competências a nível profissional serão adquiridas trabalhando em grupo, e desempenhado papeis associados ao mundo do trabalho e trocando papeis entre elementos do grupo para cada um sair deste projeto com as competências desse papel.

1.3. Estrutura do documento

O documento encontra-se organizado em 5 capítulos sendo estes a Introdução, Estado de Arte, Conteúdo dos Ficheiros, Realização do Trabalho, Conclusão e Referencias.

2. Contextualização

Quanto à contextualização será abordado o cliente, utilizadores, melhorias no dia a dia dos utilizadores e as partes interessadas e não interessadas neste projeto.

a) Cliente:

O cliente do projeto é o IPCA.

b) Utilizadores:

Os utilizadores do projeto são os alunos, docentes e visitantes do IPCA.

c) Melhorias no dia a dia dos utilizadores:

O utilizador desta aplicação nunca terá problemas em encontrar a sala onde pretende estar e consoante o tipo de utilizador poderá saber exatamente onde tem de estar em específicas horas.

d) Partes interessadas:

As partes interessadas deste projeto é o IPCA, financiador do mesmo e único cliente visto que o projeto é feito para o Campus de Barcelos.

3. Estado de Arte

No presente capítulo será apresentado o estado da arte, relativo a tecnologias, aplicações e recursos que atualmente se encontram disponíveis para a realização/implementação de metodologias colaborativas.

3.1. *Mapsted*



Figura 1 - Mapsted

A Mapsted é uma empresa inovadora de tecnologia baseada no Canadá que tem clientes em várias indústrias tais como centros comerciais, universidades, hospitais, estações de comboio, resorts, etc., que usam uma tecnologia avançada para a localização dentro e fora de edifícios e que não necessita de *hardware* sendo so necessário um *smartphone* sem custos acrescentados.

3.2. *Nearmotion*



Figura 2 -
Nearmotion

A Nearmotion é uma empresa baseada na Arabia Saudita que em parceria com a *Saudi Aramco Entrepreneurship Ventures* para providenciar soluções de confiança que capacitam organizações governamentais e setores empresariais com ferramentas pioneiras que levam o envolvimento e experiência dos clientes a um novo nível.

Edifícios inteligentes que interagem com os visitantes de acordo com a sua localização, dão-lhes as boas-vindas à chegada, orientam-nos passo a passo através da orientação digital até ao seu destino e recompensam-nos com ofertas e cupões de acordo com as suas preferências. É isso que o NEARMOTION oferece por meio de uma plataforma fácil de usar que permite que os locais ofereçam níveis avançados de experiência aos visitantes e tornem a sua visita uma viagem inesquecível.

3.3. *indoo.rs*



Figura 3 - indoo.rs

A indoo.rs foi fundada em 2010 com o nome de CustomLBS por 2 alunos, *Bernd Gruber* e *Markus Krainz*.

A ideia nasceu quando, numa escala extremamente longa num aeroporto, *Bernd* achou mais difícil do que deveria ser localizar os edifícios e salas certas.

Desde então, a empresa cresceu para 20 funcionários, com sede e um escritório de desenvolvimento em Viena e um escritório de vendas em San Francisco.

Em fevereiro de 2019, a empresa foi adquirida pela *Esri*, fornecedora líder internacional de *software* SIG (sistema de informação geográfica).

3.4. *what3words*



Figura 4 - what3words

Endereços de ruas não são precisos o suficiente para especificar locais precisos, como entradas de prédios, e não existem para parques e muitas áreas rurais.

Isso dificulta a localização de lugares e impede que as pessoas descrevam exatamente onde a ajuda é necessária em caso de emergência.

Para resolver esses problemas foi criado o what3words, que consiste em dividir o mundo em 3 metros quadrados e dá a cada quadrado uma combinação única de três palavras. É a maneira mais fácil de encontrar e partilhar localizações exatas.

4. Proposta do Sistema

1) Requisitos Funcionais:

- a. O utilizador devia de ser capaz de conseguir chegar ao seu destino, independentemente de onde se situar dentro do campus;
- b. O utilizador deve ser capaz de inserir as suas credenciais para aceder às funcionalidades da aplicação;
- c. O utilizador deve ser capaz de aceder ao seu horário de aulas;
- d. O sistema deve ser capaz de obter posição do utilizador;
- e. O sistema deve ser capaz de atualizar a posição atual do utilizador;
- f. O sistema deve ser capaz de obter os horários de todos os cursos e anos do IPCA;
- g. O sistema deve ser capaz de obter e reconhecer a planta do IPCA.

2) Requisitos Não Funcionais

- a. Usabilidade: Um utilizador deverá conseguir operar o sistema sem necessitar de um guia do mesmo;
- b. Eficiência: O programa deve ser capaz de conseguir atualizar as rotas sem perder o destino;
- c. Confiabilidade: O sistema terá de estar sempre operacional, tendo as exceções de raras pausas de manutenção;
- d. Portabilidade: O programa deverá conseguir correr em todos os dispositivos *Android*;
- e. Implementação: O programa deverá ser desenvolvido na linguagem *Kotlin*;
- f. Interoperabilidade: O sistema deverá obter as suas informações através do *Maps* da *Google* e com o *SiGES* (Sistema de Gestão de Ensino Superior) para obter os horários do docentes e alunos;

- g. Legais: O programa deverá atender às normas legais, tais como padrões, leis, etc.

5. Prototipagem

Visual Paradigm Standard (Diogo Oliveira/Instituto Politécnico do Cávado e do Ave)

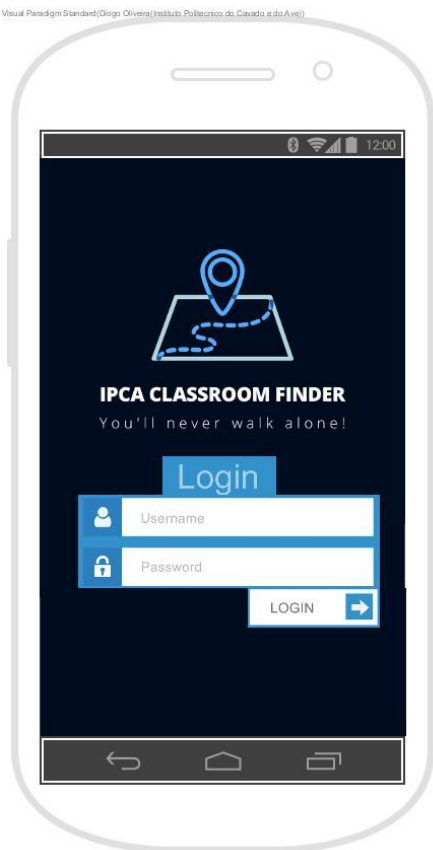


Figura 6 - Ecrã Login

Visual Paradigm Standard (Diogo Oliveira/Instituto Politécnico do Cávado e do Ave)

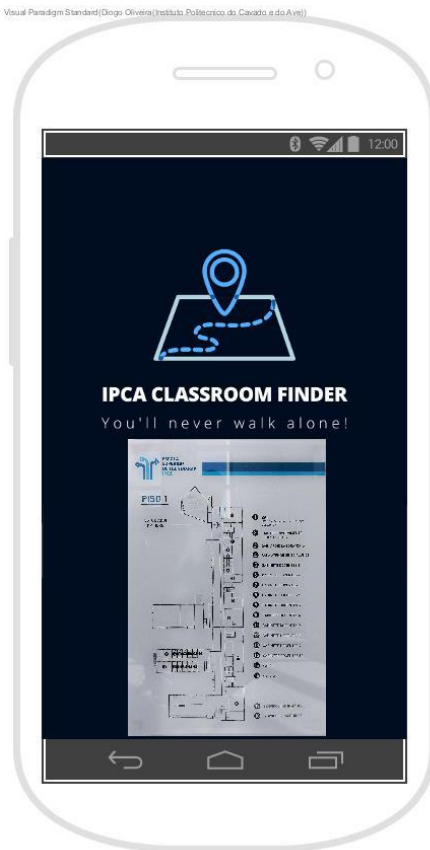


Figura 5 - Ecrã Mapa



Figura 8 - Ecrã Rota

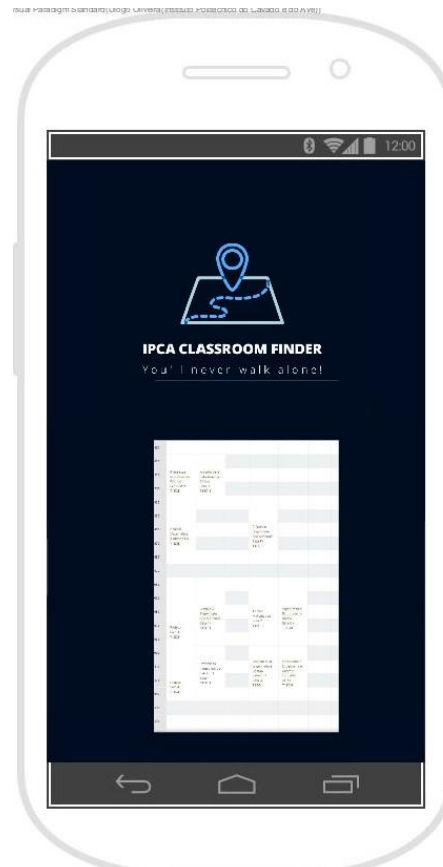


Figura 7 - Ecrã Horário

6. Diagramas

6.1. Diagrama Casos de Uso

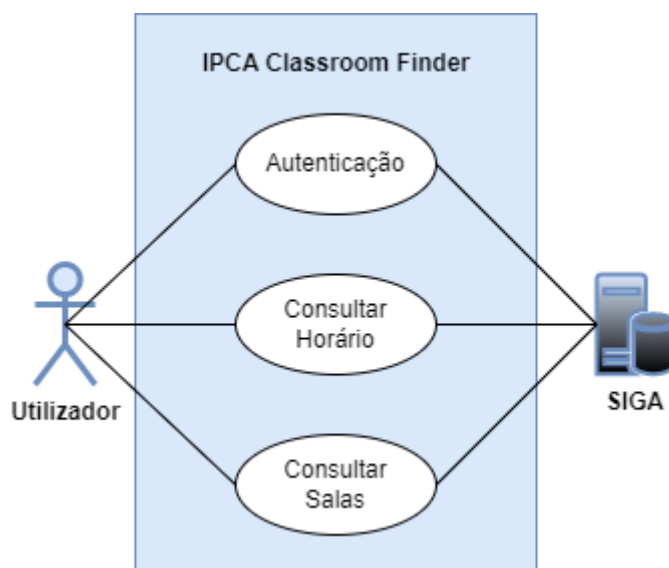


Figura 9 - Diagrama Casos de Uso

Caso de Uso: Gerir Usuários da Aplicação		
Descrição: Usuário utiliza a aplicação para obter direções		
Pré-Condição: Aplicação tem mapas		
	Actor	Sistema
Comportamento Normal	1. Apresenta o horário	
		2. Valida as credenciais
		3. Apresenta a sala
	4. Indica a Sala	
		5. Indica o caminho até à sala
		6. Fornece gps dinâmico
	7. Desloca-se até à sala	

Figura 10 - Comportamento Normal Diagrama Casos de Uso

UML Diagrama Caso de Uso: Login na Aplicação

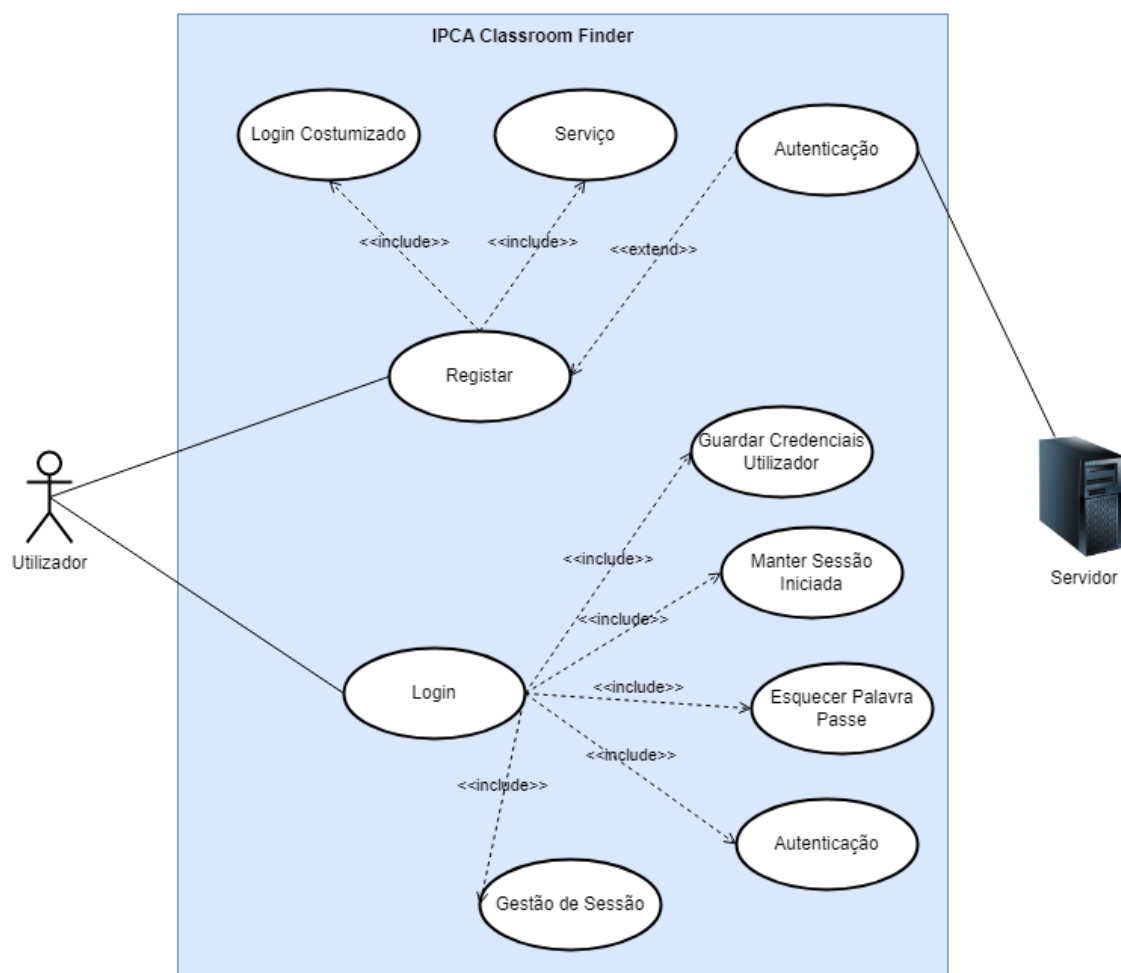


Figura 11 - Diagrama Caso de Uso Login

6.2. Diagrama de Atividades

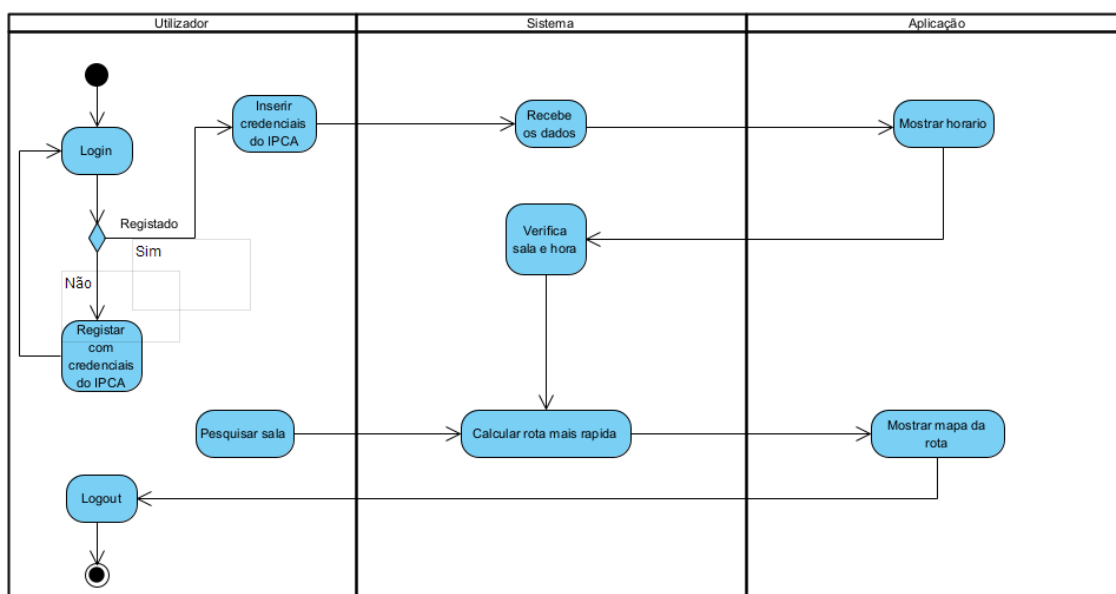


Figura 12 - Diagrama de Atividades

6.3. Diagrama de Sequência por Ecrã

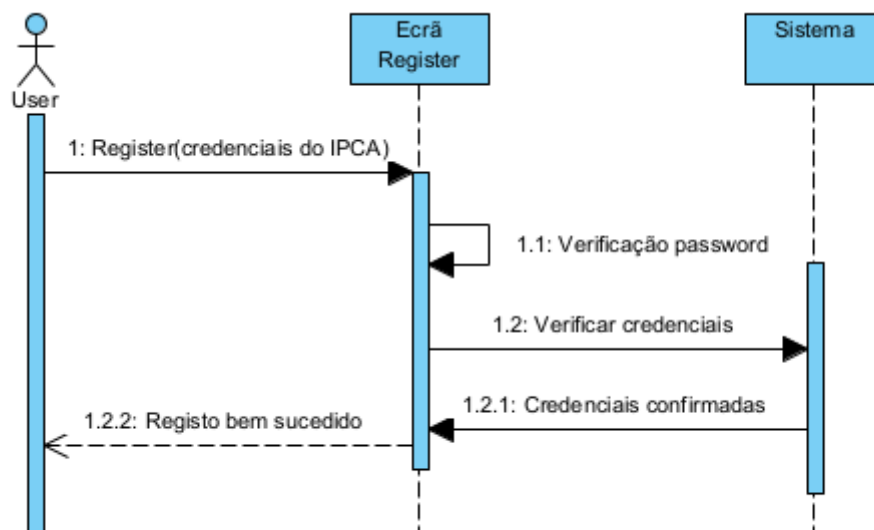


Figura 13 - Diagrama de Sequência Ecrã Registo

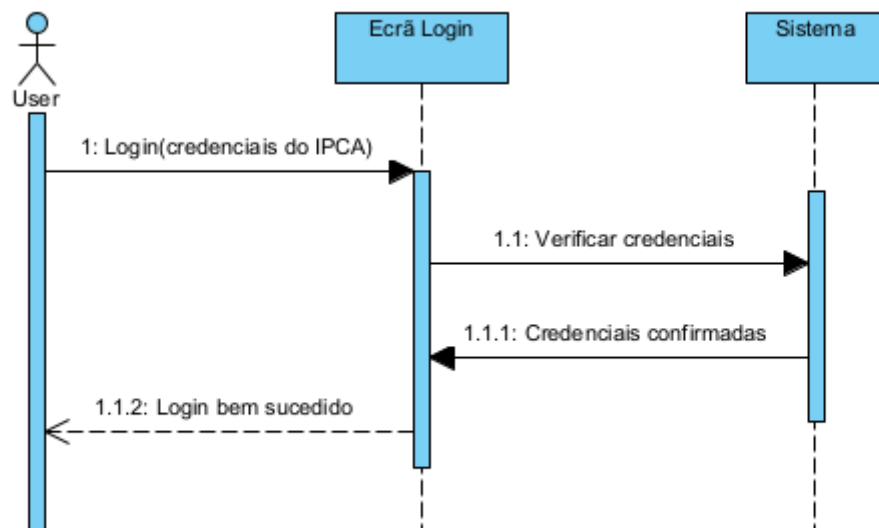


Figura 14 - Diagrama de Sequência Ecrã Login

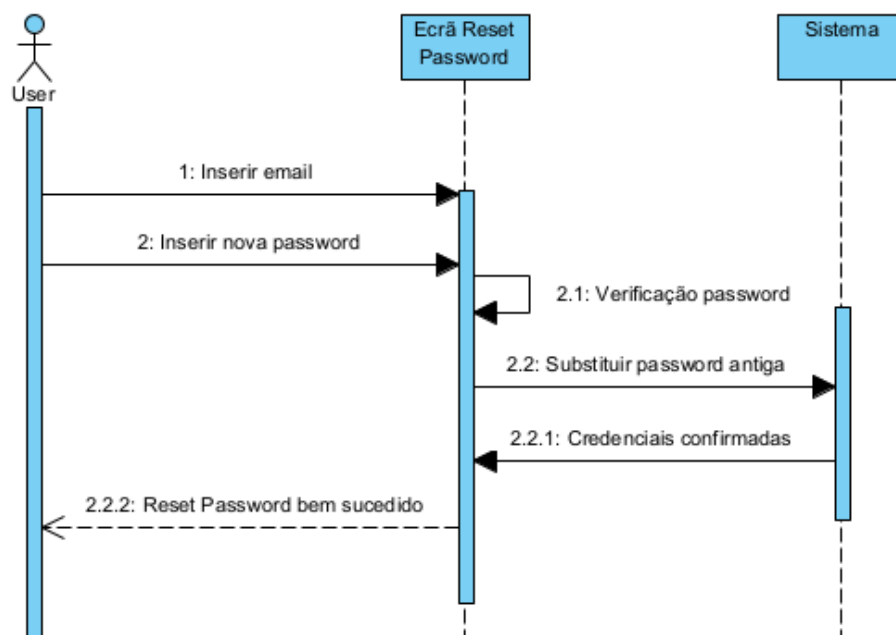


Figura 15 - Diagrama de Sequência Reset Password

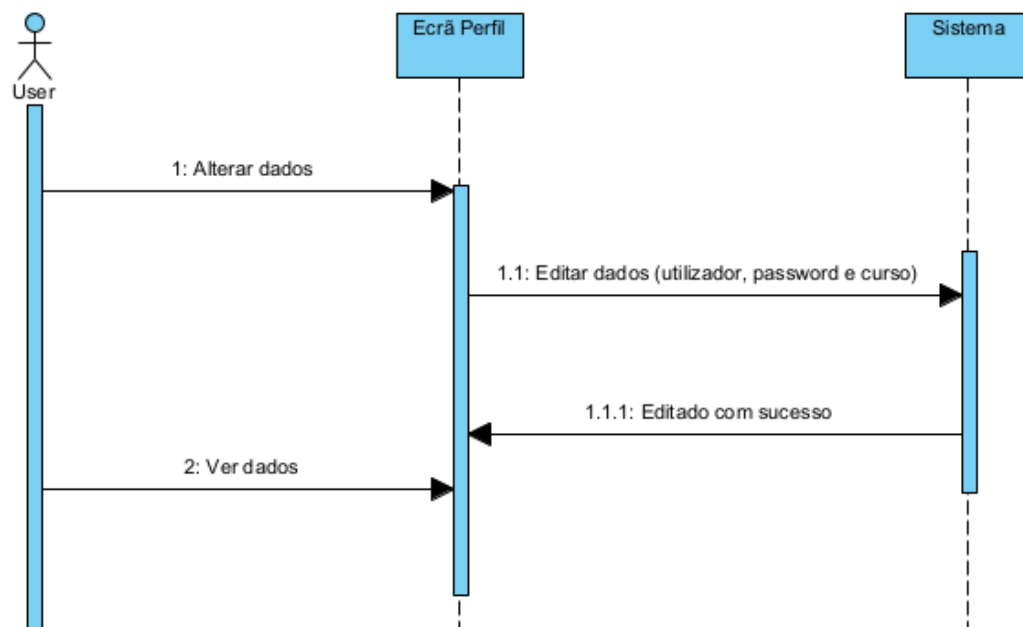


Figura 16 - Diagrama de Sequência Ecrã Perfil

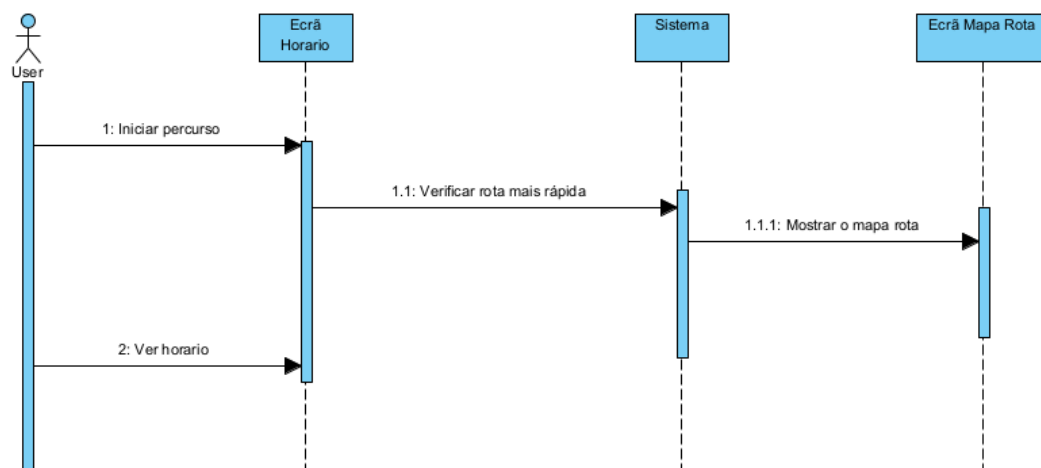


Figura 17 - Diagrama de Sequência Ecrã Horário

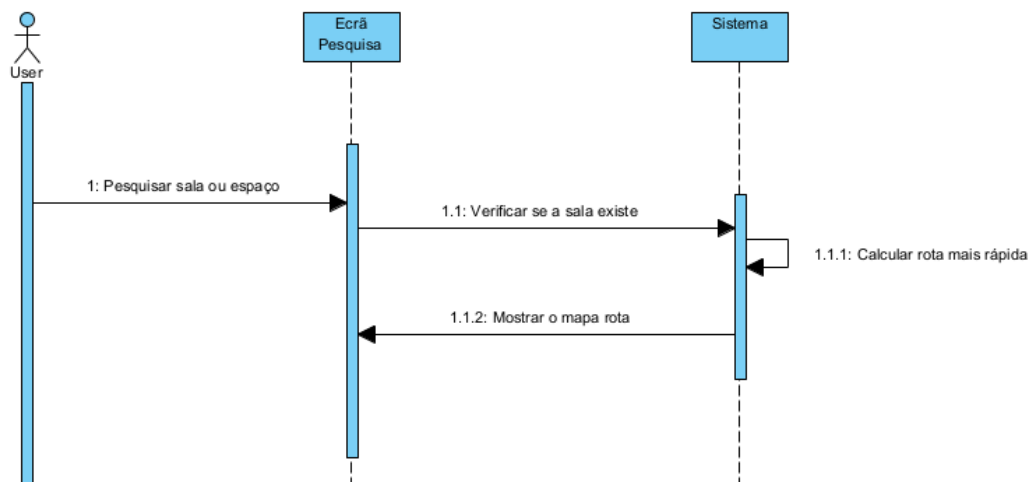


Figura 18 - Diagrama de Sequência Ecrã Pesquisa

6.4. Diagrama de Entidade Relação

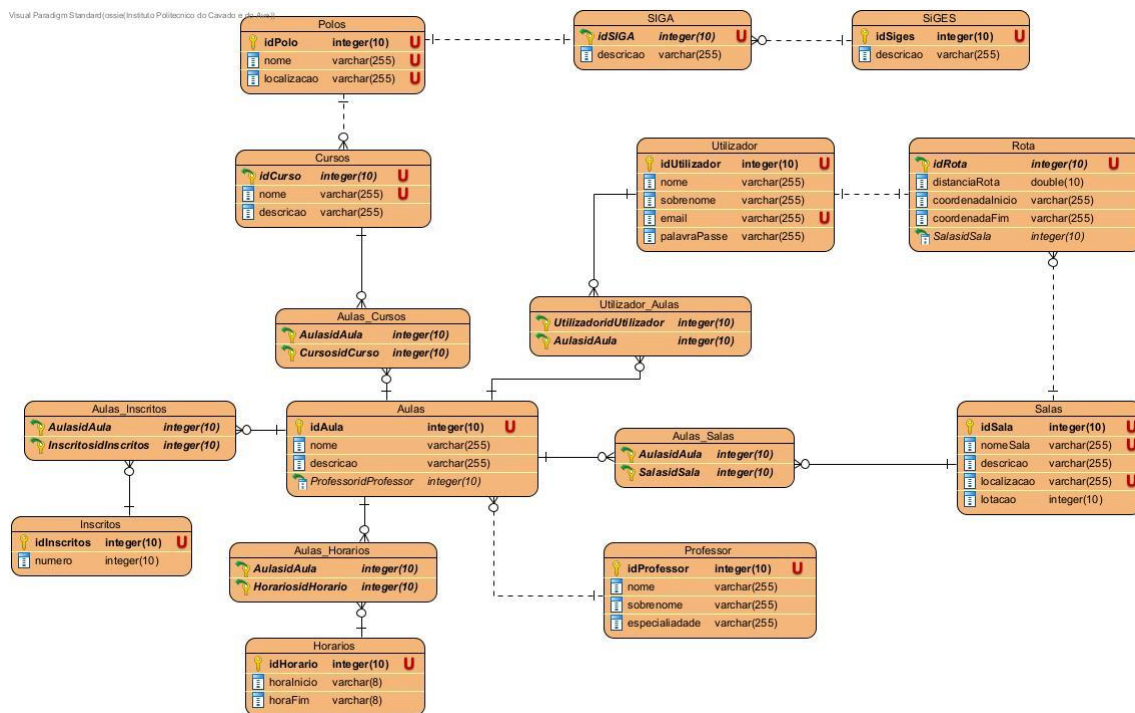
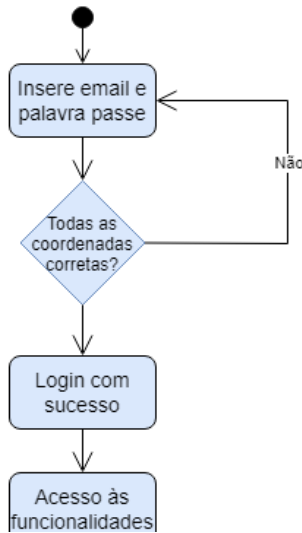


Figura 19 - Diagrama de Entidade Relação

6.5. Diagrama de Atividades

UML Diagrama de Atividades: Login na Aplicação



UML Diagrama de Atividades: Percurso até sala com horário

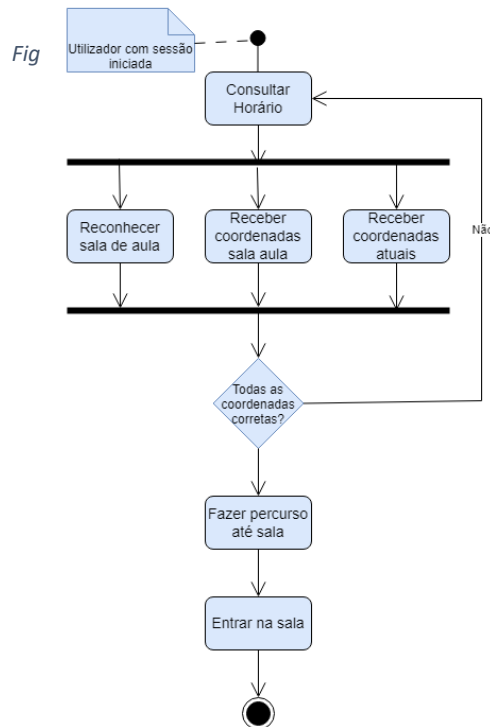
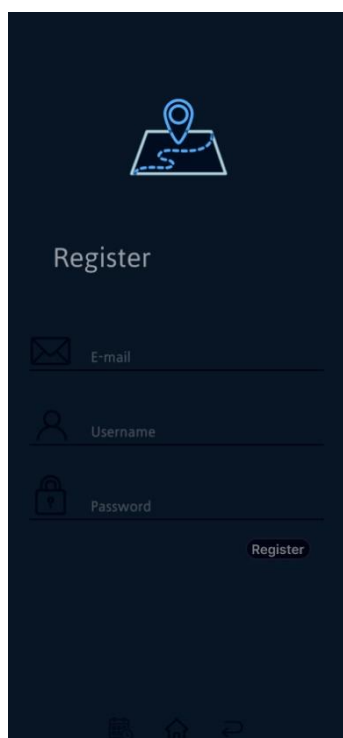


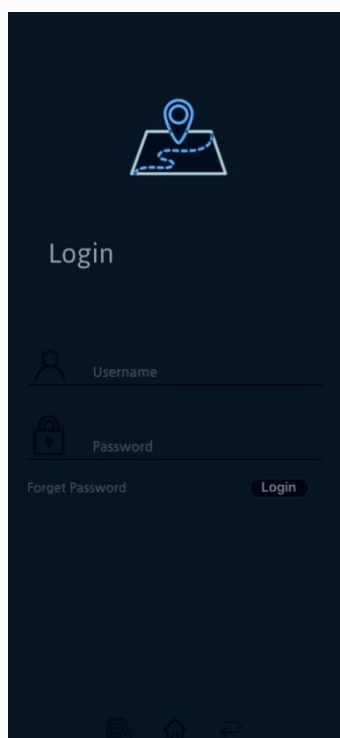
Figura 21 - Diagrama de Atividades Percurso

7. Mockups



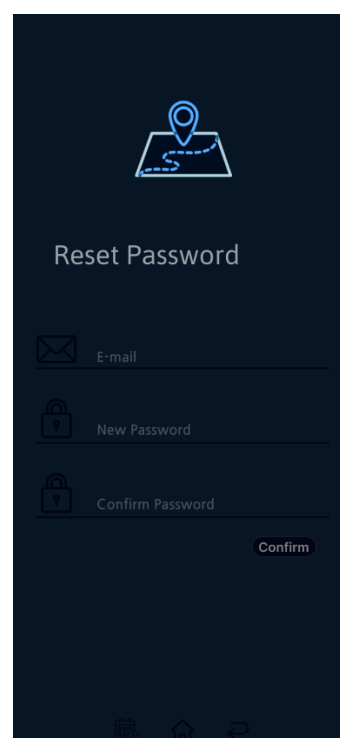
The Register screen features a dark blue background with a white location pin icon at the top. Below the icon, the title "Register" is displayed. The form includes three input fields: "E-mail" (with an envelope icon), "Username" (with a person icon), and "Password" (with a lock icon). A "Register" button is positioned at the bottom right of the form. At the very bottom, there is a navigation bar with three icons: a location pin, a house, and a car.

Figura 24 - Mockup Register



The Login screen has a dark blue background with a white location pin icon at the top. Below the icon, the title "Login" is displayed. The form includes two input fields: "Username" (with a person icon) and "Password" (with a lock icon). A "Login" button is located at the bottom right of the form. A "Forget Password" link is positioned below the password field. At the very bottom, there is a navigation bar with three icons: a location pin, a house, and a car.

Figura 23 - Mockup Login



The Reset Password screen features a dark blue background with a white location pin icon at the top. Below the icon, the title "Reset Password" is displayed. The form includes three input fields: "E-mail" (with an envelope icon), "New Password" (with a lock icon), and "Confirm Password" (with a lock icon). A "Confirm" button is positioned at the bottom right of the form. At the very bottom, there is a navigation bar with three icons: a location pin, a house, and a car.

Figura 22 - Mockup Reset Password

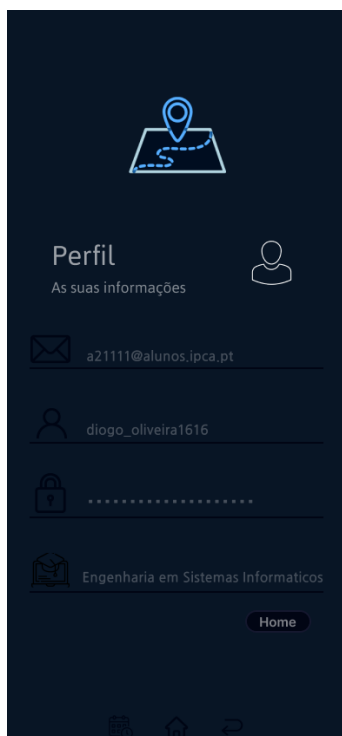


Figura 27 - Mockup Perfil

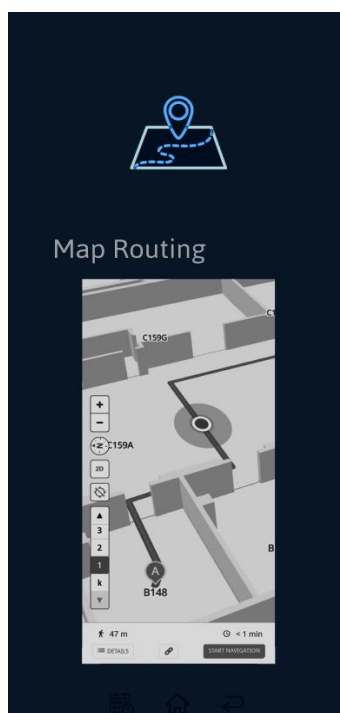


Figura 26 - Mockup Ecrã Inicial

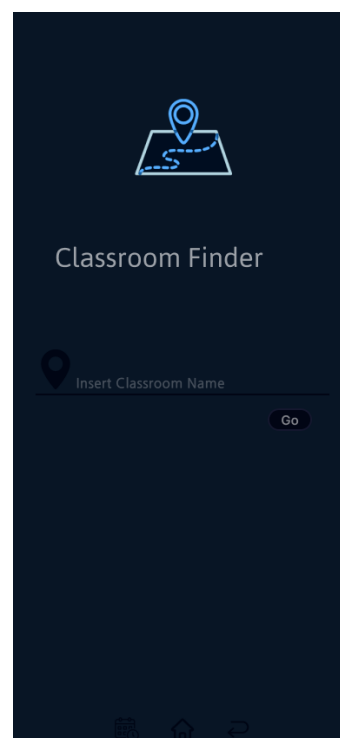


Figura 25 - Mockup Encontrar Sala



Figura 28 - Mockup Horário

8. Implementação API

8.1. Source

8.1.1. Config

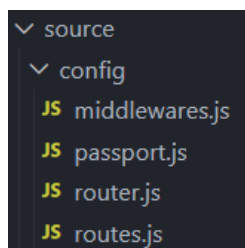


Figura 29- source -config

A pasta config guarda os ficheiros .js que configuram a API

Middlewares:

No contexto do desenvolvimento web, *middleware* refere-se ao software que fica entre o sistema operacional e uma aplicação, realizando uma função específica ou um conjunto de funções. Numa aplicação web, as funções de *middleware* são funções que têm acesso aos objetos de solicitação e resposta e podem realizar uma variedade de tarefas, como autenticação, log, servir arquivos estáticos, entre outros.

No contexto desta aplicação JavaScript web, as funções de *middleware* são escritas num arquivo chamado `middleware.js` e usado em conjunto com um framework web, como o Express.js. Essas funções de *middleware* podem ser consideradas como um *request* que tem a habilidade de executar código, tomar decisões e realizar tarefas antes que a solicitação seja tratada pela aplicação principal.

Passport:

Passport.js é um middleware de autenticação para o Node.js que oferece uma maneira simples e flexível de autenticar utilizadores e proteger rotas. Ele usa "estratégias" para autenticar solicitações, que são funções que verificam as credenciais de um usuário e fornecem um objeto de usuário após a autenticação bem-sucedida.

O Passport.js suporta muitos mecanismos de autenticação diferentes, chamadas "estratégias", incluindo OAuth, SAML e OpenID Connect. Ele também fornece uma API simples para autenticar solicitações e suporta mais de 500 mecanismos de autenticação.

Router:

Router.js é um módulo de software que é comumente usado em aplicações web para gerir rotas (caminhos para diferentes páginas ou recursos da aplicação). Ele pode ser usado para mapear rotas para diferentes funções de manipulação de requisições HTTP (como GET, POST, PUT e DELETE), que podem ser usadas para ler e escrever dados em um banco de dados ou para realizar outras operações no lado do servidor.

Routes:

Um arquivo routes.js é um arquivo de configuração de rotas em uma aplicação web. Ele é usado para definir as rotas e as funções de manipulação de requisições HTTP (como GET, POST, PUT e DELETE) que estão associadas a elas.

8.1.2. Errors

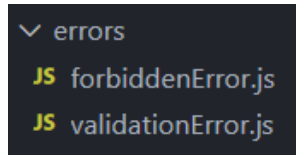


Figura 30 - source - errors

O ficheiro “Errors” é constituído por ficheiros .js usados para apresentar mensagens de erro sempre que um erro for encontrado num pedido à API.

8.1.3. Routes

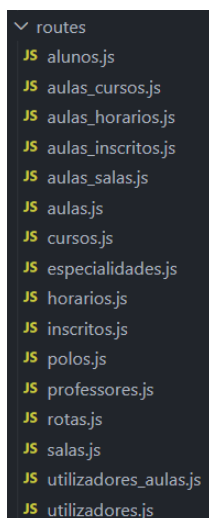


Figura 31 - source - routes

A pasta de *Routes* é constituída por todos os ficheiros *JavaScript* com o código que executa o *CRUD* da API, fazendo estas consoantes as instruções criadas nos *Services*.

8.1.4. Services

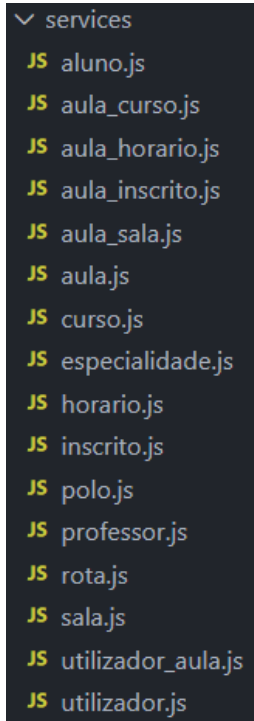


Figura 32 - source - services

A pasta de *Services* é constituída por código que executa operações de CRUD para cada tabela específica da base de dados.

8.2. Services/utilizador.js

```
const ValidationError = require('../errors/validationError');

module.exports = (app) => {
  const findOne = (filter = {}) => {
    return app.db('utilizador').where(filter).select(['idUtilizador', 'nome', 'sobrenome', 'email', 'palavraPasse']);
  }

  /**Selecionar todos os utilizadores */
  const getAll = async () => {
    return app.db('utilizador').select(['*']);
  };

  /**Filtragem apenas os utilizadores por ID */
  const getAllID = async (filter) => {
    return app.db('utilizador').where(filter).select(['*']);
  };

  /**Criação do registo de um novo utilizador */
  const create = async (req, res) => {
    if(!req.nome) throw new ValidationError('O Nome é um campo obrigatorio');
    if(!req.sobrenome) throw new ValidationError('O Sobrenome é um campo obrigatorio');
    if(!req.email) throw new ValidationError('O E-mail é um campo obrigatorio');
    if(!req.palavraPasse) throw new ValidationError('A Palavra Passe é um campo obrigatorio');

    const newUtilizador = {...req};
    return app.db('utilizador').insert(newUtilizador, ['nome', 'sobrenome', 'email', 'palavraPasse']);
  };

  /**Atualizar o utilizador selecionado */
  const update = async (id, utilizador) => {
    return app.db('utilizador').where({ id }).update(utilizador, ['nome', 'sobrenome', 'email', 'palavraPasse']);
  };

  /**Remover um utilizador */
  const remove = async (id) => {
    return app.db('utilizador').where(id).del();
  };

  return {
    findOne,
    getAll,
    getAllID,
    create,
    update,
    remove,
  };
};
```

Figura 33 - Services/utilizador

Os Services do Utilizador.js fazem 6 operações:

- *findOne*: Aqui, através de um filtro vindo do *frontend* da *API*, o utilizador consegue encontrar outro utilizador na base de dados a partir de um filtro que ele queira usar para pesquisar.
- *getAll*: Código de uma operação GET para o utilizador receber os dados todos de todos os utilizadores que estão inseridos na base de dados.
- *getAllID*: Código de uma operação GET para o utilizador receber os dados todos de um certo utilizador através do seu ID.
- *create*: Código de uma operação POST onde um utilizador insere os dados de um novo utilizador. O campo *idUtilizador* é auto-incrementado, o restante campo tem de ser ~~inc~~ não o programa retorna uma mensagem de erro a pedir os dados em falta.
- *update*: Código de uma operação PUT que, a partir de novos dados inseridos, atualiza o utilizador selecionado através de um ID.
- *remove*: Código de uma operação REMOVE que apaga um utilizador a partir de um ID inserido, isto claro se coincidir.

8.3. Routes/utilizadores.js

```
const express = require('express');

module.exports = (app) => {
  const router = express.Router();

  router.get('/', (req, res, next) => {
    app.services.utilizador.getAll()
      .then((result) => res.status(200).json(result))
      .catch((err) => next(err));
  });

  router.post('/', async (req, res, next) => {
    try {
      const result = await app.services.utilizador.create(req.body);
      return res.status(201).json(result[0]);
    } catch (err) {
      return next(err);
    }
  });

  router.get('/:id', (req, res, next) => {
    app.services.utilizador.getAllID({ id: req.params.id })
      .then((result) => res.status(200).json(result))
      .catch((err) => next(err));
  });

  router.delete('/:id', (req, res, next) => {
    app.services.utilizador.remove({ id: req.params.id })
      .then((result) => res.status(204).json(result[0]))
      .catch((err) => next(err));
  });

  router.put('/:id', (req, res, next) => {
    app.services.utilizador.update(req.params.id, req.body)
      .then((result) => res.status(204).json(result[0]))
      .catch((err) => next(err));
  });

  return router;
};
```

O *Routes/Utilizadores.js* define o que cada rota faz, consoante as operações criadas nos *Services* do mesmo.

30

Figura 34 - Routes/utilizadores

8.4. Middlewares.js

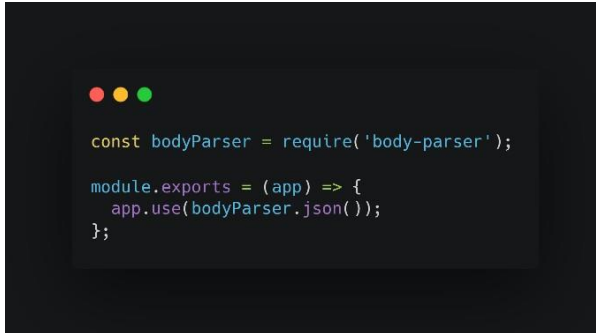


Figura 35 - Middlewares

8.5. Passport.js



Figura 36 - Passport

8.6. Router.js

```
const express = require('express');

module.exports = (app) => {
  app.use('/auth', app.routes.utilizadores);

  const secureRouter = express.Router();

  secureRouter.use('/utilizador', app.routes.utilizadores);
  secureRouter.use('/aula', app.routes.aulas);
  secureRouter.use('/curso', app.routes.cursos);
  secureRouter.use('/horario', app.routes.horarios);
  secureRouter.use('/inscrito', app.routes.inscritos);
  secureRouter.use('/polo', app.routes.polos);
  secureRouter.use('/professor', app.routes.professores);
  secureRouter.use('/rota', app.routes.rotas);
  secureRouter.use('/sala', app.routes.salas);
  secureRouter.use('/aluno', app.routes.alunos);
  secureRouter.use('/aulaCurso', app.routes.aulas_cursos);
  secureRouter.use('/aulaHorario', app.routes.aulas_horarios);
  secureRouter.use('/aulaInscrito', app.routes.aulas_inscritos);
  secureRouter.use('/aulaSala', app.routes.aulas_salas);
  secureRouter.use('/utilizadorAula', app.routes.utilizadores_aulas);
  secureRouter.use('/especialidade', app.routes.especialidades);

  app.use('/v1', secureRouter);
};
```

Figura 37 - Router

O *Router.js* é onde são definidas as rotas que no serviço web vão executar as funções do código dos *Services*.

8.7. Config/routes.js

```

module.exports = (app) => {
  // AUTH
  app.route('/auth/signin')
    .get(app.routes.utilizadores.getAll);

  app.route('/auth/signup')
    .post(app.routes.utilizadores.signup);

  // UTILIZADOR
  app.route('/utilizador')
    .all(app.config.passport.authenticate())
    .get(app.routes.utilizadores.getAll)
    .post(app.routes.utilizadores.create);

  app.route('/utilizador/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.utilizadores.findById)
    .put(app.routes.utilizadores.update)
    .delete(app.routes.utilizadores.remove);

  // AULA
  app.route('/aula')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas.getAll)
    .post(app.routes.aulas.create)
    .delete(app.routes.aulas.remove);

  app.route('/aula/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas.findById)
    .post(app.routes.aulas.create)
    .put(app.routes.aulas.update)
    .delete(app.routes.aulas.remove);

  // ALUNO
  app.route('/aluno')
    .all(app.config.passport.authenticate())
    .get(app.routes.alunos.getAll)
    .post(app.routes.alunos.create)
    .delete(app.routes.alunos.remove);

  app.route('/aluno/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.alunos.findById)
    .post(app.routes.alunos.create)
    .put(app.routes.alunos.update)
    .delete(app.routes.alunos.remove);

  // AULA_CURSO
  app.route('/aulaCurso')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas_cursos.getAll);

  app.route('/aulaCurso/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas_cursos.findById);

  // AULA_HORARIO
  app.route('/aulaHorario')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas_horarios.getAll)
    .post(app.routes.aulas_horarios.create)
    .delete(app.routes.aulas_horarios.remove);

  app.route('/aulaHorario/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas_horarios.findById)
    .post(app.routes.aulas_horarios.create)
    .put(app.routes.aulas_horarios.update)
    .delete(app.routes.aulas_horarios.remove);

  // AULA_INSCRITO
  app.route('/aulaInscrito')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas_inscritos.getAll)
    .post(app.routes.aulas_inscritos.create)
    .delete(app.routes.aulas_inscritos.remove);

  app.route('/aulaInscrito/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas_inscritos.findById)
    .post(app.routes.aulas_inscritos.create)
    .put(app.routes.aulas_inscritos.update)
    .delete(app.routes.aulas_inscritos.remove);

  // AULA_SALA
  app.route('/aulaSala')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas_salas.getAll)
    .post(app.routes.aulas_salas.create)
    .delete(app.routes.aulas_salas.remove);

  app.route('/aulaSala/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.aulas_salas.findById)
    .post(app.routes.aulas_salas.create)
    .put(app.routes.aulas_salas.update)
    .delete(app.routes.aulas_salas.remove);

  // CURSO
  app.route('/curso')
    .all(app.config.passport.authenticate())
    .get(app.routes.cursos.getAll)
    .post(app.routes.cursos.create)
    .delete(app.routes.cursos.remove);

  app.route('/curso/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.cursos.findById)
    .post(app.routes.cursos.create);

  // ESPECIALIDADE
  app.route('/especialidade')
    .all(app.config.passport.authenticate())
    .get(app.routes.especialidades.getAll)
    .post(app.routes.especialidades.create)
    .delete(app.routes.especialidades.remove);

  app.route('/especialidade/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.especialidades.findById)
    .post(app.routes.especialidades.create)
    .put(app.routes.especialidades.update)
    .delete(app.routes.especialidades.remove);

  // HORARIO
  app.route('/horario')
    .all(app.config.passport.authenticate())
    .get(app.routes.horarios.getAll)
    .post(app.routes.horarios.create)
    .delete(app.routes.horarios.remove);

  app.route('/horario/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.horarios.findById)
    .post(app.routes.horarios.create)
    .put(app.routes.horarios.update)
    .delete(app.routes.horarios.remove);

  // INSCRITO
  app.route('/inscrito')
    .all(app.config.passport.authenticate())
    .get(app.routes.inscritos.getAll)
    .post(app.routes.inscritos.create)
    .delete(app.routes.inscritos.remove);

  app.route('/inscrito/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.inscritos.findById)
    .post(app.routes.inscritos.create)
    .put(app.routes.inscritos.update)
    .delete(app.routes.inscritos.remove);

  // POLO
  app.route('/polo')
    .all(app.config.passport.authenticate())
    .get(app.routes.polos.getAll)
    .post(app.routes.polos.create)
    .delete(app.routes.polos.remove);

  app.route('/polo/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.polos.findById)
    .post(app.routes.polos.create)
    .put(app.routes.polos.update)
    .delete(app.routes.polos.remove);

  // PROFESSOR
  app.route('/professor')
    .all(app.config.passport.authenticate())
    .get(app.routes.professores.getAll)
    .post(app.routes.professores.create)
    .delete(app.routes.professores.remove);

  app.route('/professor/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.professores.findById)
    .post(app.routes.professores.create)
    .put(app.routes.professores.update)
    .delete(app.routes.professores.remove);

  // ROTA
  app.route('/rota')
    .all(app.config.passport.authenticate())
    .get(app.routes.rotas.getAll)
    .post(app.routes.rotas.create)
    .delete(app.routes.rotas.remove);

  app.route('/rota/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.rotas.findById)
    .post(app.routes.rotas.create)
    .put(app.routes.rotas.update)
    .delete(app.routes.rotas.remove);

  // SALA
  app.route('/sala')
    .all(app.config.passport.authenticate())
    .get(app.routes.salas.getAll)
    .post(app.routes.salas.create)
    .delete(app.routes.salas.remove);

  app.route('/sala/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.salas.findById)
    .post(app.routes.salas.create)
    .put(app.routes.salas.update)
    .delete(app.routes.salas.remove);

  // UTILIZADOR_AULA
  app.route('/utilizadorAula')
    .all(app.config.passport.authenticate())
    .get(app.routes.utilizadores_aulas.getAll)
    .post(app.routes.utilizadores_aulas.create)
    .delete(app.routes.utilizadores_aulas.remove);

  app.route('/utilizadorAula/:id')
    .all(app.config.passport.authenticate())
    .get(app.routes.utilizadores_aulas.findById)
    .post(app.routes.utilizadores_aulas.create)
    .put(app.routes.utilizadores_aulas.update)
    .delete(app.routes.utilizadores_aulas.remove);
};

```

Figura 38 - Config/routes

O código acima tem as rotas todas da API, com as operações que conseguem executar também as mesmas em cenários onde a pesquisa é por ID.

9. Aplicação MoveI

Room: O Room é um framework do Android Studio que facilita a criação de aplicativos com banco de dados SQLite. Ele fornece uma abstração de camada de persistência que permite aos desenvolvedores trabalhar com objetos Java simples ao invés de lidar diretamente com o banco de dados SQLite. O Room também fornece recursos como geração automática de código SQL, validação de consultas e suporte para transações.

Retrofit: Retrofit GSON é uma extensão do Retrofit, ele é utilizado para realizar requisições HTTP e converter as respostas em objetos Java, especificamente para o formato JSON. Ele utiliza a biblioteca GSON para serializar e desserializar os objetos Java automaticamente. GSON é uma biblioteca de código aberto criada pela Google, que é usada para converter objetos Java para formato JSON e vice-versa. A combinação do Retrofit e GSON permite que os desenvolvedores trabalhem com objetos Java simples ao invés de lidar diretamente com o formato JSON, facilitando a implementação de operações de rede e acesso a dados em aplicativos Android que se comunicam com APIs que retornam dados em formato JSON.

SQLite: SQLite é um banco de dados relacional de código aberto que é usado para armazenar e recuperar dados estruturados. Ele é embutido no sistema operacional Android, o que significa que ele não precisa ser instalado ou configurado separadamente. O SQLite é utilizado para gerenciar dados de aplicativos Android, tais como armazenamento de configurações, usuários, mensagens e outros tipos de dados. Ele é uma ferramenta útil para aplicativos que precisam armazenar dados localmente e trabalhar offline, sem a necessidade de acesso a uma conexão de rede. O Android Studio fornece suporte nativo para trabalhar com o SQLite, incluindo a capacidade de criar e gerenciar tabelas, consultas e transações.

Link do ilustre vídeo de utilização da aplicação: <https://youtu.be/nK-TcMnJOjQ>

Desenvolvimento do código:

```
@Entity(tableName = "utilizador")
data class utilizador(
    @PrimaryKey(autoGenerate = true) val idUtilizador: Int = 0,
    @ColumnInfo(name = "nome") val nome: String,
    @ColumnInfo(name = "sobrenome") val sobrenome: String,
    @ColumnInfo(name = "email") val email: String,
    @ColumnInfo(name = "palavraPasse") val palavraPasse: String)

@Dao
interface utilizadorDao {
    @Query("SELECT * FROM utilizador")
    fun getAll(): List<utilizador>
    @Query("SELECT * FROM utilizador WHERE idUtilizador = :id")
    fun findById(id: Int): utilizador
    @Insert
    fun insertAll(vararg utilizador: utilizador)
    @Delete
    fun delete(utilizador: utilizador)
    @Update
    fun updateUtilizador(utilizador: utilizador)
}

@Database(entities = arrayOf(utilizador::class), version = 1)
abstract class MyDatabase: RoomDatabase() {
    abstract fun utilizadorDao(): utilizadorDao
    companion object {
        @Volatile private var instance: MyDatabase? = null
        private val LOCK = Any()
        operator fun invoke(context: Context)= instance ?: synchronized(LOCK){
            instance ?: buildDatabase(context).also { instance = it}
        }
        private fun buildDatabase(context: Context) = Room.databaseBuilder(context,
            MyDatabase::class.java, "utilizador.db")
                .build()
    }
}
```

Figura 39 - Código database.kt

Este código cria uma classe de dados para um utilizador, com propriedades para armazenar informações como nome, sobrenome, email e palavra-passe, e uma interface para interagir com a tabela de utilizadores na base de dados. A classe MyDatabase é anotada como uma base de dados Room e fornece uma instância única da base de dados, "utilizador.db", que é construída usando o contexto passado. A interface utilizadorDao fornece métodos para buscar, inserir, atualizar e deletar utilizadores na tabela.

```
@Dao
interface SalaDao {
    @Query("SELECT * FROM sala")
    fun getAll(): List<sala>

    @Query("SELECT * FROM sala WHERE idSala = :idSala")
    fun findById(idSala: Int): sala

    @Insert
    fun insertAll(vararg sala: sala)

    @Delete
    fun delete(sala: sala)

    @Update
    fun updateSala(sala: sala)
}

@Database(entities = arrayOf(sala::class), version = 1)
abstract class MyDatabaseSala: RoomDatabase() {
    abstract fun SalaDao(): SalaDao
    companion object {
        @Volatile private var instance: MyDatabaseSala? = null
        private val LOCK = Any()
        operator fun invoke(context: Context)= instance ?: synchronized(LOCK){
            instance ?: buildDatabase(context).also { instance = it}
        }
        private fun buildDatabase(context: Context) = Room.databaseBuilder(context,
            MyDatabaseSala::class.java, "sala.db")
                .build()
    }
}
```

Figura 40 - Código Sala Dao

Este código fornece uma interface chamada SalaDao que possui métodos para interagir com a tabela sala na base de dados, e uma classe chamada MyDatabaseSala que é anotada como uma base de dados Room, e que possui uma instância única da base de dados chamada "sala.db". A classe MyDatabaseSala também define um método abstrato SalaDao() que retorna uma instância da interface SalaDao e o objeto companheiro fornece um operador invoke() para acessar a instância da base de dados criada.

```
data class LoginResponse(
    val token: String,
    val user: utilizador)

data class LoginRequest(
    val email: String,
    val password: String)

@Dao
interface LoginResponseDao{
    @POST("v1/auth/signin")
    fun checkLogin(@Body LoginRequest: LoginRequest): Call<LoginResponse>
}
```

Figura 41- Código LoginResponse e LoginRequest

Este código define duas classes de dados, LoginResponse e LoginRequest. A classe LoginResponse contém dois campos, token e user, enquanto a classe LoginRequest contém dois campos, email e password.

A interface LoginResponseDao contém um método chamado checkLogin que é anotado com @POST e tem uma url especificada("v1/auth/signin"). Este método espera um objeto LoginRequest como parâmetro e retorna uma chamada do tipo LoginResponse.

```
@Entity(tableName = "sala")
data class sala(
    @PrimaryKey() val idSala: Int,
    @ColumnInfo(name = "nomeSala") val nomeSala: String,
    @ColumnInfo(name = "descricao") val descricao: String,
    @ColumnInfo(name = "localizacao") val localizacao: String,
    @ColumnInfo(name = "lotacao") val lotacao: Int )

interface SalasDAO{
    @GET("v1/sala")
    fun getSalas() : Call<List<sala>>

    @GET("v1/sala/{id}")
    fun getSalasByID(@Path("id") id: Int) : Call<List<sala>>
}
```

Figura 42 - Código Classe Dados Sala

Este código define uma classe de dados chamada sala que contém informações como idSala, nomeSala, descricao, localizacao e lotacao. Esta classe é anotada com @Entity, indicando que será usada para criar uma tabela no banco de dados e o nome da tabela é especificado como "sala". A chave primária para a tabela é idSala.

A interface SalasDAO contém dois métodos, um para buscar todas as salas(getSalas) e outro para buscar salas por id(getSalasByID). Ambos os métodos são anotados com @GET e tem url específica("v1/sala" e "v1/sala/{id}") respectivamente. Eles retornam chamadas do tipo List<sala>.

```
class LoginActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)
        actionBar?.hide()
        supportActionBar?.hide()

        val button1 = findViewById<Button>(R.id.buttonLogin)
        val email = findViewById<EditText>(R.id.Email_login)
        val password = findViewById<EditText>(R.id.Password_Login)

        button1.setOnClickListener {
            val intent = Intent(this, ClassroomFinderActivity::class.java)
            checkLogin(email.text.toString(), password.text.toString(), intent)
        }
    }

    fun checkLogin(email: String, password: String, intent: Intent){
        val BASE_URL = "https://ipca-classroom-finder.onrender.com/"

        val db = Room.databaseBuilder(
            applicationContext,
            MyDatabase::class.java, "utilizador.db"
        ).build()

        var retrofit = Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        val apiService = retrofit.create(LoginResponseDao::class.java)
        val call = apiService.checkLogin(LoginRequest(email, password))

        call.enqueue(object : Callback<LoginResponse> {
            override fun onResponse(call: Call<LoginResponse>, response: Response<LoginResponse>) {
                if (response.code() == 200) {
                    GlobalScope.launch {
                        var data = db.utilizadorDao().getAll()
                        if(data.isNotEmpty()) {
                            data.forEach {
                                db.utilizadorDao().delete(it)
                            }
                            db.utilizadorDao().insertAll(response.body()?.user!!)
                        } else {
                            db.utilizadorDao().insertAll(response.body()?.user!!)
                        }
                    }
                    startActivity(intent)
                } else {
                    Toast.makeText(this@LoginActivity, "Credencias Erradas", Toast.LENGTH_LONG).show()
                }
            }
            override fun onFailure(call: Call<LoginResponse>, t: Throwable) {
                Log.d("BOAS?", t.toString())
            }
        })
    }
}
```

Figura 43 - Código LoginActivity PT1

```
val apiService = retrofit.create(LoginResponseDao::class.java)
val call = apiService.checkLogin(LoginRequest(email, password))

call.enqueue(object : Callback<LoginResponse> {
    override fun onResponse(call: Call<LoginResponse>, response: Response<LoginResponse>) {
        if (response.code() == 200) {
            GlobalScope.launch {
                var data = db.utilizadorDao().getAll()
                if(data.isNotEmpty()) {
                    data.forEach {
                        db.utilizadorDao().delete(it)
                    }
                    db.utilizadorDao().insertAll(response.body()?.user!!)
                } else {
                    db.utilizadorDao().insertAll(response.body()?.user!!)
                }
            }
            startActivity(intent)
        } else {
            Toast.makeText(this@LoginActivity, "Credencias Erradas", Toast.LENGTH_LONG).show()
        }
    }
    override fun onFailure(call: Call<LoginResponse>, t: Throwable) {
        Log.d("BOAS?", t.toString())
    }
})
}
```

Figura 44 - Código LoginActivity PT2

Este código define uma classe chamada LoginActivity que é uma atividade da AppCompatActivity. Ele possui um método onCreate que é chamado quando a atividade é criada. Também é definido um botão de login, os campos de email e password. Quando o botão é clicado, é chamado o método checkLogin que passa os valores dos campos email e password e uma intent para a próxima atividade.

O método checkLogin cria uma conexão com a base de dados e verifica as credenciais passadas, se as credenciais estiverem corretas, ele insere o usuário na base de dados e inicia a próxima atividade, caso contrário, exibe uma mensagem de erro.

```
class ClassroomFinderActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_classroom_finder)
        actionBar?.hide()
        supportActionBar?.hide()

        val button1 = findViewById<ImageButton>(R.id.my_button)
        val salas = findViewById<Button>(R.id.buttonGo)
        val horario = findViewById<Button>(R.id.buttonGo2)
        val nomeSala = findViewById<EditText>(R.id.nomeSalaFinder)

        button1.setOnClickListener{
            val intent = Intent(this, PerfilActivity::class.java)
            startActivity(intent)
        }
        salas.setOnClickListener{
            checkSala(nomeSala.text.toString())
        }
        horario.setOnClickListener{
            val intent = Intent(this, ScheduleActivity::class.java)
            startActivity(intent)
        }
    }

    fun checkSala(nomeSala : String)
    {
        val BASE_URL = "https://ipca-classroom-finder.onrender.com/"

        var retrofit = Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        val db = Room.databaseBuilder(
            applicationContext,
            MyDatabaseSala::class.java, "sala.db"
        ).build()
    }
}
```

Figura 45 - Código ClassroomFinderActivity PT 1

```

val api = retrofit.create(SalasDAO::class.java)
val call = api.getSalas()

call.enqueue(object : Callback<List<sala>> {
    override fun onFailure(call: Call<List<sala>>, t: Throwable) {
        Log.d("sala", t.toString())
    }
    override fun onResponse(call: Call<List<sala>>, response: Response<List<sala>>) {
        if(response.code() == 200)
        {
            val body = response.body() as List<sala>
            if(body?.isNotEmpty() == true)
            {
                GlobalScope.launch {
                    var data = db.SalaDao().getAll()
                    if(data.isNotEmpty())
                    {
                        data.forEach()
                        {
                            db.SalaDao().delete(it)
                        }
                    }
                    body.forEach()
                    {
                        if(it.nomeSala == nomeSala.toUpperCase())
                        {
                            val intent = Intent(this@ClassroomFinderActivity,
                                RoutingActivity::class.java)
                            intent.putExtra("nome", it.nomeSala)
                            intent.putExtra("descricao", it.descricao)
                            intent.putExtra("localizacao", it.localizacao)
                            intent.putExtra("lotacao", it.lotacao.toString())
                            startActivity(intent)
                        }
                        db.SalaDao().insertAll(it)
                    }
                }
            }
        }
    }
})

```

Figura 46 - Código ClassroomFinderActivity PT 2

Este código define uma classe chamada ClassroomFinderActivity que é uma atividade da AppCompatActivity. Ele possui um método onCreate que é chamado quando a atividade é criada. São definidos dois botões, um para ir para a atividade de horários e outro para buscar salas pelo nome. Também é definido um botão para ir para a atividade de perfil. Quando o botão de buscar salas é clicado, é chamado o método checkSala que passa o nome da sala para ser buscada.

O método checkSala cria uma conexão com a base de dados e busca as salas pelo nome passado, se encontrar, insere as salas na base de dados e inicia a próxima atividade, caso contrário, exibe uma mensagem de erro.


```
class RoutingActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_routing)  
        actionBar?.hide()  
        supportActionBar?.hide()  
  
        val intent = intent  
        val nome = intent.getStringExtra("nome").toString()  
        val descricao = intent.getStringExtra("descricao").toString()  
        val localizacao = intent.getStringExtra("localizacao").toString()  
        val lotacao = intent.getStringExtra("lotacao").toString()  
  
        findViewById<TextView>(R.id.nome_Sala).text = nome  
        findViewById<TextView>(R.id.Descricao_sala).text = descricao  
        findViewById<TextView>(R.id.localizacao_Sala).text = localizacao  
        findViewById<TextView>(R.id.lotacao_Sala).text = lotacao  
    }  
}
```

Figura 47 – Códig Classe RoutingActivity

Este código define uma classe chamada RoutingActivity que é uma atividade da AppCompatActivity. Ele possui um método onCreate que é chamado quando a atividade é criada. São definidos os dados da sala, como nome, descrição, localização e lotação, para serem exibidos na tela.

```
class PerfilActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_perfil)
        actionBar?.hide()
        supportActionBar?.hide()

        val nome = findViewById<TextView>(R.id.Username_perfil)
        val email = findViewById<TextView>(R.id.Email_perfil)

        val db = Room.databaseBuilder(
            applicationContext,
            MyDatabase::class.java, "utilizador.db"
        ).build()

        GlobalScope.launch {
            var data = db.utilizadorDao().getAll()

            if(data.isNotEmpty())
            {
                data.forEach()
                {
                    email.text = it.email
                    nome.text = it.nome + " " + it.sobrenome
                }
            }
        }
        val button1 = findViewById<Button>(R.id.buttonChangePassword)
        val button2 = findViewById<Button>(R.id.buttonHome)

        button1.setOnClickListener{
            val intent = Intent(this, ResetPasswordActivity::class.java)
            startActivity(intent)
        }
        button2.setOnClickListener{
            val intent = Intent(this, ClassroomFinderActivity::class.java)
            startActivity(intent)
        }
    }
}
```

Figura 48 - Código PerfilActivity

Este código define uma classe chamada PerfilActivity, que é uma atividade da AppCompatActivity. Ele possui um método onCreate que é chamado quando a atividade é criada. É definido os dados do utilizador como nome e email. Também existem dois botões, um para mudar a senha e outro para voltar à tela inicial. Quando esses botões são clicados, são iniciadas novas atividades.

O XML do frontend da nossa aplicação é igual ao das Mockups, tendo exportado os mesmos do *Figma*. Um exemplo de código é o seguinte:

```
<TextView
    android:id="@+id/textView2"
    android:layout_width="178dp"
    android:layout_height="42dp"
    android:layout_marginTop="20dp"
    android:text="Login"
    android:textAlignment="center"
    android:textColor="#FFFFFF"
    android:textSize="30dp"
    app:layout_constraintEnd_toEndOf="@+id/Logo"
    app:layout_constraintStart_toStartOf="@+id/Logo"
    app:layout_constraintTop_toBottomOf="@+id/Logo" />

<Button
    android:id="@+id/buttonRegister"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:backgroundTint="#081525"
    android:text="Forget Password"
    android:textSize="10dp"
    app:layout_constraintStart_toStartOf="@+id/PasswordImg"
    app:layout_constraintTop_toBottomOf="@+id/Password_Login" />

<Button
    android:id="@+id/buttonLogin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="70dp"
    android:backgroundTint="#010514"
    android:text="Login"
    app:layout_constraintEnd_toEndOf="@+id/Password_Login"
    app:layout_constraintTop_toBottomOf="@+id/Password_Login" />

<EditText
    android:id="@+id/Email_login"
    android:layout_width="276dp"
    android:layout_height="47dp"
    android:layout_marginTop="80dp"
    android:layout_marginLeft="30dp"
    android:ems="10"
    android:hint="Email"
    android:inputType="textPersonName"
    android:textColorHint="#DDAAAAAA"
    android:textColor="@color/white"
    app:layout_constraintEnd_toEndOf="@+id/textView2"
    app:layout_constraintStart_toStartOf="@+id/textView2"
    app:layout_constraintTop_toBottomOf="@+id/textView2" />
```

Figura 49 - Login XML PT 1

```
<EditText
    android:id="@+id/Password_Login"
    android:layout_width="276dp"
    android:layout_height="47dp"
    android:layout_marginTop="30dp"
    android:ems="10"
    android:hint="Password"
    android:inputType="textPassword"
    android:textColorHint="#DDAAAAAA"
    android:textColor="@color/white"
    app:layout_constraintEnd_toEndOf="@+id/Email_login"
    app:layout_constraintStart_toStartOf="@+id/Email_login"
    app:layout_constraintTop_toBottomOf="@+id/Email_login" />

<ImageView
    android:id="@+id/PasswordImg"
    android:layout_width="60dp"
    android:layout_height="48dp"
    android:layout_marginTop="30dp"
    android:src="@drawable/password"
    app:layout_constraintEnd_toStartOf="@+id/Email_login"
    app:layout_constraintStart_toStartOf="@+id/UserImg"
    app:layout_constraintTop_toBottomOf="@+id/UserImg" />

<ImageView
    android:id="@+id/UserImg"
    android:layout_width="60dp"
    android:layout_height="48dp"
    android:layout_marginTop="80dp"
    android:src="@drawable/user"
    app:layout_constraintEnd_toStartOf="@+id/Email_login"
    app:layout_constraintTop_toBottomOf="@+id/textView2" />

<ImageView
    android:id="@+id/Logo"
    android:layout_width="148dp"
    android:layout_height="148dp"
    android:src="@drawable/you_never_walk_alone_"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Figura 50 - Login XML PT 2

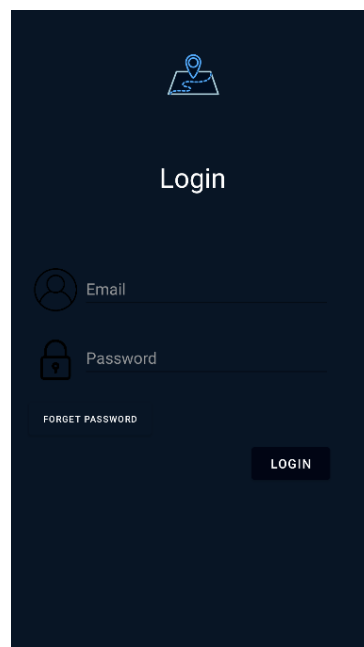


Figura 51 - Activity Login

10. Conclusão

Ao longo do desenvolvimento deste trabalho, foi possível consolidar conceitos sobre integração de sistemas de informação, projetar arquiteturas de integração de sistemas, explorar ferramentas de suporte ao desenvolvimento de serviços web, investigar novas tecnologias e frameworks para implementação de serviços web, potenciar a experiência no desenvolvimento de aplicações, explorar funcionalidades de programação em android e aprofundar os conhecimentos adquiridos na unidade curricular. Todo o grupo achou este projeto e esta cadeira uma mais-valia na medida em que englobou diversas unidades curriculares em simultâneo e em concordância para que fosse possível chegar à implementação final.

Como grupo tivemos as nossas dificuldades e momentos mais frágeis uma vez que algumas das unidades curriculares e os seus objetivos abordados eram novidade. Foi necessário bastante entreajuda e pesquisa aprofundada para a conclusão deste trabalho prático.

Em suma pensamos que como programadores e como pessoas obtivemos valores bastantes positivos para o mundo de trabalho e desenvolvimento pessoal.

11. Bibliografia:

Reportório Aulas: <https://elearning2.ipca.pt/>

GitHub Professor Lufer: <https://github.com/luferIPCA/ISI/tree/master/ESI>

Microsoft Azure: <https://azure.microsoft.com/pt-pt/>

SQL Workbench: <https://www.mysql.com/products/workbench/>

Postman: <https://www.postman.com/>

NodeJS: <https://nodejs.org/en/>

Express: <https://www.expresstecnologia.com/>