

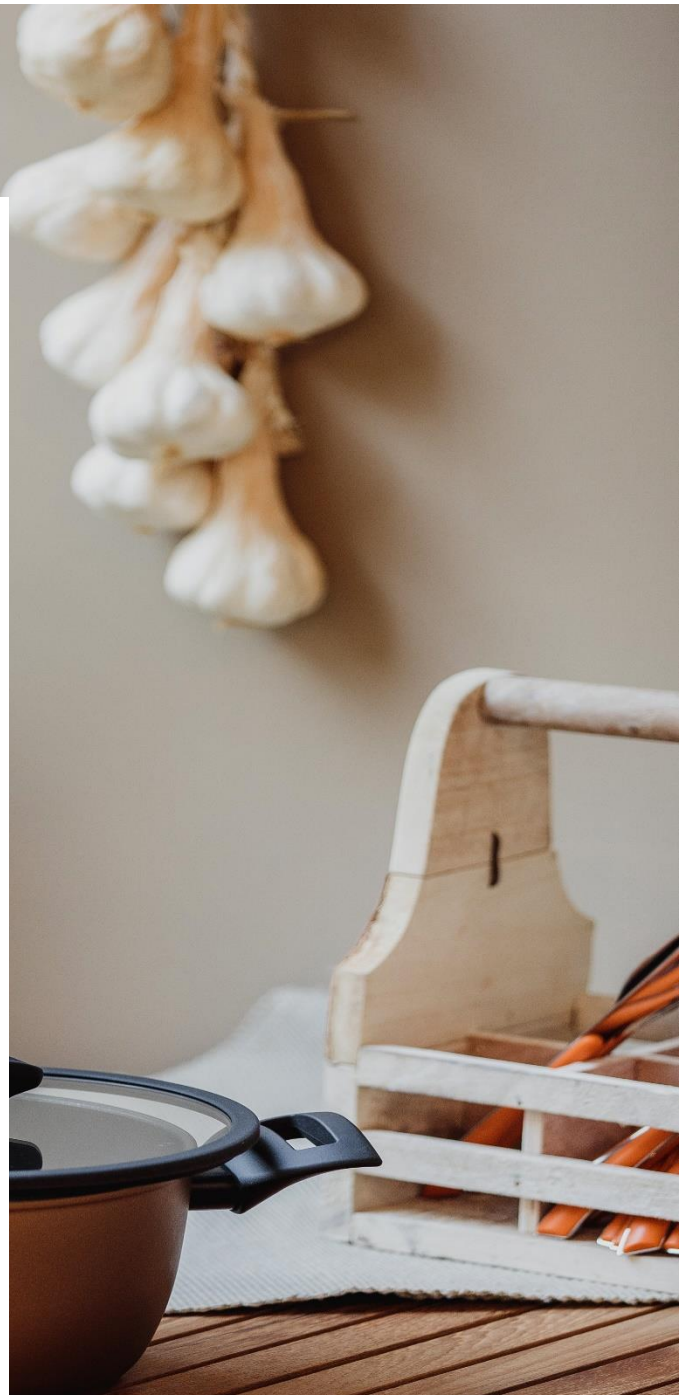
# CookBook

---

**2020**

---

**Instituto Superior Miguel Torga  
Programação III**





**Beatriz Pereira nº 11313**

**Tiago Baptista nº 11385**

---

## Introdução

Para este projeto optámos por usar o projeto anterior e convertê-lo em uma aplicação android, para isso, tivemos que fazer algumas alterações na base de dados de modo a esta coincidir com a lógica da nossa API.

O tema do projeto trata-se de uma plataforma para inserção de receitas de culinária. Escolhemos este tema porque havia em ambos um leve interesse de como seria se houvesse uma plataforma que nos ajudasse a aprender a cozinhar, ou que desse a oportunidade de conhecer receitas novas para experimentar.

Ficou decidido atribuir vários tons de amarelo torrado à plataforma, devido à sua tipologia. Amarelo representa a alegria, positividade, simpatia e otimismo, mas o mais importante é que esta é a cor da fome. É usada para estimular o apetite.

Em relação à tipografia, no logótipo do site foi usada a tipografia “Gabriola” tendo sido modificada em alguns aspetos para uma melhor apresentação de imagem. No entanto, o tipo de letra usada no interior da plataforma é a Helvética para facilitar uma melhor visibilidade ao utilizador.

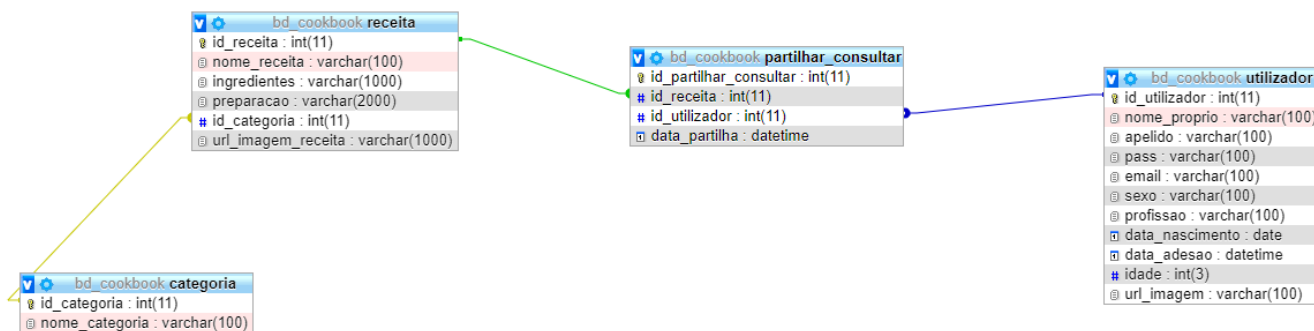
## Base de Dados

Anteriormente, a base de dados foi criada com o nome de bd\_cookbook.

Dentro da mesma foram feitas quatro tabelas:

- Utilizador;
- partilha\_consulta;
- receita;
- categoria.

A partilha\_consulta é a relação entre o utilizador e a receita, sendo que esta está ligada à tabela categoria. Para este projeto tivemos que fazer algumas alterações na base de dados de modo a esta coincidir com a lógica da nossa API. Como por exemplo, acrescentámos um id\_partilhar\_consultar e o id\_receita e id\_utilizador tornaram-se chaves forasteiras.



## Definição mapa de navegação

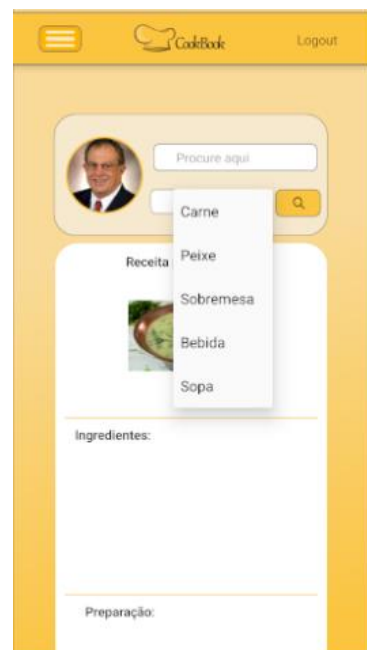
Esta plataforma inicia-se na MainActivity. A partir da mesma, podemos aceder à conta através do botão “Entrar”, ou então, ir para a página de registo ao clicar na frase em baixo do botão.



*Figura MainActivity*



*Figura RegistrarActivity*



*Figura HomeUserActivity*

*Ao carregar no botão “Entrar” seremos enviados para a HomeUserActivity.*

*Nesta temos a opção de procura de receitas através da escolha de categorias. Para além disto, é nos dado dois botões, um para sair da conta e outro para aceder ao perfil.*

No perfil (UserProfileActivity), é nos dado a informação do utilizador e três botões para além do botão de sair.

Estes três botões levam-nos para as seguintes páginas:

- Editar Informações -> EditarIfomacoesActivity;
- Minhas Receitas -> ReceitasUser;
- Adicionar Receita -> ReceitaAddActivity.



Figura UserProfileActivity



Figura EditarIfomacoesActivity



Figura ReceitasUser

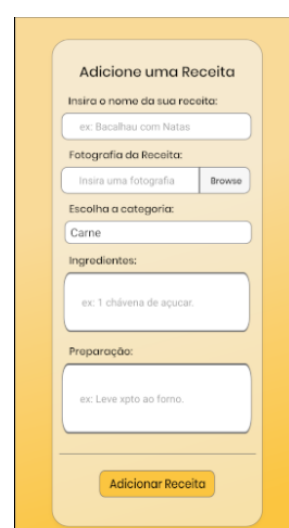


Figura ReceitaAddActivity

Na página de editar informação (EditarIfomacoesActivity), podemos digitar novos dados sobre o utilizador e carregar no botão “Editar” para alteração dos mesmos.

Nas minhas receitas (ReceitasUser), é nos mostrado as receitas do utilizador e ainda nos dá a oportunidade de procurar pela receita e escolher a categoria.

No adicionar receita (ReceitaAddActivity) é dado um formulário para criação de uma nova receita.

---

## Etapas

Este projeto foi realizado através de quatro etapas.

- 1ª Etapa – Apresentação do Tema e Constituição do Grupo

Como referido em cima, o tema do projeto trata-se de uma plataforma para inserção de receitas de culinária. Escolhemos este tema porque havia em ambos um leve interesse de como seria se houvesse uma plataforma que nos ajudasse a aprender a cozinhar, ou que desse a oportunidade de conhecer receitas novas para experimentar.

- 2ª Etapa – Especificação da API.

A documentação da API foi realizada a partir do SwaggerEditor.

Nesta etapa estabelecemos de um modo geral como vai funcionar a API da nossa aplicação. Estabelecemos os endpoints (PUT, GET, POST, DELETE) para utilizadores, receitas e categorias.

- 3ª Etapa – Protótipo funcional da App Android.

Nesta etapa, fizemos o design da nossa aplicação no AndroidStudio e as respetivas navegações entre activitys como referido no tópico a cima.

- 4ª Etapa – Projeto Final

Implementação de código no Node.js e design final da aplicação e implementação de código no AndroidStudio.

# Node.js

Ficheiro app.js, este é o ficheiro principal da api.

A partir deste importa-se as dependências, a middleware, as configurações do servidor e sua respetiva porta e o caminho para as routes da API, que neste caso optámos por utilizar routes separadas, a utilizadorApi e a receitaApi.

```
src > app.js > ...
1 const express = require("express");
2 const cors = require("cors");
3 const bodyParser = require("express");
4
5 const app = express();
6
7 //Configuração da porta do servidor
8 app.set("port", process.env.port || 3001);
9
10 //middleware
11 app.use(express.json());
12
13 //CORS
14 app.use(cors());
15
16 //parser
17 app.use(bodyParser.urlencoded({ extended: true }));
18
19 //ruta API (user_login_register)
20 const utilizadorApi = require("../routes/utilizador.route");
21 app.use("/api/v1", utilizadorApi);
22 //ruta API (receita)
23 const receitaApi = require("../routes/receita.route");
24 app.use("/api/v1", receitaApi);
25
26 //servidor (definição da porta)
27 app.listen(app.get("port"), () => {
28   console.log("servidor iniciado na porta: " + app.get("port"));
29 });
```

A route utilizador permite ao utilizador efetuar o login, consultar as suas informações, fazer update nos seus dados e apagar os seus dados. Estabelece-se a ligação aos controllers invocando o ficheiro e as respetivas funções tendo em conta os seus métodos: get, post, put, delete.

```
src > routes > JS utilizador.routes.js > ...
1 const express = require('express');
2 const router = express.Router();
3
4 //importar os controladores do user
5 const utilizadorController = require('../controllers/utilizador.controller');
6
7 //endpoints correspondente à parte do user da API
8 /* test api connection */ router.get('/utilizadores', utilizadorController.utilizador_list);
9 router.get('/utilizador/:id', utilizadorController.utilizador_details);
10 //router.get('/utilizador/:id', utilizadorController.utilizador_login);
11 router.post('/utilizador', utilizadorController.utilizador_create);
12 router.put('/utilizador/:id', utilizadorController.utilizador_update);
13 //router.get('/utilizador/:id', utilizadorController.utilizador_logout);
14 router.get('/login', utilizadorController.utilizador_login);
15
16
17 module.exports = router;
```



---

A route receita, funciona da mesma maneira que a route utilizador, no entanto esta destina-se a funcionalidades relacionadas com as receitas.

Por exemplo, consultar todas as receitas ou só uma receita em específico e fazer atualização ou apagar uma receita.

```
src > routes > # receita.routes.js > ...
1  const express = require('express');
2  const router = express.Router();
3
4  //importar os controladores das receitas
5  const receitaController = require('../controllers/receita.controller');
6
7  //endpoints correspondente à parte da receita da API
8  router.get('/receitas', receitaController.receita_list);
9  //router.get('/:id', receitaController.receita_details);
10 router.post('/receita', receitaController.receita_create);
11 router.put('/receita/:id', receitaController.receita_update);
12 router.delete('/receita/:id', receitaController.receita_delete);
13
14 module.exports = router;
```

Os controllers é a parte da Api onde se encontra as funções de cada route. No caso do receita.controller podemos encontrar as seguintes funções:

- receita\_list;
- receita\_create;
- receita\_update;
- receita\_delete.

No caso utilizador.controller temos as funções:

- utilizador\_list;
- utilizador\_details;
- utilizador\_create;
- utilizador\_update;
- utilizador\_login.

Cada ficheiro controller contém uma ligação ao respetivo model.

Os models são as partes da API que gere a informação relacionada com a base de dados.

Neste ficheiro utiliza-se as funcionalidades da dependência Sequelize para gerir a ligação à base de dados e importa-se a funcionalidade das configs que por sua vez contém a ligação à base de dados situada no phpmyAdmin.

Podendo assim, as informações de cada tabela.

Cada ficheiro model contém as informações da respetiva tabela da base de dados, declarando o tipo de atributo e a chave primária da tabela.

```
src > models > JS utilizador.models > ...
1  const sequelize = require('sequelize');
2  //const Receita = require('../receita.model');
3  const db = require('../config/database');
4
5  var Utilizador = db.define('utilizador', {
6    id_utilizador: {
7      type: sequelize.INTEGER, primaryKey: true,
8      autoIncrement: true
9    },
10
11    nome proprio: sequelize.STRING,
12    apelido: sequelize.STRING,
13    idade: sequelize.INTEGER,
14    pass: sequelize.STRING,
15    email: sequelize.STRING,
16    sexo: sequelize.STRING,
17    profissao: sequelize.STRING,
18    data_nascimento: sequelize.DATEONLY,
19    data_adesao: sequelize.DATE,
20    url_imagem: sequelize.STRING,
21  }, {
22    timestamps: false,
23    tableName: 'utilizador'
24  });
25  //Utilizador.hasOne(Receita, {foreignKey: 'id'});
26
27  module.exports = Utilizador;
```

```
src > models > JS receita.model.js > ...
1  const sequelize = require('sequelize');
2  //const Categoria = require('../categoria.model');
3  const db = require('../config/database');
4
5  var Receita = db.define('receita', {
6    id_receita: {
7      type: sequelize.INTEGER, primaryKey: true,
8      autoIncrement: true
9    },
10    nome_receita: sequelize.STRING,
11    ingredientes: sequelize.STRING,
12    preparacao: sequelize.STRING,
13    url_imagem_receita: sequelize.STRING,
14    id_categoria: sequelize.INTEGER,
15  }, {
16    timestamps: false,
17    tableName: 'receita'
18  });
19  //Receita.hasOne(Categoria, {foreignKey: 'id_categoria'});
20  module.exports = Receita;
```

```
src > models > JS partilhar_consultar.models.js > ...
1  const sequelize = require('sequelize');
2  const Utilizador = require('../utilizador.model');
3  const Receita = require('../receita.model');
4  const db = require('../../config/database');
5
6  /* modelo correspondente à relação entre o utilizador e a receita...
7  relação de n-a, um utilizador pode partilhar várias receitas e uma receita
8  pode ser partilhada por vários utilizadores */
9
10 var Partilhar_Consultar = db.define('partilhar_consultar', {
11   id_partilhar_consultar: {
12     type: sequelize.INTEGER, primaryKey: true,
13     autoIncrement: true
14   }
15 }, {
16   timestamps: true,
17   tableName: 'aluno_disciplina'
18 });
19 Utilizador.belongsToMany(Receita, {through: 'partilhar_consultar'});
20 Receita.belongsToMany(Utilizador, {through: 'partilhar_consultar'});
21
22 module.exports = Partilhar_Consultar;
```

```
src > models > JS categoria.models.js > ...
1  const sequelize = require('sequelize');
2  //const Receita = require('../receita.model');
3  const db = require('../../config/database');
4
5  var Categoria = db.define('disciplina', {
6   id_categoria: {
7     type: sequelize.INTEGER, primaryKey: true,
8     autoIncrement: true
9   },
10   nome_categoria: sequelize.STRING,
11 }, {
12   timestamps: true,
13   tableName: 'Categoria'
14 });
15 //Receita.belongsToMany(Categoria, {through: 'receita'});
16 //Categoria.belongsToMany(Receita, {through: 'receita'});
17 module.exports = Categoria;
```

Na pasta configs situa-se um ficheiro chamado database.js, que a partir da dependência sequelize estabelece a ligação à base de dados situada no phpMyAdmin.

```
src > config > JS database.js > ...
1  //Estabelecer a ligação à base de dados
2  const sequelize = require('sequelize');
3  const ligacao = new sequelize('bd_cookbook', 'root', '', {
4    host: 'localhost',
5    dialect: 'mysql'
6  });
7  module.exports = ligacao;
```

## Android Studio – Código importante

- Botões

Este código permite-nos que ao clicar no “userbtn” o utilizador seja redirecionado para a página UserProfileActivity. O mesmo acontece com os outros botões mas com “ids” e destinos diferentes.

```
public void BtnUserInfo (View v) {  
    Intent userbtn = new Intent( packageContext: this, UserProfileActivity.class);  
    startActivity(userbtn);  
}
```

- Spinner

Spinner permite-nos fazer um dropdown, isto é, ao clicarmos nele irá aparecer uma lista de várias categorias. Esta lista de categorias foram criadas num ficheiro chamado “arrays.xml” inserida na pasta “values”.

```
<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string-array name="spinnerCateg">  
        <item>Carne</item>  
        <item>Peixe</item>  
        <item>Sobremesa</item>  
        <item>Bebida</item>  
        <item>Sopa</item>  
    </string-array>  
</resources>
```

Figura arrays.xml

```
public class HomeUserActivity extends AppCompatActivity {  
    private Spinner spinnerCategoria;
```

```
//Drop down  
spinnerCategoria = findViewById(R.id.SpinnerCategoria);  
  
String[] SpCateg = getResources().getStringArray(R.array.spinnerCateg);  
ArrayAdapter adapter = new ArrayAdapter( context: this, android.R.layout.simple_spinner_item, SpCateg);  
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);  
spinnerCategoria.setAdapter(adapter);
```

Figura código java “Spinner”

- Calendário

Os int year, month, day é o que vai permitir ir buscar o ano, mês e dia para que a partir do DatePickerDialog seja nos mostrado a data atual.

“Theme\_DeviceDefault\_Light\_Dialog\_MinWidth” foi o estilo de calendário escolhido.

Ao escolhermos a data, a mesma será mostrada no ImageView da data de nascimento.

```
Date = (TextView)findViewById(R.id.date);
Date.setOnClickListener((view) -> {
    Calendar calendar = Calendar.getInstance();
    //obtem a data atual e define o seletor de datas pop-up

    int year = calendar.get(Calendar.YEAR);
    int month = calendar.get(Calendar.MONTH);
    int day = calendar.get(Calendar.DAY_OF_MONTH);

    DatePickerDialog dialog = new DatePickerDialog(
        context: EditarInformacoesActivity.this
        ,android.R.style.Theme_DeviceDefault_Light_Dialog_MinWidth
        ,dateSetListener,year,month,day);

    dialog.getWindow().setBackgroundDrawable(new ColorDrawable(Color.WHITE));
    dialog.show();
});

//Botão para mostrar o calendário

dateSetListener = (OnDateSetListener) (datePicker, year, month, day) -> {
    month = month+1;
    String date = day + "/" + month + "/" + year;
    Date.setText(date);
};
```

---

## Não funcional

No Android Studio, temos uma activity chamada ApiConnect que contém algumas linhas de código para estabelecer a ligação entre a API e o Android Studio, no entanto, essas não estão a funcionar, ou seja, a ligação entre eles está inacabada.

## Links Bibliográficos

- [https://www.youtube.com/watch?v=E1LSY3g-CtY&fbclid=IwAR2kgYw8gnTCluMxvR-6WKtimmwfw2eb8LrAGvNssT\\_H8zcvwBpnxvt5tjQ](https://www.youtube.com/watch?v=E1LSY3g-CtY&fbclid=IwAR2kgYw8gnTCluMxvR-6WKtimmwfw2eb8LrAGvNssT_H8zcvwBpnxvt5tjQ)
- <https://www.youtube.com/watch?v=salqHKolTHc>
- <https://www.youtube.com/watch?v=X7Xz5ixKVhs>