



### 1. Descrição geral

A avaliação da cadeira de aplicações distribuídas está dividida em quatro projetos. O projeto 3 não tem ligação com os anteriores.

O objetivo geral do projeto será concretizar um serviço WEB para gerir um sistema simplificado de classificação de albuns de música de utilizadores. A implementação vai utilizar o estilo arquitetural REST [1] e uma base de dados relacional acessível pela linguagem SQL. Para este efeito no servidor será utilizada a *framework* de desenvolvimento WEB *Flask* [2] e o motor de base de dados SQL *sqlite* [3]. O programa cliente utilizará o módulo *requests* [4] para implementar a interação cliente/servidor baseada no HTTP.

### 2. Esquema da base de dados

A definição da base de dados assenta nos conceitos envolvidos: utilizador, album, banda e lista de albuns (lista\_albuns). Cada lista de albuns criada por um utilizador contem um ou mais albuns e cada album está associado a uma banda (podendo vários albuns estarem associados à mesma banda). Cada conceito corresponde a uma tabela de acordo com a figura seguinte, onde também se ilustram as várias relações.

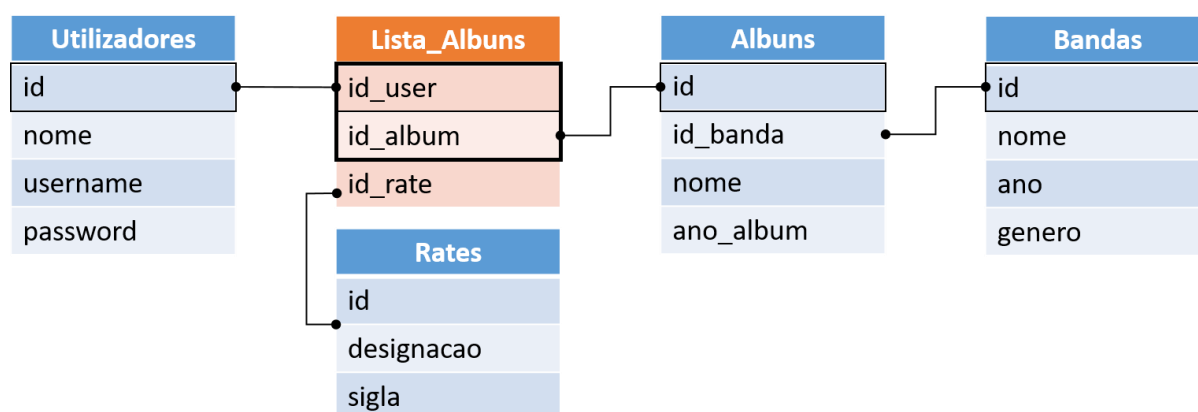


Figura 1 - Esquema da base de dados.

```
PRAGMA foreign_keys = ON;

CREATE TABLE utilizadores (
    id            INTEGER PRIMARY KEY,
    nome          TEXT,
    username      TEXT,
    password      TEXT
);

CREATE TABLE albuns (
    id            INTEGER PRIMARY KEY,
    id_banda      INTEGER,
    nome          TEXT,
    ano_album     INTEGER,
    FOREIGN KEY(id_banda) REFERENCES bandas(id)
);
```

```

CREATE TABLE bandas (
    id            INTEGER PRIMARY KEY,
    nome          TEXT,
    ano           INTEGER,
    genero        TEXT
);

CREATE TABLE rates (
    id            INTEGER PRIMARY KEY,
    designacao    TEXT,
    sigla         TEXT
);

CREATE TABLE listas_albums (
    id_user       INTEGER,
    id_album      INTEGER,
    id_rate       INTEGER,
    PRIMARY KEY (id_user, id_album),
    FOREIGN KEY(id_user) REFERENCES utilizadores(id),
    FOREIGN KEY(id_album) REFERENCES albums(id)
    FOREIGN KEY(id_rate) REFERENCES rates(id)
);

```

Para criar a base de dados poderá ser utilizado o código SQL apresentado acima.

A primeira linha desta listagem serve para que o *sqlite* possa suportar chaves estrangeiras. Por omissão, na instalação de alguns sistemas operativos, essa opção está desabilitada.

Os dados inseridos na tabela *rates* são os que se seguem e poderão ser inseridos pelo seguinte código SQL.

```

INSERT INTO rates (id, designacao, sigla) VALUES
    (1, "Medíocre", "M"),
    (2, "Mau", "m"),
    (3, "Suficiente", "S"),
    (4, "Bom", "B"),
    (5, "Muito bom", "MB")
;

```

A aplicação pretendida deverá contemplar uma rotina de inicialização que verifica se a base de dados já existe. Caso não exista ela deverá ser criada e inicializada com o código apresentado (criação das tabelas e inserção dos registos na tabela *rates*).

### 3. O programa cliente

O programa cliente aceita interactivamente três operações e seus parâmetros (introduzidos via teclado numa consola), e comunica com o servidor para que este processe as operações e armazene a informação numa base de dados. A tabela 1 mostra detalhadamente as operações que o cliente deverá suportar.

Tabela 1 - Lista de operações que o cliente aceita e parâmetros correspondentes.

Operação	Parâmetros	Observações
<b>ADD</b>	USER <username> <password> <nome> ou BANDA <genero> <nome> <ano> ou ALBUM <id_banda> <nome> <ano album> ou <id_user> <id_album> <rate>	genero = pop rock indy  metal trance classic  A última variante serve para o utilizador classificar (rate) um album rate = M m S B MB
<b>REMOVE</b> <i>ou</i> <b>SHOW</b>	USER <id_user> ou BANDA <id_banda> ou ALBUM <id_album> ou ALL <USERS   BANDAS   ALBUNS> ou ALL ALBUNS_B <id_banda> ou ALL ALBUNS_U <id_user> ou ALL ALBUNS <rate>	As quatro últimas variantes permitem obter: <ul style="list-style-type: none"> <li>• todos os utilizadores, ou albuns ou bandas</li> <li>• todos os albuns de uma banda;</li> <li>• todos os albuns classificados por um aluno;</li> <li>• todas os albuns classificados por uma dada rate (M, m, S, B, MB);</li> </ul>
<b>UPDATE</b>	ALBUM <id_album> <rate> <id_user> USER <id_user> <password>	

Convém lembrar que os id's (chaves primárias) dos elementos nas tabelas de utilizadores, albuns e bandas são números inteiros.

O cliente comunicará com o servidor através de mensagens em HTTP e usará uma representação utilizando JSON [5]. Para este efeito os alunos utilizarão o módulo *requests* [4]. As mensagens terão de respeitar a API REST definida pelo serviço WEB de criação de listas de albúns.

#### 4. O serviço WEB

O serviço WEB será implementado com recurso à *framework* Flask [2] e a API REST será disponibilizada através de três URLs de base:

1. /utilizadores  
Para operações relativas a utilizadores.
2. /bandas  
Para operações relativas a bandas.
3. /albuns  
Para operações relativas a albúns.

É muito importante que os alunos planeiem a API REST antes de iniciarem a implementação. Sugere-se que façam uma tabela onde definam a correspondência entre as operações suportadas, o método do HTTP, as URLs dos recursos, os parâmetros das operações, e as possíveis respostas HTTP com que o serviço responderá ao cliente.

Quando o serviço recebe uma mensagem de um cliente, a operação deverá ser implementada sobre a base de dados e a resposta será preparada segundo os padrões REST utilizando JSON para transmitir a representação dos recursos ou o conteúdo de outras mensagens.

Para o acesso à base de dados, os alunos devem procurar na documentação sobre Flask a forma de fazer com que a ligação à base de dados exista de forma automática sempre que a aplicação recebe um pedido.

## 5. Integração com serviço REST público da Internet

De forma a obter mais informação sobre os álbuns, autores e músicas, os alunos devem recorrer à informação pública disponível sobre os mesmos. No âmbito da disciplina, iremos recorrer à API REST disponível no *spotify* [6]. A informação disponível no *spotify* deve completar a informação disponível no servidor a construir, i.e., de cada vez que a informação relativa a um álbum, autor ou música for requerida ao servidor *flask*, esta deve ser complementada com a informação vinda do *spotify*. No entanto, a informação vinda do *spotify* não será guardada localmente, será apresentada ao utilizador, requerendo um novo pedido de cada vez que se queira listar a informação sobre os álbuns, autores e músicas.

**Nota: de forma a facilitar a gestão da informação sobre os álbuns e músicas, os alunos devem utilizar como referência os *ids* gerados/utilizados pelo *spotify*.**

## 6. Tratamento de erros

Sempre que a operação não possa ser executada ou que algum erro inesperado ocorra, o serviço deverá responder ao cliente com uma resposta HTTP incluindo uma descrição detalhada do problema segundo o formato apresentado nas aulas TP sobre JSON. O cliente apresentará a informação da descrição detalhada na consola.

## 7. A entrega

Atendendo à situação atual, decorrente da pandemia de SARS-CoV-2, e ao facto dos projetos 3 e 4 estarem relacionados, o projeto 3 terá entrega na mesma data do projeto 4. No entanto, para um adequado fluxo de desenvolvimento e gestão de tempo, os alunos deverão ter o projeto 3 concluído antes da data em que será publicado o enunciado do projeto 4, a 4 de maio de 2020.

## 8. Referências

- [1] [http://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)
- [2] <http://flask.pocoo.org/>
- [3] <https://www.sqlite.org/>
- [4] <http://docs.python-requests.org/en/master/>
- [5] <http://www.json.org/>
- [6] <https://developer.spotify.com/documentation/web-api/reference/>