Ciências ULisboa | Informática

Aplicações Distribuídas

2022/23

Projeto 3

1. Descrição geral

Nota Prévia: A avaliação da cadeira de aplicações distribuídas está divida em quatro projetos. O Projeto 3 não tem ligação com os dois anteriores.

O objetivo geral do presente projeto será concretizar um serviço Web – Escapadinha Solarenga – para procura de locais com bom tempo para fazer férias de curta duração, algures por uma capital da Europa. O cliente deverá ser capaz de informar o servidor de:

1. A capital de onde irá partir (cidade base).
2. Qual o preço máximo que está disposto a gastar na sua viagem de avião (ida-e-volta).

Há apenas 5 requisitos:

1. A previsão meteorológica no destino deverá ser Tempo Limpo / Sol durante +2 dias.
2. Apenas considerar viagens com voos diretos (entre cidade base e destino de férias).
3. A viagem dura exactamente 4 dias (exº partida a dia 1/Jan, regresso a dia 4/Jan).
4. O regresso acontecerá no máximo em 14 dias a partir do dia da pesquisa (pesquisando dia 1/Jan, o regresso será no máximo dia 14/Jan).
5. Apenas 1 adulto irá viajar, no plano *Economy* (sem extras).

Por uma questão de simplicidade, iremos reduzir as pesquisas a 10 capitais, nomeadamente: Lisboa, Madrid, Paris, Dublin, Bruxelas, Liubliana, Amsterdão, Berlim, Roma, Viena.

A implementação deverá utilizar o estilo arquitetural REST [1] e uma base de dados relacional acessível pela linguagem SQL. No servidor serão utilizados 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 em HTTP. A organização destes elementos está mostrada na Figura 1.

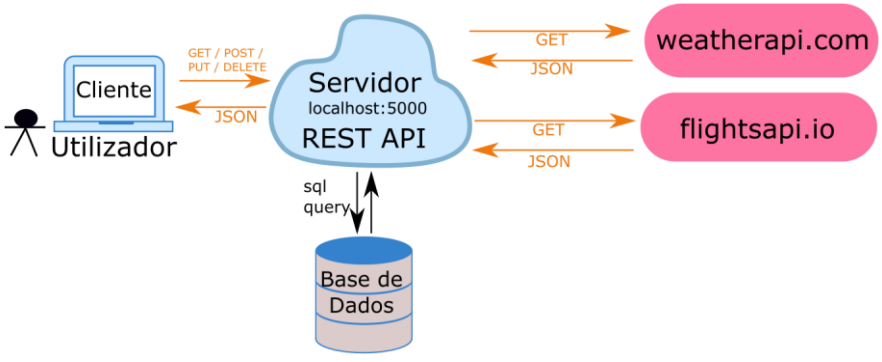

Figura 1 – Esquemático da arquitetura do projecto a desenvolver.

2. Definições

- **Viagem** (*round-trip*) – um conjunto de dois voos directos - ida e volta – entre dois aeroportos, com um custo associado assim como um identificador único
- **Voo** (*leg*) - uma travessia entre dois aeroportos, com aeroporto de origem e destino, com hora e data de partida e de chegada, com uma duração associada assim como uma (ou várias) companhia(s) aérea(s) que opera(m) o voo.
- **IATA** - código universal identificador de um aeroporto.

3. Esquema da base de dados

Deverá criar e eventualmente popular uma base de dados como a representada pelo diagrama da Figura 2.

A tabela **roundtrips** conterá as viagens encontradas, definidas pelo seu voo de ida e regresso (id_leg0 e id_leg1) assim como o seu custo. A Figura 3 dá um exemplo de três entradas nesta tabela. Cada um destes voos individuais está definido na tabela **legs**, particularmente pelos códigos de aeroportos do local de partida e chegada (IATA), pela data e hora de partida e chegada (DDMMYYYY HHMM), pela duração em minutos do voo, e por fim pelo código da companhia aérea (um, ou mais separados por espaço) que opera o voo. Um exemplo de algumas entradas nesta tabela está presente na Figura 4. Para suportar a resolução de códigos da operadora em nomes por extenso, deverá preencher a tabela **airlines.**

Haverá ainda uma tabela **weather** que possuirá as últimas previsões de tempo para vários locais e respectivos dias. Além da condição do tempo (Sol, Limpo, Chuva, etc.), a previsão temperatura mínima e máxima deverá estar também patente.

De forma a serem inequívocas as localizações que o Cliente, a tabela **weather** e a tabela **legs** utilizam, deverá ser criada uma tabela **locations**. Nesta, para cada uma das 10 capitais irá estar associado o nome que o cliente deve usar para se referir ao local, um código IATA e nome da localidade/estação meteorologia. Pode assumir que estes três campos possuem uma relação de um para um para um, ie, cada capital tem apenas um aeroporto e uma estação meteorológica associada.

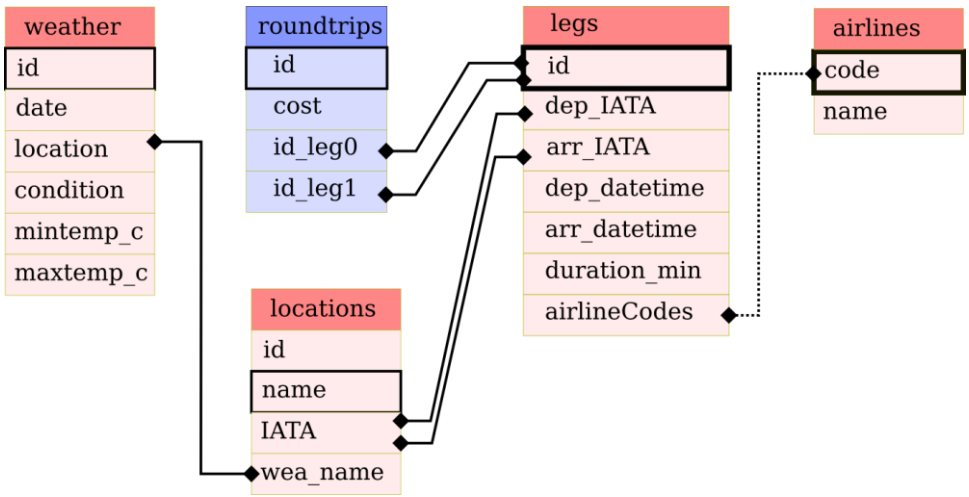

Figura 2 - Esquema da base de dados.

roundtrips			
id	**cost**	**leg0**	**leg1**
95fcd254b597	123	xyzx	abcdaabcdaabcda
asdc2342asda	345	xyzx	uytfvcz
cd2345aasdcd	39	abcdaabcdaabcda	uytfvcz

Figure 3 – Exemplo da Tabela **roundtrips** com 3 viagens. Cada viagem é composta por dois voos (**legs**).

legs						
id	**dep_IATA**	**arr_IATA**	**dep_datetime**	**arr_datetime**	**duration_min**	**airlineCodes**
xyzx	LIS	MAD	20230405 1201	20230405 1405	62	TP UA
qrtxsdfsfsdfsfa	LIS	BER	....	....	....	....
abcdaabcdaabcda	MAD	LIS	....	....	....	....

Figura 4 – Exemplo da tabela **legs** com três voos.

Para criar a base de dados, poderá ser utilizado o código SQL apresentado da Listagem 1. 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. Repare também na opção "`ON DELETE CASCADE`", a qual serve para indicar que a remoção de um elemento numa tabela pai, causará a remoção de todas linhas que apontavam para este elemento numa tabela filha.

```
PRAGMA foreign_keys = ON;



CREATE TABLE legs (
        id                              INTEGER PRIMARY KEY,
        dep_IATA                          TEXT,
        arr_IATA                           TEXT,
        dep_datetime                      TEXT,
        ...
        FOREIGN KEY(dep_IATA)  REFERENCES locations(IATA) ON DELETE CASCADE,
        FOREIGN KEY(arr_IATA)  REFERENCES locations(IATA) ON DELETE CASCADE
);

...
```

Listagem 1 SQL para criar as tabelas para o projeto.

A aplicação pretendida deverá contemplar uma rotina de inicialização que verifica se a base de dados já existe. Caso esta não exista, ela deverá ser criada e inicializada com a estrutura mostrada acima (i.e., a criação das tabelas, a inserção dos 10 registos na tabela **locations** e os que achar necessário na tabela **airlines**).

4. O programa cliente

O programa cliente aceita interactivamente 3 operações e os seus parâmetros (de forma semelhante à leitura do *stdin* como nos projetos anteriores), 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*
SEARCH	**<location> <cost>**	Dado um <cost> e <location>, retornar todas as viagens até <cost> baseados de <location>
FILTER	**DST <location> AIRLINE <code> SUN <n>** parâmetros facultativos 4 >= n >= 2	Dada <location>, <code> e <n> assim como uma lista de IDs de viagens, o servidor deve filtrar e retornar apenas as viagens para o destino <location>, na companhia <code> e com exactamente <n> dias de sol.
FILTER	**DIVERSIFY**	Dada uma lista de IDs de viagens, o servidor deve filtrar / retornar: A viagens mais barata para cada um dos destinos.
		Retornar viagens é o equivalente a retornar uma lista com: • o id das viagens, • o custo • a IATA origem e destino, • e a data dos 2 voos.
DETAILS	**<viagem_ID>**	Dado uma viagem <viagem_ID>, obter a previsão detalhada para os 4 dias de viagem + custo + todos os dados dos dois voos (ie, datas, horas, IATAs da origem destino, duração dos voos, e companhias aéreas)

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.

5. 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. `/search`
   Para desencadear a pesquisa da previsão de tempo e da **subsequente** pesquisa de voos em third-party APIs (na Internet). Deverá retornar viagens.
2. `/filter`
   Para operações relativas a filtrar resultados duma dada lista de viagens. Deverá retornar viagens. **Não deverão ser mais usadas as third-party APIs.**
3. `/details`
   Para obter informação relativas a uma viagem. Deverá retornar informação de uma viagem.

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 o Flask a forma de fazer com que a ligação à base de dados exista de forma automática sempre que a aplicação recebe um pedido.

5.1. Integração com serviço REST público da Internet

De forma a obter informações sobre:

- Previsão de Tempo, deverão usar a API weatherapi.com [6]
- Preços de viagens, deverão usar a API flightapi.io [7]

NOTA: Apenas o servidor interage com as APIs externas.

6. Tratamento de erros

Sempre que a operação não possa ser executada ou que algum erro inesperado ocorra, o servidor deverá responder ao cliente com uma resposta HTTP incluindo uma descrição detalhada do problema segundo o formato apresentado nas aulas TP sobre a representação JSON (i.e., `application/api-problem+json` [8]). O cliente apresentará a informação da descrição detalhada na consola.

7. Entrega

A entrega do Projeto 3 consiste em colocar todos os ficheiros do projeto numa diretoria cujo nome deve seguir exatamente o padrão `grupoXX` (onde `XX` é o número do grupo). Juntamente com os ficheiros, deverá ser enviado um ficheiro de texto `README.txt` (é em TXT, não é PDF, RTF, DOC, DOCX, etc.) onde os alunos podem relatar a informação que acharem pertinente sobre a sua implementação do projeto (por exemplo, limitações). A diretoria será incluída num ficheiro ZIP cujo nome deve seguir exatamente o padrão

`grupoXX.zip` (novamente `XX` é o número do grupo). Esse ficheiro será submetido num recurso a disponibilizar para o efeito na página de AD no moodle da FCUL. O não cumprimento destas regras podem anular a avaliação do trabalho.

O prazo de entrega do Projeto 3 é **domingo, 23 de abril de 2023, às 23:59 (Lisboa)**.

8. Plágio

Não é permitido aos alunos dos grupos partilharem código com soluções, ainda que parciais, a nenhuma parte do projeto com alunos de outros grupos (nem através do Fórum da disciplina, nem por qualquer outro meio). Além disso, todos os códigos serão testados por um sistema verificador de plágio. Caso alguma irregularidade seja encontrada, os projetos de todos os alunos envolvidos serão anulados e o caso poderá ser reportado aos órgãos responsáveis na Ciências@ULisboa.

9. Referências

[1] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[2] https://flask.palletsprojects.com

[3] https://www.sqlite.org/

[4] http://docs.python-requests.org/en/master/

[5] http://www.json.org/

[6] https://www.weatherapi.com/docs/

[7] https://www.flightapi.io/docs/

[8] https://datatracker.ietf.org/doc/html/draft-nottingham-http-problem-02