UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
Departamento de Engenharia Informática

Project #2
Engenharia de Serviços

2016/17 – 2nd Semester
MEI

Entrega: 2017-04-21

Nota: A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior e futuro profissional. Qualquer tentativa de fraude pode levar à reprovação na disciplina tanto do facilitador como do prevaricador, para além de outras consequências previstas na lei.

A Three-Tier Application on the Cloud

Objectives

- Create scalable and strictly divided three-tier enterprise applications
- Use the **OpenAPI** standard to expose REST services
- Learn to create JavaScript clients with **React**
- Use the **Flask** microframework to provide the web services
- Use an **Object Relational Mapper, SQLAlchemy**

Final Delivery

- You must submit your project in a zip file using Inforestudante. Do not forget to associate your work group mate at submission time.
- The submission contents are:
  - Source code of the project ready to deploy on Amazon.
  - A small report in pdf format (max. of 8 pages) about the implementation of the project, including details of the React interface, the security mechanism and the scaling experiments.

Software

Flask

For this project, you are required to use Flask and deploy the code on Amazon Elastic Beanstalk. You have two options, to develop the early parts of this project: install Apache and WSGI[1] and test everything in your computer before deploying and testing on the cloud; or avoid installing anything and test the project from the beginning on the cloud. Note that **in both cases you should write the code in your computer**, to minimize interaction with the cloud.

You may use **PyCharm** (https://www.jetbrains.com/pycharm/) or **PyDev**, the **Python IDE for Eclipse** (http://www.eclipse.org/ and https://marketplace.eclipse.org/content/pydev-python-ide-eclipse) to develop the project. However, other IDEs or simple text editors may do the job perfectly well. You are free to choose.

Data Persistence

When programming a web application, the use of a database or persistence engine to save data is quite common. In this project, students will have to use the **SQLAlchemy** Object Relational Mapper as it provides Python developers with a facility for managing relational data in Java applications. The suggested database is **MySQL**, but you are free to choose another one. A very important detail here is that your choice **must** exist in the Amazon cloud. Hence, not all options are valid. You should do a small test to check whether your preferred database will work there.

Open API Specification

The OpenAPI specification defines a standard interface to REST APIs for humans and computers to discover the capabilities of some service without access to source code.[2] Programmers can write a file describing the services they want to provide and tie these services to server-side methods that implement them. This greatly improves the organization and maintainability of the code. The OpenAPI specification is orthogonal to the programming language and supports several different options including Python and the Flask microframework.

---

[1] "WSGI is a specification that describes how a web server communicates with web applications, and how web applications can be chained together to process one request" (http://wsgi.readthedocs.org/en/latest/what.html.).
[2] https://github.com/OAI/OpenAPI-Specification.

References

Online Resources

You can find plenty information online for this project:
Flask: http://flask.pocoo.org
SQLAlchemy: http://www.sqlalchemy.org
Flask in Amazon Beanstalk:
http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-flask.html
The OpenAPI Specification:
https://github.com/OAI/OpenAPI-Specification

Project Description

Overview

In this project, students should develop a Web application to manage multi-user music playlists. A playlist contains the identification of a set of songs to be played and the indication of the order in which this playback should take place. This application must store information about users, songs and playlists.

This application should have three distinct layers:

- A database to keep all the information (songs might be stored on the server file system). You should ensure that your project has a clearly demarcated CRUD[3] layer.
- A business layer that interacts with the database, and provides a REST interface to the outer world. Students must define the interaction between client and server using the OpenAPI specification. This will provide a single point of access that is easy to read and understand for all the available services.
- A presentation layer for web browsers, developed in React.

The server should run in the Amazon AWS cloud, using the Elastic Beanstalk service. Refer to the following figure:

[3] Create, Read, Update, Delete.

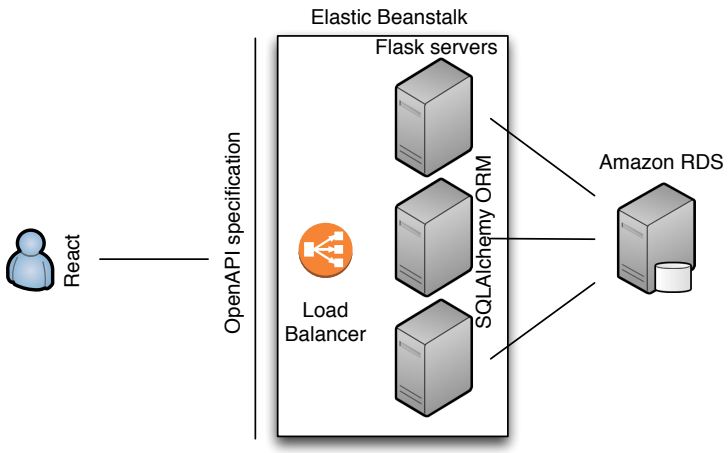

Figure 1 - Overview of the Assignment

Functional Requirements

1. As a new user, I want to create an account, edit my personal information (at any time) and delete the account (deleting any traces of your presence from the system). The personal data to be associated with each account must be (at least) the name, e-mail and user password.

2. As a user, I want to authenticate myself and start a session using my e-mail and the password I chose.

3. As a user, I want to be able to logout from any location or screen of the application.

4. As an unauthenticated user, I should be denied access to any location, except to the login screen.

5. As a user, I want to create new playlists and give them a name.

6. As a user, I want to edit the name of the playlists I have added to the application.

7. As a user, I want to list my playlists in ascending or descending order of name, creation date and size.

8. As a user, I want to list the songs associated with each playlist, by selecting the playlist from my list.

9. As a user, I want to delete a previously created playlist. Deleting the playlist does not delete the songs that are associated with it.

10. As an application user, I want to add and remove songs to a playlist I created.

11. As a user, I want to upload new songs to the server, identifying their title, artist, album, release year and the path to the file.

12. As a user, I want to delete songs that I have added to the application, without deleting the files with those songs or deleting those songs from existing playlists in the application.

13. As a user, I want to edit the song data I have added to the application.

14. As a user, I want to list all the songs registered in the application by all users.

15. As a user, I want to list songs registered in the application that meet search criteria defined by me on the title and/or artist of the songs I'm looking for.

16. As a user, I want to add songs to my playlists from the playlists produced in (14) and (15).

Quality Attributes

A1. Users must be registered and have valid credentials to authenticate and access the application.

A2. User passwords should never be displayed in readable text.

A3. Passwords must be stored in encrypted form.

A4. Users should not be able to view personal data or playlists of other users.

A5. Songs entered in the application can be viewed by any user, but can only be modified or deleted by the owner.

Other Requirements

1. Your server should forbid unauthorized accesses to the services layer. Since these services will serve for a web client, you may use a session-based scheme, i.e., your services layer does not need to be 100% pure RESTful. You should describe the authentication and authorization scheme in the report. You do not need to use HTTPS.

2. One of the crucial aspects of this work is that students will deploy their projects on the cloud. This lets the service scale very easily to accommodate varying numbers of clients (if the application supports scaling). Amazon Beanstalk can auto-scale to manage a (default) limit of up to 4 servers. Then, one might use different rules to

control the scaling in and out of these servers. For example, students could use CPU occupation to control the scaling. If the CPU occupation of the machine(s) raise(s) above a given level, a new machine might be needed. Conversely, if CPU occupation drops below some low threshold, a machine might be released. Students may try other rules as well: for example, using the latency of the responses, or any other rules they like.

To test their service, students may use a tool like Apache Jmeter (http://jmeter.apache.org). Please refer to the manual for help. The idea is to slowly increase the load, to make the auto-scaling mechanism add machines, one by one, and then decrease it again, until all but one machines leave the service, again, one by one. Students should register the results of this experiment in the report. Concretely, they should observe the response times of requests and relate them to the number of machines on the service.

Testing may involve a lengthy execution of some function. I suggest a sloppy implementation of the Fibonacci function (for n >= 1), but students may use some other:

```
def fibonacci(n):
    if n <= 2:
        return 1
    return fibonacci(n - 1) + fibonacci(n - 2)

@application.route('/computefibonacci')
def httpfibonacci():
    try:
        n = int(request.args['n'])
    except:
        return 'Wrong arugments', 404
    return 'Fibonacci(' + str(n) + ') = ' + str(fibonacci(n))
```

Invoking with the argument 34 takes near 10 seconds (depending on the hardware). For example, in the localhost:

```
http://localhost:5000/computefibonacci?n=34
```

DESIGN YOUR EXPERIMENTS CAREFULLY, BECAUSE SCALING INVOLVES COSTS. ALSO, NOTE THAT EACH MACHINE IS CHARGED AT LEAST 1 HOUR. E.G., SCALING OUT FOR 4 MACHINES DURING 5 MINUTES WILL COST YOU, AT LEAST, 4 HOURS.

Good Work!