Instituto Superior Técnico

Mestrado Bolonha Engenharia Eletrotécnica e de Computadores

Distribuited Applications on the Internet

Project - Part 1

Group 2

José Duarte Ferrão de Oliveira Lopes, nº 100336

Francisco Miguel Bernardo Mendes, nº 96529

Professor João Nuno de Oliveira e Silva

October 2023

Index

1 - Qr Code Generator

1.1 Endpoints and Functionalities

1.1.1 Endpoint "/"

Main page that renders the input form for the qr code: a string.

1.1.2 Endpoint "/qrgenerate"

Gets the string input from the previous post, converts it to a qr code, returns it as a png.

2 - Food Service

2.1 End Points and Functionalities

2.1.1 Endpoint "/"

Main Page that lists all the available restaurants on the database.
For each restaurant there are two buttons:

- Menu - go the the menu page for that restaurant (see 2.1.2)
- Rate - go to the evaluate page for that restaurant (see 2.1.3)

2.1.2 Endpoint "/restaurant"

Page that displays the menu items on the database for a given restaurant.

2.1.3 Endpoint "/evaluate"

In this page, users are able to leave an evaluation of a restaurant, submitting the following information:

- User Id
- Restaurant Id
- Evaluation - written text based comentary (optional)
- Rating - integer from 1-5

2.1.4 Endpoint "/post_evaluate"

This endpoint receives the form information from endpoint "/evaluate" (see 2.1.3) and submits it to the database.
The user is served an HTML template based on success or on failure of the evaluation.

2.2 Database Classes

There are 3 database classes for this Component:

- class **Restaurant** (2.2.1)
- class **Menu** (2.2.2)
- class **Ratings** (2.2.3)

2.2.1 class Restaurant

In these table, the information regarding each restaurant is stored:

- **id** - unique int that identifies each restaurant
- **name** - restaurant name
- **menu** - restaurant menu (back populated from class Menu (2.2.2)
- **ratings** - restaurant ratings (back populated from class Ratings (2.2.3))

2.2.2 class Menu

This table stores the menus for each restaurant. Specifically, each instance of Menu contains a dish served in the restaurant, so the entire menu is a collection of Menu objects.
It contains:

- **id** - unique int
- **food** - dish name
- **restaurantId** - id of restaurant it belongs to
- **restaurant** - name of restaurant it belongs to
- **ratings** - we thought users should be able to evaluate each dish if they wanted (might delete later) (back populated from class Ratings (2.2.3))

2.2.3 class Ratings

This table stores each evaluation left by the users:

- **id** - unique int
- **evaluation** - text based evaluation
- **rating** - 1-5 rating
- **userId** - Id of User that left the evaluation
- **restaurantId** - id of restaurant it belongs to
- **restaurant** - name of restaurant it belongs to
- **menuId** - id of menu it belongs to (optional)
- **menu** - name of menu it belongs to (optional)

3 - Food Admin App

3.1 Usage

To use the Food Admin App, follow these steps:

1. Ensure that the XML-RPC server is running at the specified `SERVER_URL`.
2. Run the Python script provided.
3. Choose from the menu options to perform various tasks.

3.2 Functions

3.2.1 `newRestaurant()`

- **Purpose:** Create a new restaurant.
- **Input:** Restaurant Name.
- **Output:** Confirmation message or error message.

3.2.2 `newMenu()`

- **Purpose:** Create a new menu item for a restaurant.
- **Input:** Restaurant ID and menu item name.
- **Output:** Confirmation message or error message, if restaurant id not found or if server not reachable.

3.2.3 `delRestaurant()`

- **Purpose:** Delete a restaurant.
- **Input:** Restaurant ID.
- **Output:** Confirmation message or error message if restaurant id not found or if server not reachable.

3.2.4 `delMenu()`

- **Purpose:** Delete a menu item.
- **Input:** Menu item ID.
- **Output:** Confirmation message or error message if menu id not found or server not reachable.

3.2.5 `listRestaurants()`

- **Purpose:** List all restaurants.
- **Input:** None.
- **Output:** List of restaurants or error message if server not reachable.

3.2.6 `showMenu()`

- **Purpose:** View a restaurant's menu.
- **Input:** None (optional restaurant Id to see specific restaurant menu)
- **Output:** Restaurant menu(s) or error message.

3.2.7 `showRatings()`

- **Purpose:** Show ratings for a restaurant.
- **Input:** Restaurant ID.
- **Output:** Ratings for the restaurant or error message if restaurant not found or server not reachable.

3.3 Main Loop

The program runs in an interactive main loop where the user can choose from various options.

4 - Room Service

4.1 Endpoints

4.1.1 Endpoint "/"

The "Spaces" page lists all the available spaces on the database.
For each space, there is a button to see its schedule.

4.1.2 Endpoint "/schedule/<space_id>"

A page that displays the schedule for the space with Id `space_id`.
If given a non-existing `space_id`, returns `Error 404:Not Found`.

4.2 Database Classes

There are 3 database classes for this component:

- class **Space** (4.2.1)
- class **Course** (4.2.2)
- class **Event** (4.2.3)

4.2.1 class Space

In this table, for each space is stored the following information:

- **id** - unique integer that identifies the space.
- **name** - space name.
- **events** - space events (back populated from class Event (4.2.3).

There are still two methods associated with this class:

- `to_dict()` - create a dictionary with all the attributes of Space Class.
- `to_dict_simple()` - create a simpler dictionary with some attributes of Space Class.

4.2.2 class Course

In this table, for each course is stored the following information:

- **id** - unique integer that identifies the course.
- **acronym** - course acronym.
- **name** - course full name.
- **events** - course events (back populated from class Event (4.2.3).

There is a method named `to_dict()` that creates a dictionary with all the attributes of Course Class.

4.2.3 class Event

This table stores each existing lesson in a given space.

- **id** - unique integer.
- **weekday** - weekday's Portuguese name.
- **period_start** - start date-time.
- **period_end** - end date-time.
- **info** - type of lesson.
- **space_id** - id of space it belongs to.
- **course_id** - id of course it belongs to.

There are still two methods associated with this class:

- `to_dict()` - create a dictionary with detailed attributes of Event Class, including the course information.
- `to_dict_simple()` - create a simpler dictionary with some attributes of Event Class, including only the course acronym.

4.3 Other relevant information

To update the schedule of a space, the data must follow the structure implemented by "Fénix". This data is transmitted by the Room Admin App (5.2.4).

In the function `updateSchedule()`, the data received is filtered to obtain only the relevant information to save, mainly only saving the events of type "`LESSON`", and when needed converted to specific data types. Were defined two helper functions to filter the data – `getCourseDataFiltered()` and `getEventDataFiltered()`.

5 - Room Admin App

5.1 Usage

To use the Food Admin App, follow these steps:

1. Ensure that the XML-RPC server is running at the specified `SERVER_URL`.
2. Run the Python script provided.
3. Choose from the possible options to perform various tasks.

5.2 Functions

5.2.1 `newSpace()`

- **Purpose:** Create a new space.
- **Input:** Space ID and name.
- **Output:** JSON dictionary with the new space information, or error message.

5.2.2 `getSpaces()`

- **Purpose:** List all spaces.
- **Input:** None.
- **Output:** List of JSON dictionaries of the spaces, or error message.

5.2.3 `getScheduleBySpace()`

- **Purpose:** View events of a space.
- **Input:** Space ID.
- **Output:** Space events, or "No events schedule.", or "Space with Id `space_id` not found.", or error message.

5.2.4 `updateSchedule()`

- **Purpose:** Read a JSON file with the schedule of a space and send the data to the server.
- **Input:** JSON file name.
- **Output:** Confirmation of a successful update, or error message.

5.3 Main Loop

The program runs in an interactive main loop where the user can choose from various options.

6 - Check In/Out App

6.1 Endpoints and Functionalities

The Check In/Out program provides the following endpoints and functionalities:

6.1.1 Endpoint "/checkin"

- This endpoint renders the check-in page, allowing users to input their user ID and the room ID they wish to check-in in a form.

6.1.2 Endpoint "/post_checkin"

- This endpoint handles the submission of the check-in form. It checks if the user has already checked in and not checked out, then records the check-in time.
- Parameters:
  - `userId`: User's ID
  - `roomId`: Room's ID
- Response: Failure if the user is already checkec-in a room, Success otherwise.

6.1.3 Endpoint "/checkout"

- Description: This endpoint renders the check-out page, allowing users to input their user ID.

6.1.4 Endpoint "/post_checkout"

- This endpoint handles the submission of the check-out form. It records the check-out time for the user if a valid check-in record is found.
- Parameters:
  - `userId`: User's ID
- Response: Failure if the user is not checked-in a room, success otherwise.

6.1.5 Endpoint "/listcheckinout"

- This endpoint lists the check-in and check-out records for a given user. Users can input their user ID to view their check-in/out history.
- Parameters:
  - `userId`: User's ID
- Response: A table displaying the user's check-in/out history.

6.2 Database Classes

The program uses SQLAlchemy to interact with the SQLite database. It defines the following database class:

6.2.1 class CheckInOut

- It represents a check-in/check-out record in the database.
- Attributes:
  - `id`: Integer, primary key
  - `userId`: String, non-nullable
  - `roomId`: Integer, non-nullable
  - `checkin`: Time, non-nullable
  - `checkout`: Time
- Methods:
  - `__repr__()`: Returns a string representation of the record.

7 - Message App

7.1 Endpoints

7.1.1 Endpoint "/"

A simple and temporary page to redirect to the main pages.

7.1.2 Endpoint "/send_message"

A page to send messages from a user to another user. It has as required parameters:

- **From:** user-id the message is to be sent from.
- **To:** user-id the message is to be sent to.
- **Message:** message to be sent.

7.1.3 Endpoint "/post_message"

This endpoint handles the sending of the message. It records the message information, including the date-time when it was sent.

7.1.4 Endpoint "/sent_messages"

A page that lists the messages `sent` by a specific user with id `user_id` submitted by the active user.

7.1.5 Endpoint "/received_messages"

A page that lists the messages `received` by a specific user with id `user_id` submitted by the active user.

7.2 Database Classes

There is only one database class in this app.

7.2.1 class Message

It records every message sent and its information:

- **id**: primary key, unique integer.
- **from_user_id** - the user-id the message was sent from.
- **to_user_id** - the user-id the message was sent to.
- **datetime** - the date-time when the message was sent.
- **message** - message sent.

8 - Conclusion

In this part of the project it was possible to implement every service and functionality asked.