Problem Statement

Project Title: Agenda API

Date: 16/03/2020 **Prepared By: Ishdeep Bhandari**

Product Characteristics and Requirements:

1. Build using Rest Api
2. Create and manage appointments
3. Manage appointments per user
4. Create and manage users
5. Get available time slots for users

Product-related deliverables:

Report, software code, test plan.

Project Success Criteria: My main goal is to complete this project within allotted deadline. Also, if I can showcase the use of Java, my thinking and implementation process. Furthermore, solving the case with the best of my abilities.

Statement of Work

1. Scope of Work

The scope of the case was general development and testing. It involved showcasing and implementing the requirements in order to be assessed for the position of a junior java developer at Topicus.

2. Location of Work

Work from home due to the coronavirus situation.

3. Period of Performance

The duration of the implementation is slightly more than 2 days.

4. Deliverables Schedule

Deliverable	Days Worked
1. Understand the requirements of the case. 2. Devise a plan for the implementation.	Wednesday
1. Familiarize with the Rest-Api and frameworks: Spring boot, Postgres, Hibernate, JPA and Postman. 2. Create a sample project to understand the mvc model.	Saturday
1. Work on creating the agenda api (building on the sample project).	Sunday

2. Some further changes and testing	
1. Check code for better implementation 2. Create an implementation and a test document	Monday

6. Deliverables Quality Criteria

- The design and implementation plan would be later checked and verified by the interviewer and Topicus.
- Familiarity with the Rest-api framework would be checked during the final interview.
- The final product will be discussed in detail with the interviewer to test and understand its implementation.

6. Additional Requirements:

- Have some prior knowledge of java
- Familiarizing with the structure Rest Api
- Familiarizing with Spring boot MVC and JPA implementation
- Familiarizing with using Postgres server
- Familiarizing with Postman for testing use cases.

Software Design Document

The agenda API is a Rest Api that helps users to create and manage appointments with each other. Since the scope of the case was backend and due to time challenges, no views were added to the application. But based on the backend implementation essentially there are a few views that can be derived from them. Registering a User, making an appointment with another User, managing user specific appointments, Searching for available time slots. The user would be essentially logged in into the application using credentials (this was not implemented in this api), so all the web mapping for functions in the application is done beginning with the userid to bring the impression of logged in user.

1. Product Features

The features that are available to the user are:

- Create a new user
- Create an appointment with another user using timelsots of half hours.
- Edit that appointment with that user
- Delete an appointment only as an owner
- Get available timeslots for a date
- Get available timeslots for a date and time

2. Classes and Characteristics

➢ Models

1. User Model

This model was created as a java class while associating database associations to the variables using @Entity, @Column (as described by JPA and converted to sql queries using Hibernate) etc. including getters and setters. This model was used to store users in the database.

2. Appointment Model

This model was created as a java class to provide the user to manage appointments using same database associations to the variables.

3. Repositories

The two repositories user and appointment were created as interfaces that extended the Jpa repository. This allowed me to use java functions to query information directly from the database. There were custom functional queries added to each repository to help during web requests.

➢ Controllers

1. User Controller

This controller was designed to provide functions for creating, editing, deleting and getting users from the database. The web mapping for the functions of POST, GET, PUT and DELETE were done in a way that from all users pick the user with the given id.

2. Appointment Controller

This controller is where most of the functionality of the application was provided. The request mapping for all the functions were done in a way that a user would be logged in and only then can they access the separated functionalities with specific mapping.

- The first function is used to get all appointments for a given user by their id.

- The next function uses the userid of any user and the date to provide a list of half hour slots that the user can use to create an appointment with that user. This showcased that the current user (owner) would select which user (addition) to make an appointment with and at the backend the application would pass the userid and get all available slots to choose from.

- The next function has similar functionality as the previous but instead of getting all available slots, it gets all slots from a specific time (which is sent in the request mapping)

- The next function is where the user creates an appointment with the selected users. Both ids are passed in the request mapping which makes it beneficial to query relevant information for the users in order to avoid duplication of appointments, timeslots etc.

- The next function is used for editing an appointment. This function shares some functionality with the previous function. But mostly this involves getting an appointment by its id and saving the new information into the existing appointment.

- The final function is deleting an appointment. Here the only the current user (owner) of the appointment can get an appointment (in the request mapping by id) and delete it.

➢ Exceptions

In order to handle some runtime exceptions 3 exception classes were created. Since there were no views in the api, the handling of input validation was done using these exceptions. But these exceptions generally could handle situation if some appropriate validations were missing.

- Resource not found exception class was created to build on the existing exception class to handle relevant situations.
- Error details class was created to provide a timestamp and information on when the exception was thrown and for what situation.
- Global exception handler was built on the ResponseEntityExceptionHandler so that we can use our own and pre-built exceptions during Http requests. This allowed us to the appropriate exception either manually or automatically in unhandled exceptions.

➢ Application properties

The Pom.xml was generated using the web, postgress, maven and jpa dependencies. The application settings included the connection to the server and the dialect to use for conversion to Sql using hibernate.

3. Design and Implementation Constraints

Due to the time challenges and the scope of the application some design choices were made in order to make development easier and provide a working prototype.

- Only time slots of half hours (07:00-07:30 etc) are used and the time would range from 07:00 – 18:00 to represent a working day. This helped in created a simpler logic

for retrieving remaining available slots. It also helped in checking for duplicate timeslots.

- An appointment can be made only between 2 users. This was restriction due to the less experience with the tech stack involved.

- During the editing of an appointment, the additional user cannot be changed. This was an issue faced during the PUT request where if the additional appointment owner id was sent via a json request it would set it to null in the database. So, this could not be rectified in time with the current experience.

4. Testing and dependencies

Testing for the application was done using the Postman Client by using the relevant mapping addresses and the POST, PUT, GET and DELETE requests. The data was sent using json format for the requests. The date time format used was json specific (2020-06-22 and 17:00:00). Time was always entered in half hour slots during POST and PUT requests to simulate a time slot input.

5. Database

In order to manage and store the data, the pgAdmin client along with postgres was used. There were two tables used for the data each based on the user and appointment models respectively. The two tables included:

- Users Table

    1. User id (auto generated and unique) [PK]
    2. Name
    3. Email

- Appointments Table

    1. Appointment id (auto generated and unique) [PK]
    2. Appointment name (not null)
    3. Appointment message
    4. Start time (not null)
    5. End time (not null)
    6. Appointment Owner Id (not null) [FK with user table]
    7. Appointment Addition Id (not null) [FK with user table]

The functions and queries implemented in the controller used these column names as references.

7. Future additions

- Provide login security and credentials to improve security of user data.
- Provide the user with full freedom of appointment time slots.
- Provide the user with freedom to add multiple users to an appointment.

8. References

- https://www.javaguides.net/2019/01/springboot-postgresql-jpa-hibernate-crud-restful-api-tutorial.html
- https://www.callicoder.com/spring-boot-jpa-hibernate-postgresql-restful-crud-api-example/
- https://www.baeldung.com/spring-data-jpa-query
- https://stackoverflow.com/questions/tagged/java
- https://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html
- https://www.postgresql.org/docs/9.1/datatype-datetime.html