

Booking table system

Name: Titu Meda Cristina

Group: 30236

Cuprins

[Deliverable 1 2](#_Toc2055786399)

[Project Specification 2](#_Toc825512728)

[Functional Requirements 2](#_Toc362225657)

[Use Case Model 2](#_Toc324720183)

[Supplementary Specification 6](#_Toc501087280)

[Non-functional Requirements 6](#_Toc1600632703)

[Design Constraints 6](#_Toc729746755)

[Glossary 6](#_Toc178374120)

[Deliverable 2 6](#_Toc1123232848)

[Domain Model 6](#_Toc605147785)

[Architectural Design 7](#_Toc1115440048)

[Conceptual Architecture 7](#_Toc2005513277)

[Package Design 8](#_Toc596515525)

[Component and Deployment Diagram 8](#_Toc1771682286)

[Deliverable 3 9](#_Toc174956409)

[Design Model 9](#_Toc2136237097)

[Dynamic Behavior 9](#_Toc853475638)

[Data Model 4](#_Toc1322833382)

[System Testing 10](#_Toc975007147)

[Future Improvements 10](#_Toc902595900)

[Conclusion 11](#_Toc1374315000)

[Bibliography 11](#_Toc2082771450)

# Deliverable 1

## Project Specification

This project aims to develop a table booking system that allows users to make reservations at any listed restaurants, at a certain date and time, depending on the availability. Admins can register/login and add a new restaurant in the application, update an existing one or even delete one if necessary. Another type of user is the restaurant manager who can also change details of a restaurant they manage, including its bookings list. Any type of user is able to search in the list of restaurants of the application by location or by name. Finally, customers, after deciding where to eat, they can book a table at the desired restaurant, choosing an available date an hour.

## Functional Requirements

1. User registration and authentication

2. User profile management

3. Restaurant management

4. Restaurant search by location or by name

5. Bookings management

6. History of bookings search by customer’s id or restaurant’s id

## Use Case Model

### Use Cases Identification

Use Case 1: Register a new user  
  
Level: User Goal  
Primary Actor: Unregistered User  
Main scenario:  
  
1. Unregistered user visits the registration page.  
2. Unregistered user fills out the registration form with the required information.  
3. Unregistered user submits the form.  
4. System validates the information and creates a new user account.  
5. System sends a confirmation email to the user.

Use Case 2: Authenticate a user

Level: User Goal  
Primary Actor: Registered User  
Main scenario:  
  
1. Registered User visits the registration page.  
2. Registered User enters their username and password.  
3. System validates the credentials.  
4. System logs the user in and redirects them to their dashboard.

Use Case 3: Add a new restaurant  
  
Level: User Goal  
Primary Actor: Admin  
Main scenario:  
  
1. Admin visits the restaurants posting page.  
2. Admin fills out the job the required information.  
3. Application validates input and creates a new restaurant if information is valid, otherwise and error will be shown.

Use Case 4: Update a restaurant

Level: User Goal  
Primary Actors: Admin, Restaurant Manager  
Main scenario:  
  
1. Admin/Manager visits the restaurants listing page.  
2. Admin/Manager searches the restaurant to be updated by location and/or by name.  
3. Admin/Manager modifies the restaurant information as required.  
4. Application validates and updates restaurant if valid information, otherwise an error will be displayed.

Use Case 5: Delete a restaurant  
  
Level: User Goal  
Primary Actor: Admin  
Main scenario:

1. Admin visits the restaurant listing page.
2. Admin searches the restaurant to be updated by location and/or by name.
3. Admin confirms the deletion.
4. System deletes the restaurant from the platform.

Use Case 6: Search restaurant by location and/or by name  
  
Level: User Goal  
Primary Actors: Admin, Restaurant Manager, Customer (any User)  
Main scenario:

1. User visits the restaurant listings page.
2. User browses through the available restaurants.
3. User filters the restaurant listing by location or searches by restaurant name.
4. Application displays the filtered restaurant listing.

Use Case 7: Make a new booking  
  
Level: User Goal  
Primary Actor: Customer  
Main scenario:

1. Customer logins.
2. Customer searches a restaurant on the listing page.
3. Customer chooses a restaurant and selects a date and an hour for a restaurant reservation, along with additional infromation regarding the number of persons and tables needed for the booking.
4. . System validates if there is enough restaurant capacity left for that date.
5. If there are chairs and tables available for the booking, it is created and listed in their reservations history found on their profile page.

Use Case 8: View the bookings history  
  
Level: User Goal  
Primary Actors: Customer, Restaurant Manager  
Main scenario fro Customer:

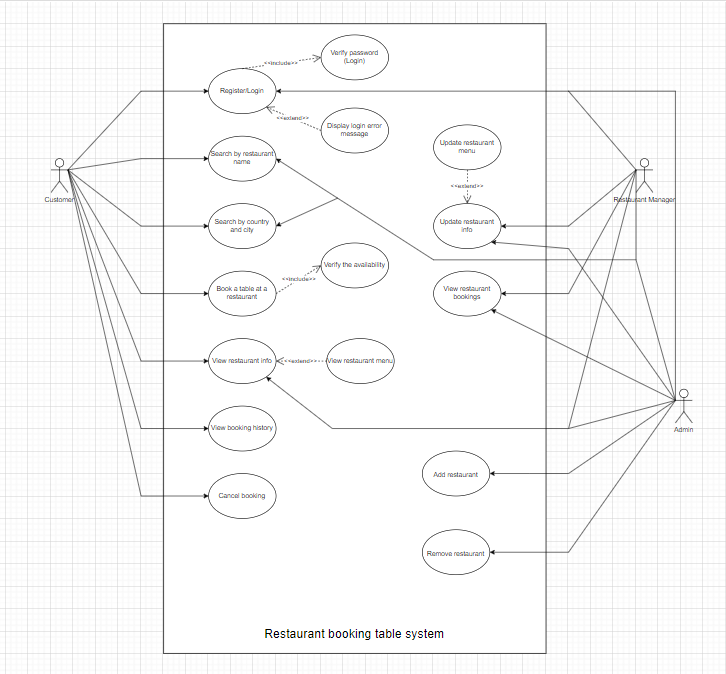
1. Customer logins.
2. Customer enters their profile.
3. On the profile, the customer selects the “View bookings history” option.
4. Another page will be loaded where all the reservations made are displayed one after another.

Use Case 9: Cancel a booking

Level: User level  
Primary Actors: Customer  
Main scenario:

1. Customer logins.
2. Customer enters their profile and opens the bookings history.
3. Customer searches the booking to cancel and selects it.
4. Customer presses the “Cancel reservation” button.
5. The application will remove the booking from its database.

### UML Use Case Diagrams



## Supplementary Specification

### Non-functional Requirements

1. Usability: The application should have an intuitive user interface, making it easy for users to navigate and find the information they need.
2. Scalability: The system should be able to handle a growing number of users and job listings without significant performance degradation.
3. Security: User data must be protected, and sensitive information like passwords should be securely stored using encryption and hashing techniques.
4. Responsiveness: The application should provide quick response times and load times, ensuring a smooth user experience.

### Design Constraints

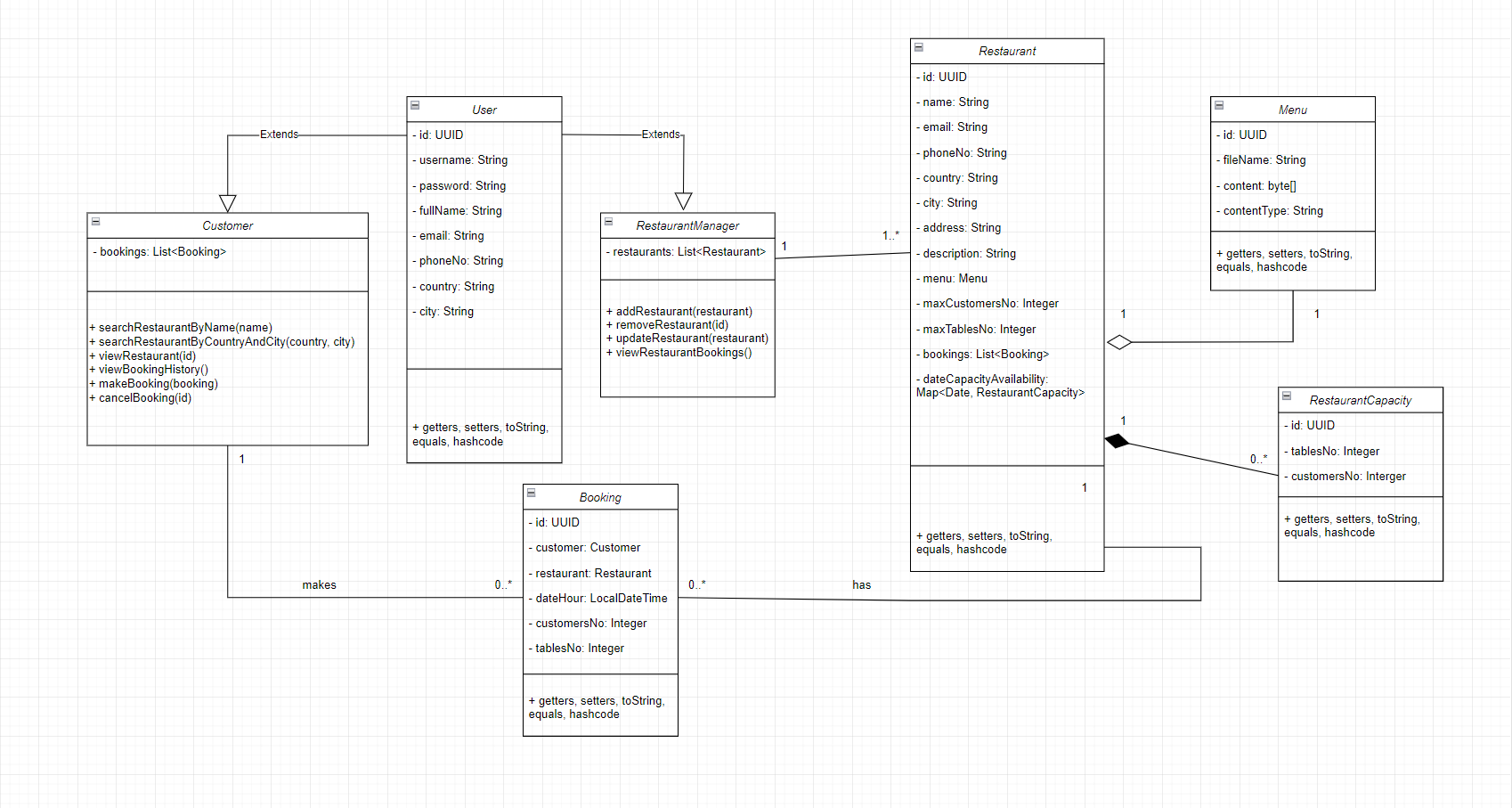
1. The project must be developed using Java and Spring Boot as the back-end framework.
2. Front-end development should use a modern JavaScript framework (e.g., React, Angular, or Vue.js).
3. The database should be designed using a relational database management system (e.g., PostgreSQL, MySQL, or SQL Server).
4. The application should follow a RESTful API design.
5. Code should adhere to best practices, including proper documentation and testing.

## Glossary

# Deliverable 2

## Domain Model

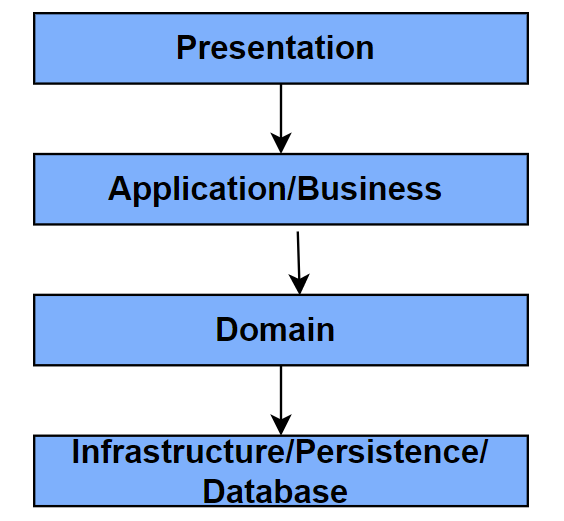
The model of the application consists of the main entities that represent different real objects and define relationships between them. In this project, the domain involves classes that describe the types of users (admin, customer and restaurant manager), the details of a restaurant and of its capacity at a certain date. It also contains classes for a menu document and for a booking.



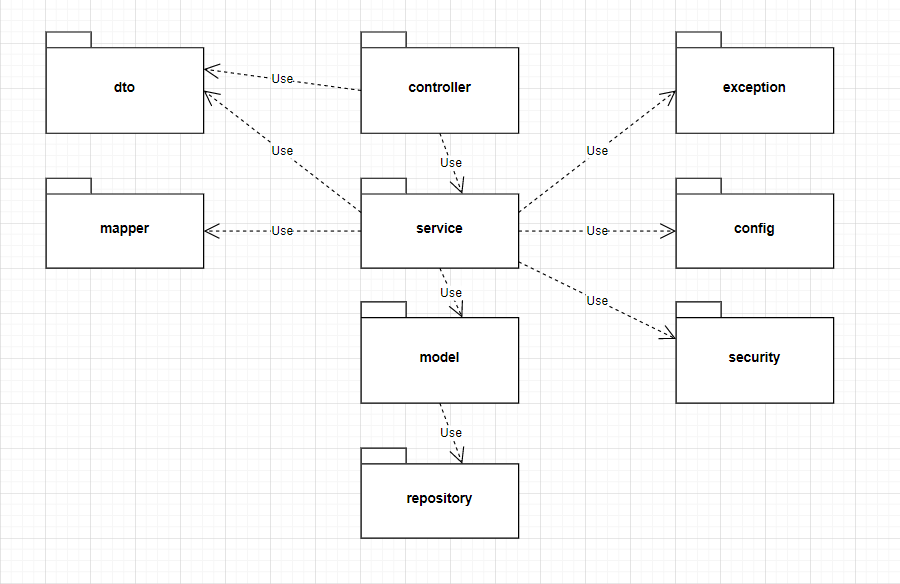
## Architectural Design

### Conceptual Architecture

This application is structured in a layered architectural design. In this way, there are the four main packages organised in a vertical order, starting from the persistence layer, the repository package. Then, above it comes the model package that consists of the entities to be saved in the database. The following layer is the service that validates inputs from user and does business logic using the repository methods. The last package, controller, implements the endpoints that are going to be used by the frontend. In this particular architecture design, every layer will be calling the layer beneath them.



### Package Design



### Component and Deployment Diagram

O imagine care conține text, diagramă, Dreptunghi, linie

Descriere generată automat

Deliverable 3

Design Model

Dynamic Behavior

O imagine care conține diagramă, linie, text, chitanță

Descriere generată automat

Data Model

Entities:

User

* Id, ursername, passwordm fullname, email, phone number, country, city
* Customer extending User has 2 more fields: one list of bookings, in a one to many relation, and another list of favourite restaurants with the same relation
* Restaurant Manager, another User child, also has an extra field, a list of managed restaurants, related in the same way to the restaurant entity (one to many)

Menu

* Id (same as the associated restaurant), file name, content, content type
* One tot one restaurant reference

RestaurantCapacity

* Id, available customers and tables number

Restaurant

* Id, name, email, phone number, country, city, address, description
* One to one relation with the menu entity
* One to one relation with the restaurant manager entity
* Maximum numeber of customers and tables available for reservation
* Map that keeps the remaining capacity of a restaurant at a specific date
* List of bookings in order to keep track of the reservations made (one to many)

Booking

* Id, date and hour, number of customers and tables to reserve
* Customer (many to one)
* Restaurant (many to one)

ContactMessage

* Id, name, email, message for users to request contact with the support

System Testing

JUnit 5 is a popular testing framework for Java applications, offering a comprehensive set of annotations, assertions, and test execution capabilities. With JUnit 5, you can write unit tests to verify the correctness of individual components or methods in isolation. It provides features like parameterized tests, test lifecycle hooks, and flexible test organization, enabling you to structure and execute tests efficiently.

Mockito is a powerful mocking framework that helps in creating mock objects for testing. It allows you to simulate the behavior of dependencies or collaborators, enabling you to focus on testing specific units of code without relying on the actual implementation of external components. Mockito simplifies the process of setting up test scenarios, stubbing method calls, and verifying interactions between objects, thereby enhancing the readability and maintainability of your tests.

Testcontainers is a Java library that provides lightweight, disposable containers for integration testing. It allows you to easily spin up containers (e.g., Docker containers) that encapsulate dependencies such as databases, message brokers, or other external services. By using Testcontainers, you can ensure consistent and reliable test environments, eliminating the need for manual setup or shared testing databases. This approach promotes better isolation and reproducibility of integration tests, enabling you to test your application's interactions with external systems effectively.

When combined, JUnit 5, Mockito, and Testcontainers offer a comprehensive testing ecosystem. You can utilize JUnit 5 for unit tests, Mockito for mocking dependencies, and Testcontainers to facilitate integration testing with realistic and isolated environments. This approach helps ensure the reliability, maintainability, and overall quality of your project by validating its functionality under different scenarios, edge cases, and dependencies.

By leveraging these testing frameworks, you can gain confidence in your codebase, detect and fix bugs early, and support continuous integration and delivery practices, ultimately leading to a more robust and stable application.

Future Improvements

* Filter by country and city with scrollable dropdown with all the countries and its cities
* Search by incomplete name
* Better interface with profile pictures
* Upload menu for restaurants
* Automatically delete reservation two hours after the established booking hour
* Restaurant capacity also per hour, not only day
* Facebook/ Google authentication
* Coupons/discounts for loyal customers (that make many reservations at a restaurant)

Conclusion

In conclusion, the development of the table reservation website brings convenience and efficiency to both customers and restaurant owners alike. By leveraging modern web technologies, intuitive user interfaces, and seamless integration with backend systems, the website provides a user-friendly platform for discovering, exploring, and booking reservations at a variety of restaurants.

Regarding the development part, the project was a very good way to learn new web technologies such as Springboot for backend and Angular for frontend, architectures, design patterns, REST and HTTP protocol, JPA and Hibernate and many others.

Bibliography

* baeldung.com
* stackoverflow.com
* youtube tutorials