Software Design Document
Holy Crushade Project

Document Content

1. System Overview
2. Design Considerations:
    - Dependencies
    - General Constraints
    - Goals and Guidelines
    - Development Methods
3. Architectural Strategies
4. System Architecture
5. Policies and Tactics

System Overview

In general, system was developed to meet the needs of Munich residents and provide them a convenient way to find restaurants in the city and book tables at any time. It provides various functionality such as: Search for different types of restaurants; Filter search results by certain criteria; Display detailed information about restaurants; Make reservation with specific date, time, amount of people; Modify reservations; Delete (cancel) reservation; Confirm reservations.

Design Considerations

1. **Dependencies:**
    - Spring framework
    - Java JDK 17+
    - Jsoup
    - Javax
    - Javafx
    - OKHttp
    - HTMLUnit
    - FasterXML

2. **General Constraints:**
    - University activities.
    - Developers’ skills.
    - Short development time.

3. **Development Methods:**

As development method, we've used shortened version of Scrum.

Scrum – is a combination of agile and waterfall methodologies, which encourages teams to learn through experience, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.

4. **Goals and Guidelines:**

- Software must have at least basic functionality.
- Team members should always communicate with each other.
- Code standards must be met.
- Team members should be always working on some parts of project.
- Opinion of all team members matters and is taken seriously.
- Team members must adapt to continuous changes in development.

Architectural Strategies

Layered (n-tier) architecture – is used for server implementation.

In this architecture the code is arranged so the data enters the top layer and works its way down each layer until it reaches the bottom, which is usually a database. Along the way, each layer has a specific task, like checking the data for consistency or reformatting the values to keep them consistent. It's common for

different programmers to work independently on different layers.

Event-driven architecture – is used for client implementation.

Sometimes there’s data that needs processing, and other times there isn’t. The event-driven architecture helps manage this by building a central unit that accepts all data and then delegates it to the separate modules that handle the particular type. This handoff is said to generate an “event,” and it is delegated to the code assigned to that type.

System Architecture

Let’s start from server implementation. It is divided into 2 parts, REST layer and SERVICE layer.

REST layer contains REST endpoints which receive requests from client application and then delegate them to SERVICE layer, then it sends the answer to client, thus showing the query result.

SERVICE layer contains all functionality. It takes query parameters received by REST endpoints and fulfills the request responding with ready to send data.

Client implementation contains one main class ClientApplication.java and different scenes. Main class accepts all data and then delegates it to different scenes, which then are displayed to the end user according to this information. It can idle and wait for further input, main class also provides connection between client controllers and scenes, so the application can send requests to the server.

In addition, system contains common package, which has classes for restaurant, review and reservation objects. This package is shared between client and server.

Policies and Tactics

1. All developers must add documentation to all methods they write.
2. All developers must write semantic commits, conventional commits.
3. No changes must be pushed directly to main branch, everything must be done via pull requests.
4. All changes must pass all tests before being committed.
5. Code must be refactored after reaching the working state.

6. Big functionality must be divided into smaller pieces if possible.
7. Big changes must be discussed before application.
8. Failed tasks must be explained.
9. Regarding any questions, contact team leader.
10. All developers must attend meetings if they are told to.