<Online Ticket System>

System Design

<Version>

<Date>

<Mert Mısırlıoğlu, Gizem Gümüşçekiççi, Dilara Ünbay, Özay Ezerceli>

Prepared for

SE301 Software Engineering



Table of Contents

[1. Introduction 1](#_Toc433996772)

[1.1. Purpose of the System 1](#_Toc433996773)

[1.2. Design Goals 1](#_Toc433996774)

[1.3. Definitions, Acronyms, and Abbreviations 1](#_Toc433996775)

[1.4. References 1](#_Toc433996776)

[2. Current Software Architecture 1](#_Toc433996777)

[3. Proposed Software Architecture 1](#_Toc433996778)

[3.1. Overview 1](#_Toc433996779)

[3.2. System Decomposition 1](#_Toc433996780)

[3.3. Hardware Software Mapping 2](#_Toc433996781)

[3.4. Persistent Data Management 2](#_Toc433996782)

[3.5. Access Control and Security 2](#_Toc433996783)

[3.6. Global Software Control 2](#_Toc433996784)

[3.7. Boundary Conditions 2](#_Toc433996785)

[4. Subsystem Services 2](#_Toc433996786)

[5. References 2](#_Toc433996787)

SYSTEM DESIGN DOCUMENT[1]

The System Design Document (SDD) is written after the initial system decomposition is done, and updated throughout the development. SDD describes the services provided by each subsystem. Although this section is usually empty or incomplete in the first versions of the SDD, this section serves as a reference for teams for the boundaries between their subsystems. The interface of each subsystem is derived from this section and detailed in the Object Design Document.

SDD is used to define interfaces between teams of developers and serve as a reference when architecture-level decisions need to be revisited. The audience for the SDD includes the project management, the system architects (i.e., the developers who participate in the system design), and the developers who design and implement each subsystem.

# Introduction (Gizem)

Provide a brief overview of the software architecture and the design goals. It also provides references to other documents and traceability information (e.g., related requirements analysis document, references to existing systems, constraints impacting the software architecture).

## Purpose of the System

## Design Goals

## Definitions, Acronyms, and Abbreviations

## References

References to existing systems, etc.

# Current Software Architecture (Mert)

Describe the architecture of the system being replaced**. If there is no previous system**, this section can be replaced by **a survey of current architectures for similar systems**. The purpose of this section is to make explicit the background information that system architects used, their assumptions, and common issues the new system will address.

# Proposed Software Architecture

Documents the system design model of the new system.

## Overview (Mert)

Present a bird’s-eye view of the software architecture and briefly describes the assignment of functionality to each subsystem.

## System Decomposition (Dilara)

A close up of a map

Description automatically generated

*Figure 1: Subsystem Decomposition*

Online Ticket System is decomposed based on a Model-View-Controller (MVC) architectural design. MVC was a fitting choice for our system since it provides a faster development process and supports various changes without affecting the entire model. The system is decomposed into three levels: Model, View and Controller.

Model level is responsible for the data and where it is stored. It is decomposed into three components.

* Event Storage is responsible for receiving the event information. Event information includes event name, artist name, event description, event date, event stage etc.
* Ticket Storage is responsible for receiving the ticket information. Ticket information includes event information and the user information.
* User directory is responsible for receiving the user information. User information includes name, surname, birthday etc.

View level is responsible for showing the output to the user using an interface. It displays the information told by the Controller level or the Model level. It also informs the Controller level about the user requests.

* User Interface provides the view to the user. Also, it is responsible for getting the user inputs and informing the Controller level.

Controller level is responsible for managing the user outputs and passing them to View level. Also, it provides appropriate inputs for the user. It establishes the connection between View and Model levels. It is decomposed into three components.

* User management handles the user related requests and provides a way to manage the user data on the Model level.
* Session management establishes the identification between the client and the server by sending and receiving requests.
* Component management handles the event and ticket related requests and provides a way to manage the event and ticket data on the Model level.

## Hardware Software Mapping (Dilara)

A screenshot of text

Description automatically generated

*Figure 2: Deployment Diagram*

Online Ticket System is distributed to 3 layers named as User Machine, Web Server and Database Server. Based on an MVC architecture User Machine corresponds to View level, Web Server corresponds to Control level and Database Server corresponds to Model level. The User Machine layer will perform as the user interface of the client that handles user inputs and sends requests to the server. The Web Server layer that consists of Website Application (Django) component handles the client requests and performs changes or additions to the data accordingly. The Database Server layer consists of the PostgreSQL component which is used as the database management system (DBMS) that manages the data of the system. The system operates with a single database which ensures data integrity. However, it lacks on security because all the data, including users, can be accessed through the same database.

## Persistent Data Management (Dilara)

Online Ticket System is a web application that allows ticket purchase to its users. Therefore, it stores user accounts. The system stores date joined, email address, first name, last name, last login, user type, password and username of the users using PostgreSQL. The system also stores events and stages to carry out its main services. Thus, the system stores stage, date, name, price, rules and quota of the events and address, place and quota of the stages. In addition, the system stores the tickets bought by the users, which include event, user and seat number fields.

## Access Control and Security (Dilara)

A screenshot of a cell phone

Description automatically generated

*Figure 3: Access Matrix*

The database server of Online Ticket System stores user information such as email address, username, password etc. To ensure that the system establishes control and security, we aim to keep critical information as safe as possible by making use of the Hash Crypto Engine provided by the Django on the system’s database.

Hash Crypto Engine is a storage algorithm that can be used by Django. Since it enables a high level of flexibility and supports many applications, it is a commonly used default encryption feature for the database. Online Ticket System uses SHA-256 hash algorithm provided by the crypto engine to create almost-unique signatures. Therefore, critical data such as user passwords are stored securely.

Since, the encryption is managed in the Model level of the system, it is unnecessary to handle any encryption in the Web Server layer.

## Global Software Control (Özay)

Describe how the global software control is implemented. In particular, this section should describe how requests are initiated and how subsystems synchronize. This section should list and address synchronization and concurrency issues.

## Boundary Conditions (Özay)

Describe the start-up, shutdown, and error behavior of the system. (If new use cases are discovered for system administration, these should be included in the requirements analysis document, not in this section.)

# Subsystem Services (Özay)

Describe the **services provided by each subsystem**. Although this section is usually empty or incomplete in the first versions of the SDD, this section serves as a reference for teams for the boundaries between their subsystems. The interface of each subsystem is derived from this section and detailed in the Object Design Document.

# References (Özay)

The following is an example of listing a book in this section. Check the text to see how it is cross referenced (The whole document is based on [1]).

1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.