<Online Ticket System>

System Design

<1.0>

<15.12.2019>

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

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]

# Introduction (Gizem)

We are providing an online ticket selling system which will provide tickets to the user. Users will be able to purchase tickets easily and safely.

## Purpose of the System

Our purpose is to create a web-based platform to provide online tickets. There are several different online ticket selling systems, but our purpose is to create better system that can be easily understood and easily used by users for this purpose we will reduce the complexity of the website which we will achieve by simplifying the interface of the website.

Our system will be used by all kind of users. In the website users can explore the events and buy tickets and when users are logged in their information will remain private for privacy reasons.

## Design Goals

The design goals represent the desired qualities of online ticket system and provide a criterion about the system.

**Performance:**

The system should respond to the user when necessary in a considerable amount of time approximately 1 second to 10 second.

**Scalability:**

The system should support and handle multiple users and responses at the same time.

**Usability:**

The system should be available and accessible to all kind of users.

**Availability:**

Users should be able to use the system without downloading anything. Since the system will be web-based users will able to access the system through the web.

**Security:**

The personal data and the payment information of the users will remain private and secure. The system will handle the security attacks by preventing unauthorized parties.

**Learnability:**

The users should learn the functions of the website easily for that purpose the interface of the website will be simplified.

**Feedback:**

The user will always get feedback when completing a function in the website also in any errors or crashes user will get proper instructions.

## Definitions, Acronyms, and Abbreviations

The abbreviations used throughout the report as well as some definitions are given below.

*Event* A visual and a written information in which enables users to view.

*Buy* Function to get ticket.

*Publish* Admins making events publicly available for the user on the website.

*Consumer* An individual using the system.

*Operator* An individual which hosts events through the system.

*Admin* User to arrange events and communicates with the operators.

*Search* Different criteria functions to find an event

*MVC* Model-View-Controller Structure

*HTTP* Hyper-Text Transfer Protocol.

## References

Biletix.com

# Current Software Architecture (Mert)



The current system is based on an MVC architecture and subsystems inside of these defined three subsystem such Model, View, Controller subsystems.

The **Model** contains only the pure application data such event, user, ticket and it contains no logic describing how to present the data to a user.

The **View** presents the model’s data to the user. The view knows how to access the model’s data, but it does not know what this data means or what the user can do to manipulate it. User requests are created here and send to Model subsystem via Controller subsystem.

The **Controller** exists between the view and the model. It listens to events triggered by the view (or another external source) and executes the appropriate reaction to these events. In most cases, the reaction is to call a method on the model. Since the view and the model are connected through a notification mechanism, the result of this action is then automatically reflected in the view.

# Proposed Software Architecture

## Overview (Mert)

To decompose our system into smaller components, we have used “MVC (Model-View-Controller)” architecture. We have tree main subsystem and other subsystems that included to these main subsystems. Model Subsystem, View Subsystem and Controller Subsystem. The subsystem decomposition, with more details, is given in the next section. Following the subsystem decomposition, hardware and software mapping od the system is illustrated in its respective section. “Persistent Data Management” part discusses the types of the data that the system will store. For the data, accessibility and security details are highlighted in the “Access Control and Security” part. Finally, in the last two sections, global software control and boundary conditions are covered up.

## 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)

Procedure-driven control would be the one that will be appropriate for our system. In our system, operations wait for user (visitor, admin, operator, registered user) to provide the required information or action whenever they need data from either website or a user type. For example, since our project is a web-based project, the web server and database server wait for requests from the web browser. Upon receipt of a request, the web server processes and dispatches it to the appropriate web page and database provides appropriate information which are requested, thus resulting in an event-based control flow. Finally, every user type can realize functions as how they intend to view events, his/her tickets or details about account.

## Boundary Conditions (Özay)

**3.7.1. Initialization**

Since Online Ticket System is web-based application, it does not need explicit installation execution. So, user access the application via a web browser. An internet connection is needed to reach the application. There is different type of users that one of them does not need to be logged in to the system but this type of users is restricted from some functionalities and the rest need to be logged in to the system to use it. First thing that users see is home page of application where all events are listed and showed.

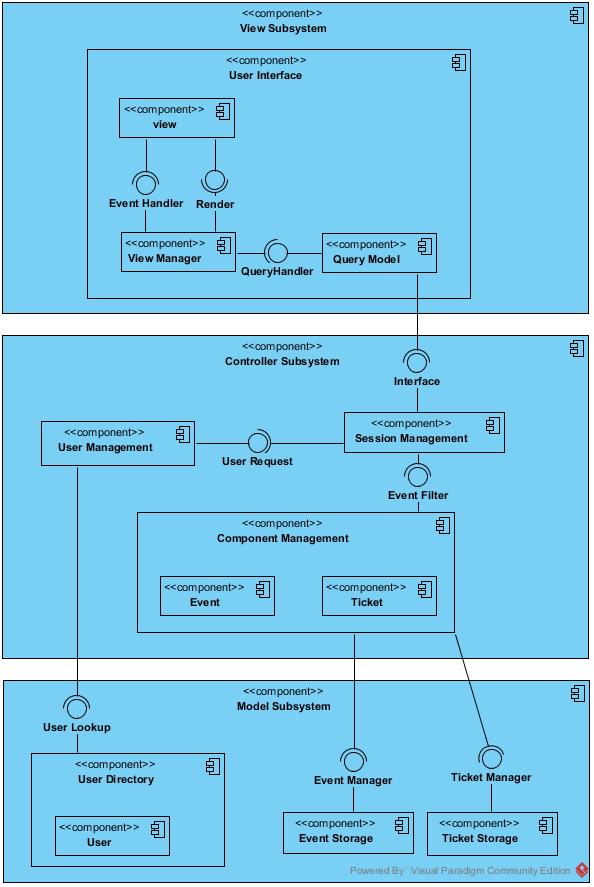
**3.7.2. Termination**

Termination is acquired by logging out from the application. This progress is related with session management subsystem. When we logout we would be terminated one of the subsystems and rest of the subsystems would be notified from this termination and they’ll keep process in an appropriate way with other subsystems and database would be updated continuously via webserver by communicating with subsystems in the application.

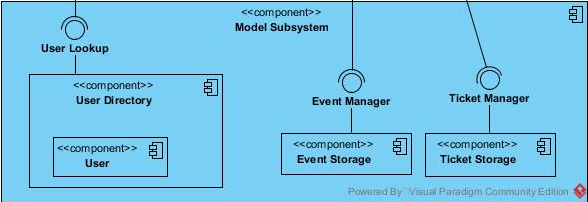
**3.7.3. Failure**

The system shows error messages when a user fails to login by giving the wrong password or username information and to sign up by not satisfying required information. Web server may not respond due to internet connection errors. When there is not any response from the sent request, the timeout will be applied in order to prevent the busy wait.

# Subsystem Services (Özay)

Figure 1

## Model Subsystem



Model Subsystem is responsible for data management and data communication. This subsystem is secure because it can be accessed by the Controller Subsystem only. Thus, every modification or action happening in Model Subsystem must have a corresponding request made by the Controller Subsystem. In addition, Model Subsystem is divided into 3 main components in order to achieve different kinds of functionality for different cases in the Data Management system. Thus, each component has different responsibilities. Those components are:

**4.1.1. User Directory Component**

User Directory Component is responsible for managing User specific data accesses and data modification. It can be accessed only if a request has been made by the Controller Subsystem. In details, Component Management in Controller Subsystem understands whether data access is necessary or not. After that, the necessary request is made into the component. Basically, this component provides two main operations:

* Storing user information such as their username, phone number, e-mail, birthday etc.
  + When a user signup or want to edit his/her account, the information will be delivered to this component in order to access and modify the relevant data as well.
* Retrieving User information
  + When a user wants to see their profile and their tickets, this component needs to be called in order to access the relevant database.

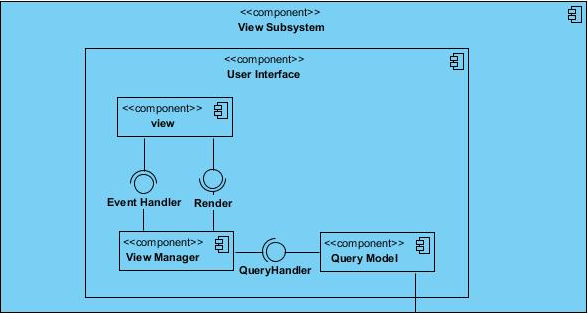
**4.1.2. Event Storage Component**

* Event Storage Component is responsible for storing the events. In addition, it is responsible for communicating with the Component Management Component in order to send relevant event into the component whenever necessary.

**4.1.3. Ticket Storage Component**

* Ticket Storage Component is responsible for storing the tickets. In addition, it is responsible for communicating with the Component Management Component in order to send relevant ticket into the component whenever necessary.

## View Subsystem



View Subsystem and subsystems that included to this subsystem are responsible for managing front-end. What this subsystem does is getting user input and first sending these requests into related component in the controller subsystem. View subsystem provides user interface and it is accessible via a web browser or a web application. The content displayed is accessible by the users and such inputs, request will be sent into controller subsystem then from controller subsystem towards to model subsystem to get and send related context to user. In this view subsystem will be used HTML5, CSS, Bootstrap, JS technologies. This subsystem consists of three components.

**4.2.1. View Component**

* View Component provides the user interface and gets the user input. These operations will be performed by the request made by the View Manager Component and actions done the input taken from the user will be provided to the View Manager Component.

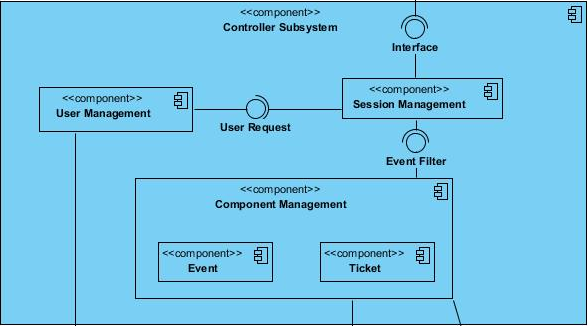
**4.2.2. View Manager Component**

* View Manager Component is the manager of this subsystem. It will get requests from the Query Component and it will receive the user input from the View Component.
* Component will get any kind of backend requests such as database requests which are relevant to the front-end and deal with such information.
* Modifying views according to requests emerged in View Subsystem or input coming from Controller Subsystem.
  + Updates the view via Render method
* Sending clients side requests coming from views to the Controller Subsystem.

**4.2.3 Query Model**

Query Component is for getting and dealing with any kind of database queries send by the user input. It is used and constructed by the View Manager Component. To give an example of Query component’s responsibility that can be getting the user’s search for filtered event and sending such request into View Manager Component.

## Controller System



Controller Subsystem mediates between the View Subsystem and Model Subsystem. It operates on the server side and performs the core functionality of the system. This subsystem features three main subcomponents:

**4.3.1. Session Management Component**

This subcomponent directs the information received from the View Subsystem to appropriate subcomponents. The functionalities of this component are:

* When user inputs for login, this information are received from Controller Subsystem and transferred to the User Management subcomponent. User Management Component checks information and validates using User Directory component in the Model Subsystem.
  + Session Management requests user information from User Management. According to that request, User Management provides necessary user information back to Session Management. After the Session Management component receives the information, it responds back to View Manager via Query Model to inform the user about authentication.
  + The searched event can be got from obtained information if there any filter by the user request via Component Management Component.

**4.3.2 User Management Component**

* This subsystem handles the management of user-related tasks. It controls database of users over the Model Subsystem. User authentication, account, my ticket functionality and other user services are addressed by this component. In brief, there are two scenarios that this component may be invoked.
  + When user want to edit his/her account, a request has been created by the Edit Account interface. The request is accepted by the Controller Subsystem, then sent to the User Management Component. Finally, edit account response is sent to the User Database.
  + When the user wants to check their profile, the request is created by the View Manager Component. Then, request is sent through Controller Subsystem and User Management Component. When the request is received, this component invokes User Directory Component for relevant information. After information is retrieved from the User Database, the received information is sent back to the requesting components.

**4.3.3. Component Management Component**

* This component deals with the event and ticket related tasks. It consults the Model Subsystem to retrieve event and ticket data. Once it has access to event, it runs other related tasks to complete buying ticket process and return ticket information. Result are passed to the Session Management subsystem back. There can be two scenarios where this component might be invoked.
  + When there is a request from the Controller Management Component regarding the events or ticket, this component should get the request and send another request to the Event Storage Component or Ticket Storage Component according to request. Finally, the relevant information from the Database Component should be retrieved and sent back to the Session Management Component.

# References

1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.
2. Visual Paradigm
3. Draw.io
4. Course materials (Slides etc.)