<Online Ticket System>

Object Design

<1.0>

<23.12.2019>

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

Prepared for

SE301 Software Engineering



Table of Contents

[1. Introduction 1](#_Toc436772639)

[1.1. Object Design Trade-offs 1](#_Toc436772640)

[1.2. Interface Documentation Guidelines 1](#_Toc436772641)

[1.3. Definitions, Acronyms, and Abbreviations 1](#_Toc436772642)

[1.4. References 1](#_Toc436772643)

[2. Packages 1](#_Toc436772644)

[3. Class Interfaces 1](#_Toc436772645)

OBJECT DESIGN DOCUMENT

# Introduction (Gizem)

Online Ticket System is designed to present a better and simpler solution for selling tickets. The complete system includes functions for the user, the operator and for the admin. In the system the user will be able to preview events and buy tickets with multiple payment methods. The operator can request to add events to the system and operator also authorized to change the event details as for admin, the admin will accept the requests coming from the operator and has wide authority in the system.

The objective of the system is to be simple but also satisfy user needs and every function needed for a ticket selling system. As we said previously our purpose is to create and effective and simple system, for that purpose the complexity of the website has been reduced. The system will include a simple interface consists of the event views and the functions. In the system the user can preview events without logging in, but the user must register and log-in to activate the buy ticket function. When a user registering to the website the information of the users will remain private. Our website has privacy protocols to ensure the privacy constraints are satisfied according to universal privacy policy.

The system will provide a Graphical User Interface (GUI) and will run on a single computer. The tickets, users, events etc. will be stored in the database.

The subsystem decomposition from which the systems and objects in this document were developed is summarized in the figure below:

A close up of a map

Description automatically generated

## Object Design Trade-offs

As with any software system, several explicit trade-off decisions had to be made in the object design of the system. All the code in the system is our own work as a project team.

Our first trade-off agreement was on memory vs response time(performance) since the database of our interface is crucial for the website, we have decided to choose the memory over the response time because it is crucial to keep the data safely over the performance. Another trade-off is about security vs performance since security is important for buying tickets because we need to keep the payment information of the user safely, so we have chosen security over performance.

As a last trade-off we had to decide between delivery time vs. functionality, in this situation we have chosen delivery time because for the testing we wanted to show every function with most of the functionality of the entire system in a limited time.

## Interface Documentation Guidelines

We have adopted specific interface documentation guidelines and conventions in order to make communication more uniform among our team. First, we have named our classes with singular nouns or noun phrases, with each word beginning with a capital letter. All objects derived from the use cases we have made earlier in the RAD document. Error status is returned to the user. Furthermore, every class will have full interfacing information commented before the implementation of that class, including all attributes and methods.

## Definitions, Acronyms, and Abbreviations

*RAD* : Requirements Analysis Document

*GUI* : Graphical User Interface

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

*Ticket*  The prerequisite to attend to the an event

*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

# Packages (Mert)

# 

# A close up of a map Description automatically generated

**2.2 Package Definitions**

**2.2.1. User Interface Package**

**2.2.1.1 The file organization of code for User Interface Package**

This package will include following files:

views.py

This file will contain the views of each functions needed for the package use

templates/

This folder contain the html files of needed for the package use.

**2.2.1.2 Overview of Interface Package**

Most of the user interface will be GUI. The user interface accesses session management and user management packages.. It is connected to the rest of the packages through the functions in the classes.

**2.2.1.3 Dependencies with other packages**

The interface package is not dependent on the other packages. All the other packages will

access interface package to display different kind of menus since it will have all the user

interface functions.

**2.2.1.4 Usage of User Interface Package**

The user interface package will contain the entire GUI so its most important use is to provide menu to all other packages and let other packages interact with each other with those menus. The system will access different areas by menus and interface package will

provide those menus.

**2.2.2 User Management Package**

**2.2.2.1 The file organization of code for User Management Package**

This package will include following files:

views.py

This file will contain the views of each functions needed for the package use

**2.2.2.2 Overview of User Management Package**

The User Management package consist of user functions like buying ticket , event display and edit profile. This package interacts with User Directory package and if it necessary it can change the information about the user in package. User Management package is the only package that has authority to access the User Directory package.

**2.2.2.3 Dependencies with other packages**

This package will get the commands from the user through the user interface package. When user wants to change him/her profile first accesses the edit profile function. Edit profile functions send a request for changing to User Directory package.

**2.2.2.4 Usage of User Management Package**

This package used to process user functions.

**2.2.3 Session Management Package**

**2.2.3.1. The file organization of code for Session Management Package**

This package uses django.contrib.sessions model provided by the django’s default session system.

**2.2.3.2 Overview of Session Management Package**

The details about the django.contrib.sessions model can be found in the document provided by the django.

<https://docs.djangoproject.com/en/3.0/topics/http/sessions/>

**2.2.3.3 Dependencies with other packages**

The interface package is dependent on the User Interface Package and also this package can access the User Management Package and Component Management Package.

**2.2.3.4. Usage of Session Management**

Session Management consists of the users activities. The Session Management provides data about the user information to the user.

**2.2.4 Component Management Package**

**2.2.4.1 The file organization of code for Component Management Package**

This package will include following files:

views.py

This file will contain the views of each functions needed for the package use

**2.2.4.2 Overview of Component Management Package**

Component management handles the event and ticket related requests and provides a way to manage the event and ticket data on the Model level.

**2.2.4.3 Dependencies with other packages**

This package is dependent to the Session Management package and can be access ticket storage package and event storage package.

**2.2.4.4 Usage of User Management Package**

This package used to handle the event and ticket related requests.

**2.2.5 User Directory Package**

User directory is responsible for receiving the user information. User information includes name, surname, birthday etc.

**2.2.6 Ticket Storage Package**

Ticket Storage is responsible for receiving the ticket information. Ticket information includes event information and the user information.

**2.2.7 Event Storage Package**

Event Storage is responsible for receiving the event information. Event information includes event name, artist name, event description, event date, event stage etc.

# Class Interfaces (Dilara-Özay)

* 1. **Event Class** (Dilara)

The Event class allows the operator to add a new event, cancel an existing event, view an event, edit (modify) an existing event and search for events. Also, it allows the admin to view a list of events and the user to view an event. The Event class is as follows:

A screenshot of a cell phone

Description automatically generated

**Class Name: Event**

**Attributes:**

1. *-name: String*

name is a value of type String. It is a private attribute. It holds the name of the event.

1. *-id: Integer*

Id is an event identification number value of type positive Integer. It is a private attribute. It is used in order to distinguish events, since name is not a unique value.

1. *-stage: Stage*

stage is a value of type Stage. It is a private attribute. It holds the stage that event is being held at.

1. *-date: Date*

date is a value of type Date. It is a private attribute. It is used so that the users can view events held on the given date.

1. *-rules: String*

rules is a value of type String. It is a private attribute. It holds the rules description of the event.

1. *-isAvailable: Boolean*

isAvailable is a value of type Boolean. It is a private attribute. It is used in order to check whether the event is available or not. Any event that has been passed will be defined as unavailable in this attribute.

1. *-isAccepted: Boolean*

isAccepted is a value of type Boolean. It is a private attribute. It is used in order to check whether the event is accepted by the admin or not. Any event that has been confirmed by the admin will be defined as accepted in this attribute.

1. *-quota: Integer*

quota is a value of type Integer. It is a private attribute. It holds the maximum amount of people that can attend the event.

1. *-isConcert: Boolean*

isConcert is a value of type Boolean. It is a private attribute. It is used in order to check whether the event is a concert or not.

1. *-isTheatre: Boolean*

isTheatre is a value of type Boolean. It is a private attribute. It is used in order to check whether the event is a theatre or not.

1. *-isSport: Boolean*

isSport is a value of type Boolean. It is a private attribute. It is used in order to check whether the event is a sport or not.

**Operations:**

1. *+addEvent()*

addEvent() is a public function which returns a value of type Event. When this function is called, a new event will be created and sent to admin for confirmation.

1. *+cancelEvent()*

cancelEvent() is a public function which does not return any type of value. When this function is called, the event will be removed from public display based on the given event id. However, it will remain in the database.

1. *viewEvents()*

viewEvents() is a public function which returns a value of type List. When this function is called, a list consisting of all the events will be displayed.

1. *viewEvent()*

viewEvent() is a public function which returns a value of type Event. When this function is called, the event will be displayed in detail based on the given event id.

1. *editEvent()*

editEvent() is a public function which does not return any type of value. When this function is called, the event will be updated based on the changed attributes and sent to admin for confirmation based on the given event id.

1. *searchEvent()*

searchEvent() is a public function which return a value of type List. When this function is called, a list of events will be created and displayed based on the given search criteria.

**Dependencies with Other Classes and Packages:**

A close up of a sign

Description automatically generated

The Event class interacts with Website class, Stage class, Ticket class and Event Storage package. Users/Visitors can handle various event related tasks by accessing the Event class through Website class. For an operator to create an event, there needs to be a stage where the event will be held. Therefore, Event class interacts with Stage class to be created. For a user to attend the event, he or she needs a ticket. Therefore, Event class interacts with the Ticket class. For an admin to see the lists of events, Event class needs to interact with Event Storage package and store the events to display later.

**Exceptions Raised and Exception Handling:**

There are several exceptions that can be raised by the Event class. For example, if an operator tries to create an event leaving some of the information unprovided an exception is raised. To handle this exception, an error message is displayed on the screen and operator is asked to provide the necessary information.

Also, if an operator deletes an information without providing a new one during an event editing process an exception is raised. To handle this exception, an error message is displayed on the screen and operator is asked to provide the necessary information.

Another exception is raised if an operator tries to add an event quota that exceeds the quota of the chosen stage. To handle this exception, an error message is displayed on the screen and operator is asked to change the event quota or the stage.

Lastly, exception is raised if an operator tries to add an event on the same stage and the same date as another existing event. To handle this exception, an error message is displayed on the screen and operator is asked to change the event date or the stage.

* 1. **Ticket Class** (Dilara)

The Ticket class allows the user to buy a ticket to an available event and preview an existing ticket. The Ticket class is as follows:

A close up of a sign

Description automatically generated

**Class Name: Ticket**

**Attributes:**

1. *-price: Integer*

price is a value of type positive Integer. It is a private attribute. It holds the price of a ticket.

1. *-id: Integer*

id is a ticket identification number value of type positive Integer. It is a private attribute. It is used in order to distinguish tickets.

1. *-seat\_number: String*

seat\_number is a value of type String. It is a private attribute. It holds the seat that is assigned to the ticket’s owner.

**Operations:**

1. *+buyTicket()*

buyTicket() is a public function which returns a value of type Ticket. When this function is called, a new ticket will be created and displayed.

1. *+previewTicket()*

previewTicket() is a public function which returns a value of type Ticket. When this function is called, the ticket will be displayed based on the given ticket id.

**Dependencies with Other Classes and Packages:**

A screenshot of a cell phone

Description automatically generated

The Ticket class interacts with Event class and Ticket Storage package. Users can buy or preview tickets by accessing the Ticket class through the Event class, which is accessed through the Website class. Ticket class interacts with Event class because it is a product of an Event. Also, Ticket class interacts with the Ticket Storage package to store the tickets.

**Exceptions Raised and Exception Handling:**

There are several exceptions that can be raised by the Ticket class. For example, if a user tries to buy an amount of tickets that exceeds the quota of the event an exception is raised.

To handle this exception, an error message is displayed on the screen and the user is asked to decrease the quantity of the tickets.

Also, if a user tries to buy a ticket with an amount of credits less than the price of the tickets an exception is raised. To handle this exception, an error message is displayed on the screen and user is redirected to the home page.

* 1. **Stage Class** (Dilara)

The Stage class defines the type of the stage that is chosen. The Stage class is as follows:



**Class Name: Stage**

**Attributes:**

1. *-address: String*

address is a value of type String. It is a private attribute. It holds the address of the stage.

1. *-quota: Integer*

quota is a value of type positive Integer. It is a private attribute. It holds the maximum amount of people that the stage can have.

1. *-place: String*

place is a value of type String. It is a private attribute. It holds the city in which the stage is located.

**Dependencies with Other Classes and Packages:**

A screenshot of a cell phone

Description automatically generated

The Stage class interacts with Event class and Event Storage package. Events can only be created if it includes the stage where it will be held. So, every time an event is created Stage class interacts with Event class. Also, Stage class interacts with the Event Storage package through Event class to store stages.

**Exceptions Raised and Exception Handling:**

The Stage class won’t raise exceptions within self because it doesn’t have any functions that provide services to other classes or packages.

* 1. **Operator Class** (Dilara)

The Operator defines a user that can manage events. Also, it allows an admin to add a new operator and view the existing operators. The Operator class is as follows:

A screenshot of a cell phone

Description automatically generated

**Class Name: Operator**

**Operations:**

1. *+addOperator()*

addOperator() is a public function which returns a value of type Operator. When this function is called, a new user with operator permissions will be created.

1. *+viewOperators()*

viewOperators() is a public function which returns a value of type List. When this function is called, a list consisting of all the operators will be displayed.

**Dependencies with Other Classes and Packages:**

A screenshot of text

Description automatically generated

The Operator class interacts with User class and Website class. Operator is a type of user that can be created by an admin; therefore, it inherits the qualities of the User class by accessing it. The Operators class manage events, so it interacts with Website class to access events.

**Exceptions Raised and Exception Handling:**

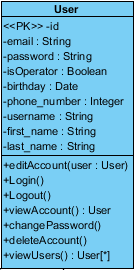
There are several exceptions that can be raised by the Operator class. For example, if an admin tries to add an operator leaving some of the information unprovided an exception is raised. To handle this exception, an error message is displayed on the screen and admin is asked to provide the necessary information.

Also, if an admin tries to add an operator with a password that contains the email or the username information an exception is raised. To handle this exception, an error message is displayed on the screen and admin is asked to change the password.

Another exception is raised if an admin tries to add an operator with a common password. To handle this exception, an error message is displayed on the screen and admin is asked to construct a stronger password.

**3.5 User Class (Özay)**

The user class allows a user to perform certain operations in order to run the system. A user can edit his/her account, view account, login and logout, change password, delete account. User class provides admin to see all users in the system. The User class is as follows:

****

**Class Name: User**

**Attributes:**

*a)-id:int*

Id is a user identification number of type positive integer. It is a private attribute. It is used in order to distinguish the user from other users.

*b)-email:String*

Email is one of the user’s information of type string and it is a private attribute. It is used for some tasks like when user forgot the password. He/she can get reset password link via his/her email. Email is an editable user attribute in edit profile section.

*c)-password:String*

Password is a user identification number of type string. It is a private attribute. It is used in order to authenticate the user with the system.

*d)-isOperator: Boolean*

isOperator is a value to identify user type of type boolean. It is a private attribute. It is used in order to verify user type when user authenticate and to limit user’s access to the functions.

*e)-birthday: Date*

Birthday is a date of type date. It is a private attribute. It is one of user’s information and used in user account.

*f)-phone\_number: Integer*

phone number is a value of type integer and indicate one of user’s information. It is a private attribute. It is used in user account.

*g)-username:String*

Username is a user identification number of type string. It is a private attribute. It is used in order to authenticate the user with the system.

*h)-first\_name:String*

First name is a user information of type string. It is private attribute. It is used in order to be used in ticket details and user account. First name is an editable user attribute in edit profile section.

*ı)-last\_name:String*

Last name is a user information of type string. It is private attribute. It is used in order to be used in ticket details and user account. Last name is an editable user attribute in edit profile section.

**Operations:**

*a)+editAccount(user:User)*

editAccount() is a public function which takes user as a parameter. When a user wants to edit information, he/she must use this function in the system. The editAccount() function saves the changes and edits database related information with this user.

*b)+login()*

login() is a public function which is used for authentication of user. When a user wants to enter the system, he/she must use this function. To access user related functions, first user is needed to be logged in.

*c)+logout()*

logout() is a public function which is used for finalize the user’s session. When a user wants to exit from the system. He/she must use this function to finalize his/her session.

*d)+viewAccount():User*

viewAccount() is public function which is used for showing user his/her profile. When a user wants to see his/her information. He/she must use this function.

*e)+changePassword()*

changePassword() is a public function that does not return a value. When this function is called, user’s password would be change according to his/her input for new password.

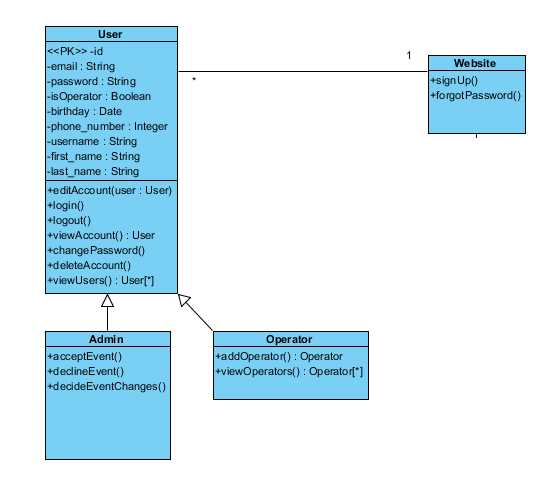
*f)+deleteAccount()*

changePassword() is a public function that does not return a value. When this function is called, user couldn’t be able to access the system through his/her account.

*g)+viewUsers():User[\*]*

viewUsers() is a public function which returns a list of user which system has. When this function is called by Admin, all users in the system is showed to the Admin.

**Dependencies with Other Classes and Packages:**

****

The User class interacts with Admin, Operator and Website class. User is a class that admin and operator type of users are inherited from this class and could be created in the system. The User class can login, logout, change password, edit account, view account, delete account and view users according the type of the user. Also, User class interact with website. Website provides users to sign up and to use forgot password function, furthermore website class is kind of user interface that makes users to access other classes’ functions which means the rest of the system.

**Exceptions Raised and Exception Handling:**

There are several exceptions that can be raised by the User class. For example, when user wants to login with username and password that doesn’t exist in user database, system display an error message to user that says username or password is not current.

When a user wants to edit his/her account and the information which is going to be changed may not be valid or wrong. So, User is notified by system says, ‘information that you’re trying to change is not valid’.

One of the exceptions may be the one when an user wants to change his/her password, if he/she enters new password which is not verified requirements of password, the system notifies the user by showing an error and says ‘entered password does not satisfy password requirements’.

**3.6 Admin Class(Özay)**

The Admin class allows admin to accept, decline and decide event changes created and sent by Operator. The Admin class is as follows:

****

**Attributes:**

This class does not contain any additional attribute since it’s inherited from User class and using its attributes. Also, according to user type this class can perform such functions which user class has.

**Operations:**

*a)+acceptEvent()*

acceptEvent() is a public functions which does not return any value. When this function is called, event which is created and sent by operator would be accepted and put into available event list.

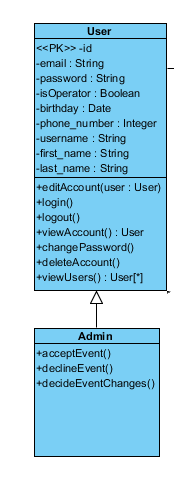
*b)+declineEvent()*

declineEvent() is a public function which does not return any value. When this function is called, event which is created and sent by operator would be declined.

*c)+decideEventChanges()*

decideEventChanges() is a public function which does not return any value. When this function is called, event changes created and sent by operator would be accepted or declined by Admin.

**Dependencies with Other Classes and Packages:**

****

The Admin class interacts User class. Admin is a one of user types that can be create operator for the system and inherited from user class. Admin class accepts, declines event and decides event changes created and sent by operator. To use the other classes’ functions admin class access these via User class.

**Exceptions Raised and Exception Handling:**

There are no exceptions for the functions that just admin class has. But, functions that come from User class because of inheritance have some errors and these errors and handling ways explained in the User Class section.

**3.7 Website Class(Özay)**

The Website class allows the visitors to sign up and use forgot password function. Also, provides users interface to perform their functions. The Website class is as follows:



**Attributes:**

This class does not contain any attribute since provides interface to users and some related functions. Also, this class provides interface to users and some related functions.

**Operations:**

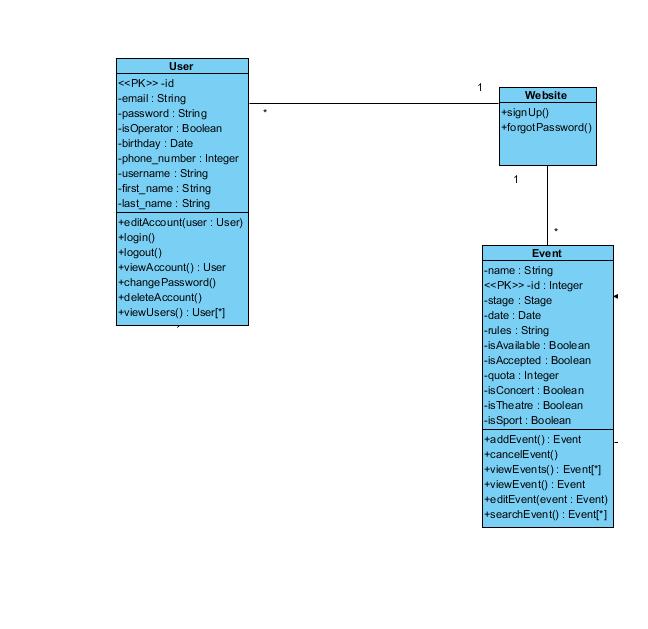
*a)+signUp()*

signUp() is a public function which does not return any value. When this function is called, user would be created and saved to user database with the information that user provided.

*b)+forgotPassword()*

forgotPassword() is a public function which doesn’t return any value. When this function is called, user is sent link via his/her email to create new password to enter the system.

**Dependencies with Other Classes and Packages:**

****

The Website Class interacts with User class mad Event class. Website is a class that provides users an interface to access with other classes and these classes’ functions. Also, Website class provides user to signup the system for the user who did not sign up the system and users who does not remember his/her password to use forgot password function to take a new password to enter the system.

**Exceptions Raised and Exception Handling:**

There are several exceptions that can be raised by the Website class. For example, when a user who did not sign up to system before wants to signup to system, there may be error such as information that user entered could be invalid and to handle this error system notifies user to enter valid information.

Another error example could be when user forgot his/her password and use forgot password function by entering invalid email, system notifies user and says ‘email is not valid’ with error message.