BILKENT UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CS 319 -- Summer 2010

Cagenda – Distributed Agenda System

Group 8

Final Report

Bilge Acun	Alper Baspinar	Ekin Oguz
20802371	20700777	20801142
Section 1	Section 1	Section 1
b_acun@ug	a_baspinar@ug	e_oguz@ug

Submission date: 20/07/2010

Table of Contents

Cagenda - Distributed Agenda System

1. Introduction

Cagenda is a Distributed Agenda System which can be used as a personal organizer. The need of a functional and user friendly is a natural consequence of today’s rush hour life. The purpose of the system is to provide a well interfaced and useful agenda system where users can organize their schedules as well seeing other user's schedule, inviting others to events by sending notifications to them. Cagenda offers a compact distributed system with excellent features. All wide range of people can use the system without limitation.

In this Final report, firstly problem statement will be explained to the reader. Then requirement analysis, analysis models, system design and object design will be followed in next sections. Report will finish with a conclusion which is about summary of what we explained and lessons that are learnt during the entire project.

2. Problem Statement

The intensity of the appointments, meetings and works in most people's life make the need of an agenda inevitable. However nowadays nobody wants to carry a notebook for agenda in their pockets. Everybody has computers or smart phones to check their schedules so the need of an agenda system which people can reach and use from the internet is increased.

Many companies such as Google, Microsoft work on this distributed agenda system. However, most of these agendas do not really differ from the old notebook agendas. They have same calendar functionalities with notebook agendas other than working on computer. These systems have some deficiencies, so they can be improved by adding some more functionality. We have an aim to solve these problems with a user-friendly, usable, powerful and fast application. Some functionalities such as inviting other people to an event or a meeting (shared events); deleting and updating this shared event by sending notifications to other people, notifying the user for coming events etc. should be included to the system.

3. Requirements Analysis

3.1 Overview:

Basically system provides a calendar and clock. Specifically, there are additional features such as adding, changing, deleting an event, inviting other users to an event and see other user's schedules. In order to support these functionalities, system should be distributed with usage of database.

3.2 Functional Requirements:

1 – User shall be able to create an account with desired password and username to log in the system.

2 – User shall add a new event.

2.1 – User shall be able to select the type of an event which is meeting, anniversary, birthday or to-do.

2.2 – User shall be able to add notifications to any type of events in desired date.

2.3 – User shall be able to set the privacy of events.

2.4 – User shall be able to set the date of the event.

2.5 – User shall be able to set the time of the event.

3 – User shall delete an existing event.

4 – User shall edit an existing event.

5 – System shall show the clock.

6 – System shall show the date in monthly view.

7 – User shall be able to see other user's schedule if the privacy settings of the other user's calendar is appropriate.

8 – User shall invite other users to an event.

9 – The system shall not allow creating an account with same user name.

10 – The system shall be able to show the date after BC to infinite.

11 – The system shall show the clock in this format: hh:mm

12 – The system clock shall show the clock GMT +2:00 Ankara.

13 – The system shall show the date in this format: dd / mm / yyyy.

14 – The system shall show Monday, Tuesday, Wednesday, Thursday, and Friday as workdays with pink color and Saturday and Sunday as weekend with purple color.

15- The system shall show the dates, which do not belong to the month in view, with gray color.

16 – The system shall show the private events with a busy symbol to other users.

3.3 Non-Functional Requirements:

1 – The system shall be able to respond between 1 and 5 seconds.

2 – The level of expertise of the user shall be intermediate.

3 – "Readme" document which explains the use of the system shall be provided to the user.

4 – The system shall store maximum 1GB data.

5 – The system shall be licensed to group Mavericks.

6 – The user shall download the program from web and shall not install the program.

7 – The user shall have JRE installed in his system in order to run the program.

8 – The user shall have at least 2GB processor, 2 GB Ram and 1024 x 766 screen resolutions.

9 – Data shall be exported / imported into the system via internet.

10 – The system shall have 10 concurrent users.

3.4 Constraints:

1 – The system must be implemented in Java.

2 – The system must use MySQL database.

3 – The system must be a distributed system.

4 – The system must be desktop application.

3.5 Scenarios

3.5.1 – First Usage of Cagenda

Emma who is a student at Bilkent University has a lot assignments and exams. Therefore she should manage her time efficiently and she decides to use Cagenda. She downloads the program and creates an account with user name and password then she logins. First, she adds an exam which is two months later and she wants Cagenda to give notification before two weeks to exam.

One week later, teacher postponed the exam to a later date. Thus, Emma opened Cagenda with her user name and password from University Lab and changed the exam date and notification time.

3.5.2 – Arrangement of a Meeting

Bilge that has a CS319 project needs to make an arrangement with her project team members. She opens Cagenda to see her friend's schedules in order to find appropriate time for everybody. She clicks on the search button and writes her friend's names. Their schedules are available and then Bilge opens their schedules. After deciding an appropriate time, she creates a new meeting in her Cagenda and then invites her friends to the meeting. Hoping that her friends will come to the meeting, Bilge closes Cagenda.

3.5.3 – Rescue of the Day

Mr. Anderson wakes up in the morning and goes to work then he wants to know what he should do during the day. He opens his laptop and runs our agenda program “Cagenda”. When he opens the program, first two notifications pop up and one says “Today is your wedding anniversary” and the other says “Morpheus (your boss) invited you to attend a meeting in conference hall at 03:00 pm”. Then Mr. Anderson clicks on the button to accept the invitation and this event is added to Mr. Anderson's Cagenda. But system gives a warning which says “There is a Basketball Game at the same time”. Because meeting with boss is more important than another event, Mr. Anderson deletes the other event and adds the meeting to Cagenda.

When Cagenda reminds the wedding anniversary, Mr. Anderson is shocked and thanks to Cagenda, possible divorce is avoided. Mr. Anderson decides to buy a present to his wife and take his wife to dinner. In order not to forget anniversary again, Mr. Anderson adds two new events to Cagenda which are “Shopping at 05:00 pm” and “Dinner with his dear wife at 08:00 pm”. He closes Cagenda

3.6 Use Case Models

3.6.1 Textual Use Case

Use Case Name: **Create Account**

Participating Actors: Cagenda User

Entry Condition: The Cagenda User downloads “Cagenda” and opens it to use.

Flow of Events:

1. The User sends a request to create new account to the system with password and name.

2. The system checks whether password and name are valid or not, then if they are valid, the system creates a new account and saves to the database, otherwise sends user a failure message.

Exceptions:

1a. If user enters an existing user name, he cannot create a new account.

2b. If internet connection fails, program will fail.

Exit Conditions: The new account is created or cannot be created.

***Use Case Name*: Manage Event**

Participating Actors: Cagenda User

Entry Condition: The Cagenda User opens the Cagenda and logins with his/her password and user name.

Flow of Events:

1. The Cagenda User adds a new event.

1.1 The user sets the time, privacy and date / time for the notification of the event.

2. The Cagenda User edits an event by changing the information of the event

3. The user deletes the event.

Exceptions:

1a. If internet connection fails, program will fail.

1.1a. If user forgets to set the notification, notification will be automatically settled to off.

Exit Conditions: The user finishes dealing with events.

***Use Case Name*: See Clock**

Participating Actors: The Cagenda User

Entry Conditions: The Cagenda User opens the Cagenda and logins with his/her password and user name.

Flow of events:

1. The user sees the clock from the right top of the program.

Exit Conditions: The user learns the clock and stops looking at.

***Use Case Name*: See Date**

Participating Actors: The Cagenda User

Entry Conditions: The Cagenda User opens the Cagenda and logins with his/her password and user name.

Flow of events:

1. The user sees the current date from the top of the program.

Exit Conditions: The user learns the date and stops looking at.

***Use Case Name*: Invite Other Users to Events**

Participating Actors: The Cagenda User

Entry Conditions: The Cagenda User opens the Cagenda and logins with his/her password and user name. There is an existing event which user wants to invite others by sending notification to them.

Flow of events:

1. The user chooses the event and then enters the people's name that he wants to invite to chosen event.

2. System sends notification to invited people.

Exceptions:

1a. If user enters a name which does not exist, system will give an error.

Exit Conditions: The user finishes inviting people to his/her event and all notifications are sent to desired users.

***Use Case Name*: See Other Users Schedules**

Participating Actors: The Cagenda User

Entry Conditions: The Cagenda User opens the Cagenda and logins with his/her password and user name.

Flow of events:

1. The user wants to see another user's schedule, and then he writes the name of the person who he wants to see.

2. If the name is valid, the system opens his/her schedule. Otherwise, user gets a warning message and the schedule cannot be opened.

Exceptions:

1a. If internet connection fails, program will fail.

Exit Conditions: The user finishes seeing others schedule.

3.6.2 Visual Use Case

In our system, there is one actor which is Cagenda User. Main use cases are create account, manage event, see date, see time, invite other users to events and see other users' schedule. All these use cases include login use case so Cagenda user needs to login to system firstly. Moreover manage event use case includes add event, delete event and edit event. Lastly, see other users' schedule includes control privacy settings. Visual Use Case diagram can be found in Figure #1.

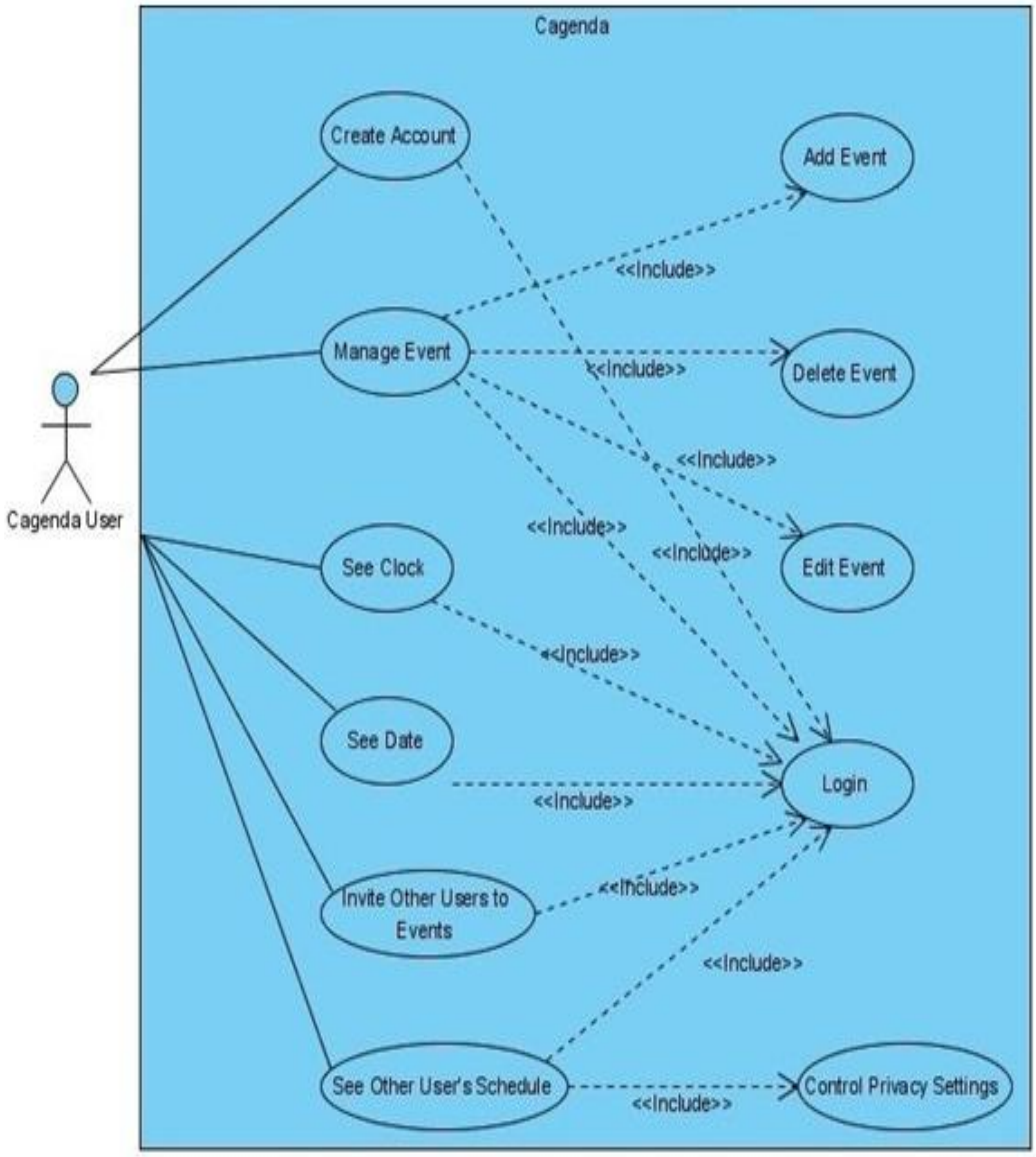


Figure 1 - Visual General Use Case for All System

3.7 User Interface

Figure #2 is the first screen of Cagenda. It is a simple login / create account screen. If user does not have an account, he can create an account with desired user name and password. If he has an account, he can simply write his information and then logins to the system. User cannot create an account with an existing user name.

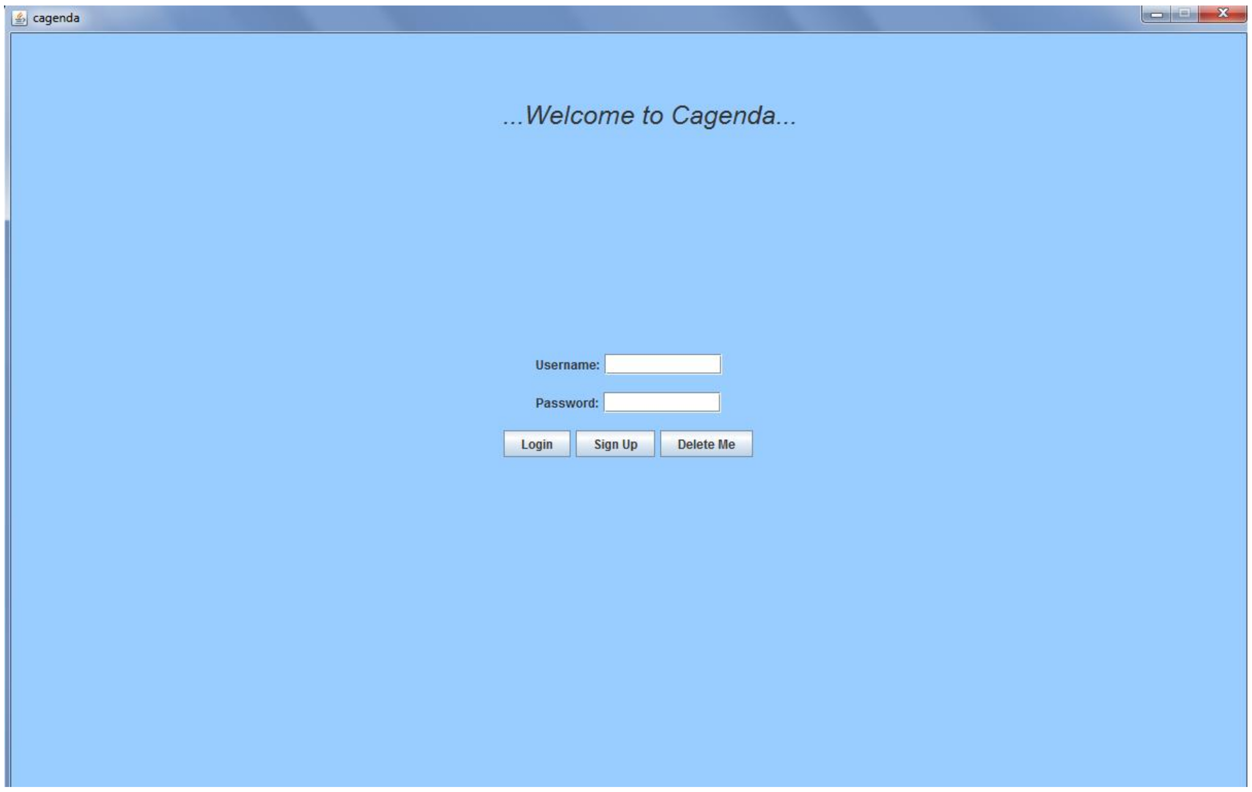
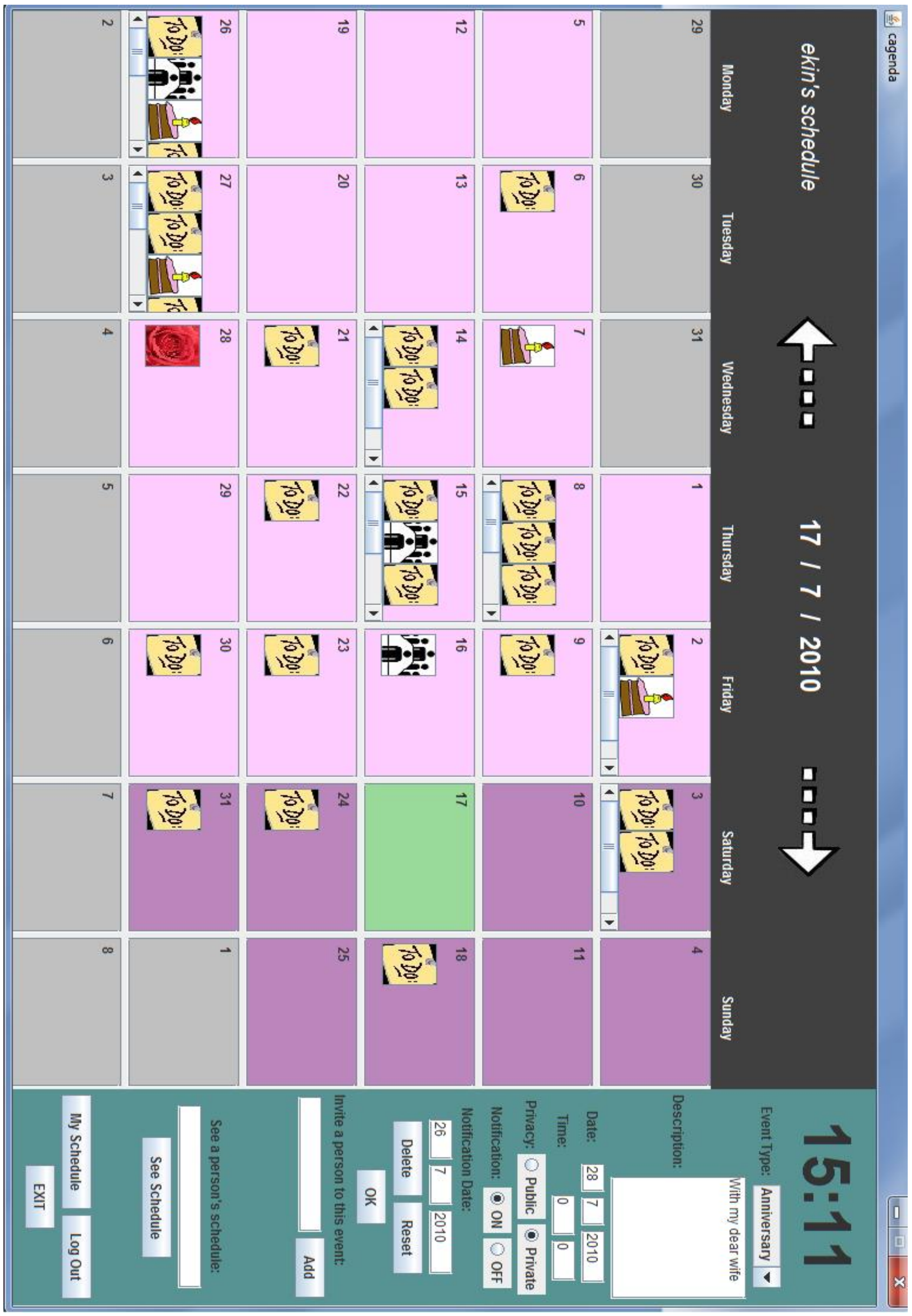


Figure 2 - User Interface for login / create account

Figure #3 is the general screen of Cagenda. User name will be written in upper left corner of the screen. Near the name, there will be current date and navigation buttons. With this buttons, user can change the date. Current date will be highlighted in date's panel. In date's panel, weekdays and weekends will be shown in different color. Moreover, the dates, which do not belong to the month in view, will be shown with gray color. In date's panel, user will see forty two days with the events that he added before. If he clicks on the any added event, details will be shown in right panel. If he clicks on any free date, he can add new event to his calendar. He can edit event information or delete any event in right panel. Moreover, with

"Invite Others" button, user can invite anybody to selected event. Furthermore, current clock will be shown in upper of right panel.



Event Type: Anniversary
Description: With my dear wife
Date: 28 7 2010
Time: 0 0
Privacy: ( ) Public [✓ Private]
Notification: [✓ ON] ( ) OFF
Notification Date: 26 7 2010
Invite a person to this event:
See a person's schedule:

Figure 3 - User Interface for General Screen of the Cagenda Program

In Figure #4, invitation screen is shown. This notification screen is sent by the user “ekin” to the current user. When current user logins to the Cagenda, he will receive invitations from other users if anyone has invited him. In this invitation screen, event type, date, time and description will be shown and there will be option for user to accept this invitation or ignore it. If user accepts, this event will automatically be added to this schedule. If he ignores, nothing will happen.

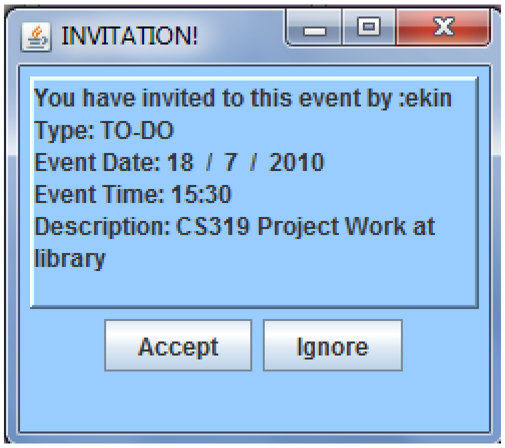

You have invited to this event by :ekin
Type: TO-DO
Event Date: 18 / 7 / 2010
Event Time: 15:30
Description: CS319 Project Work at library

Figure 4 - Invitation Screen

Figure #5 is the screens which will shows “See Schedule” function. In right panel, there will be text field to write user name. User will write the user name of the person that he wants to see his schedule and then clicks on “See Schedule” button. After that, other person's schedule will appear with his name in upper left corner. In this screen, user can only click on the event which has appropriate privacy settings. After clicking on the event, user will see its description but he cannot change or delete any event. Private events will be shown as an event with no other details to other users. If user wants to see his schedule again, he should use “My Schedule” button.

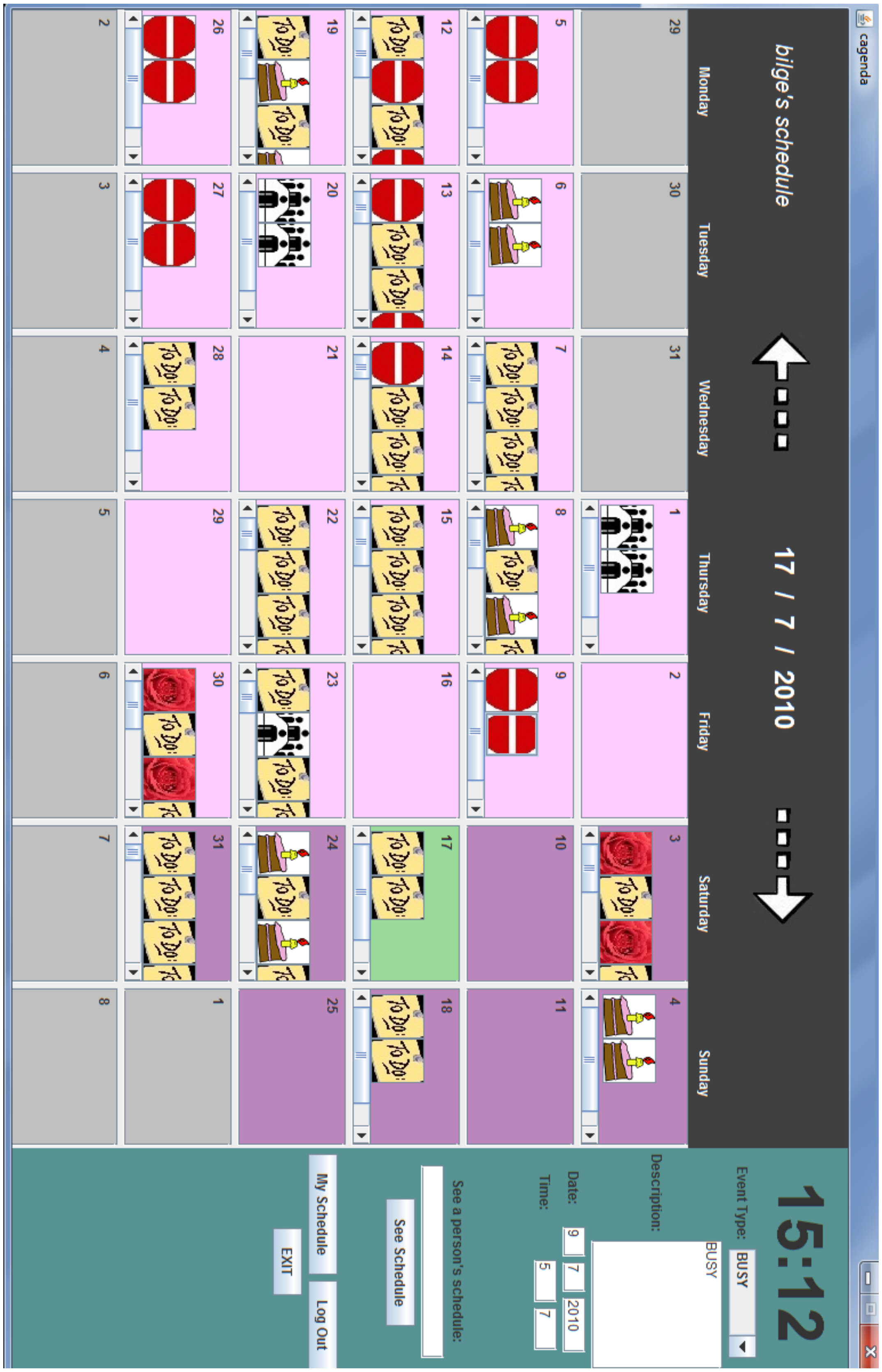


Figure 5 - User Interface for See Schedule of the Cagenda

4. Analysis Models

4.1 Object Model

4.1.1 Domain Lexicon

user: A person who interacts with the system.

account: Entity that is created by user with a password and user name and enables user to log-in the system.

username: Entity that is created by user in order to login to the system. Usernames should be specific for each user.

password: Composition of characters that are special to person. They are used with user names to access account. For security, they should not be less than six characters.

privacy: User's settings which prevents other users to see his/her schedule. These settings consist of public or private choices.

notification: A pop-up window which occurs at desired time to remind an event or summon to an event to user.

event: Entity which will happen at a given time and place and can be added to the system by user.

description: Summary of the event that is entered by user to the system in order to remind anything necessary about the event.

schedule: A composition of events including their description and other specifications which are shown on calendar.

event type: A kind of an event which can be anniversary, meeting, birthday and to-do. Beyond these kinds, user can select to-do as anything.

monthly view: View of the system which shows the whole month. Forty two days will be shown in monthly view.

4.1.2 Class Diagrams

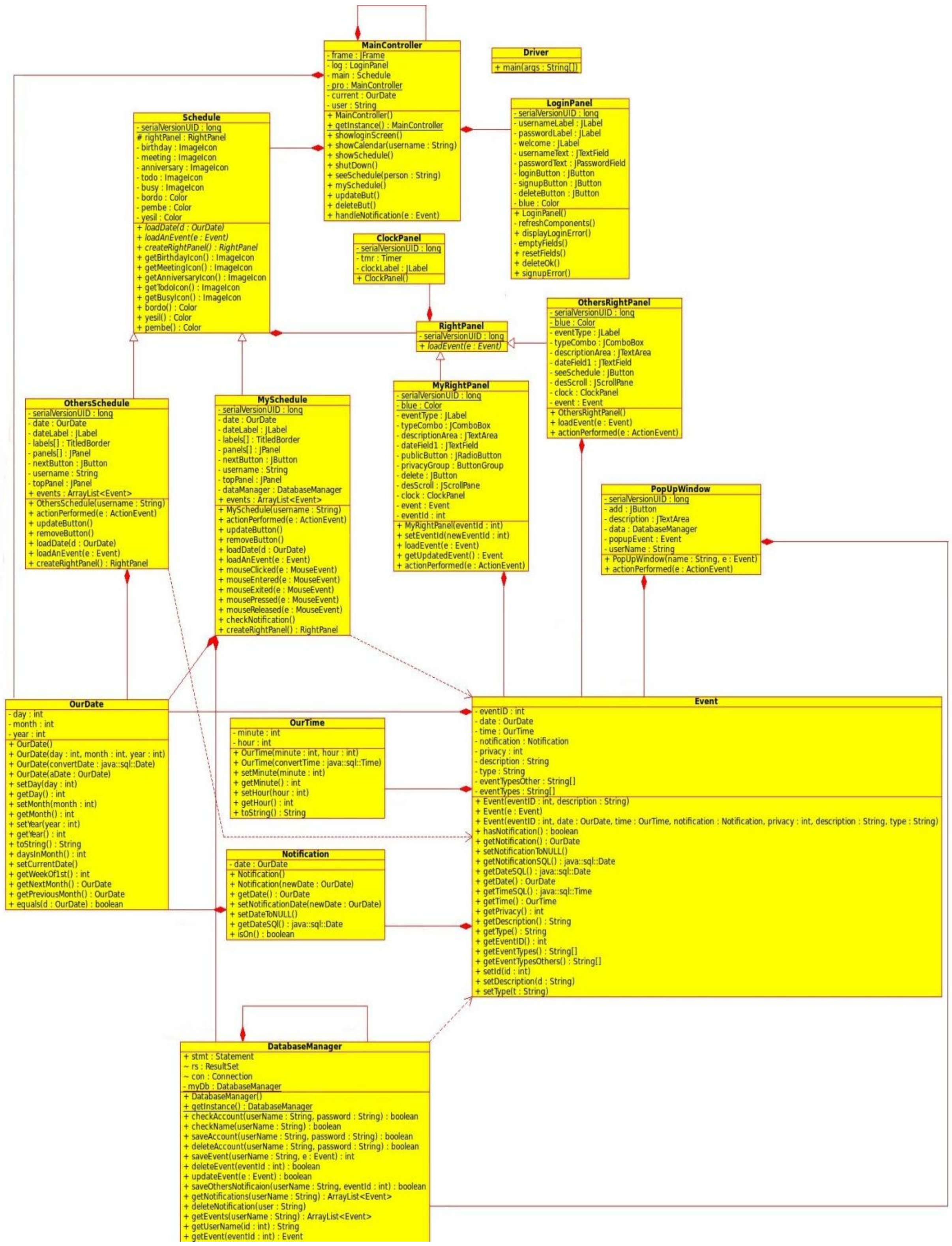


Figure 6 - Class Diagram for Whole System

4.2 Dynamic Models

4.2.1 State Chart Diagrams

<u>**Description of the State Chart Diagram for "A User Object":**</u>

This state diagram (Figure #7) shows the states of the "User" object. There are two states of the user: "Out of the System" and "Logged". The idle state is being out of the system, and then by logging in, user goes to "Logged" state. In that state, user can return to the other state by logging out.

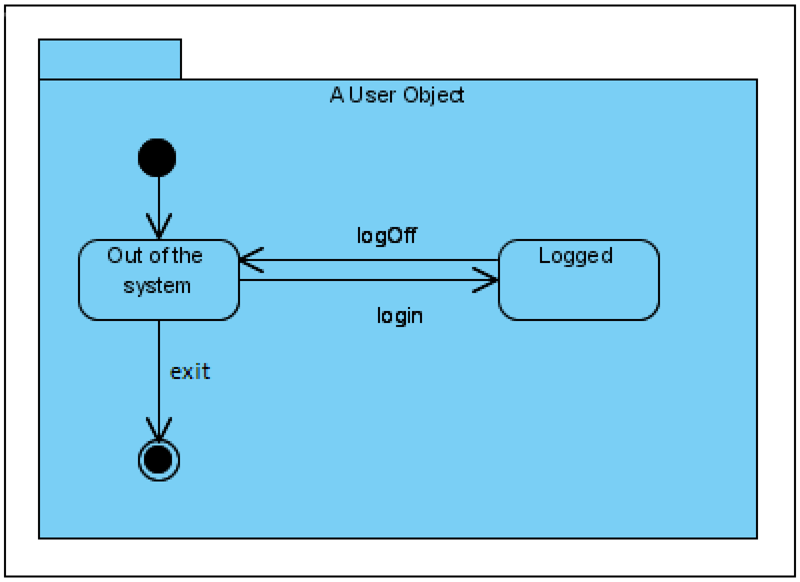


Figure 7 - State Chart Diagram for "A User Object"

<u>Description of the State Chart Diagram for “An Event Object”:</u>

This state diagram (Figure #8) shows the states of the "Event" object. There are five main states: "Created", "Changed", "People Invited", "Notification Added" and "Deleted". The idle state is "Created", here event is created with time, date etc. From this state, event can go to "Changed", "People Invited" or "Notification Added" state by doing the actions change event, invite people, add notification respectively. From all the states of "Created", "Changed", "People Invited" and "Notification Added", event can go to the "Deleted" stage with the user's request of deleting the event, so this state shown out of the package. From "Changed" state, event can go to "People Invited" or "Notification Added" states again by user's request of inviting people or adding notification. From "Notification Added" state event can go to "Changed" or "People Invited" states and finally from "People Invited" stage, event can go to "Changed" or "Notification Added" state by user's requests. Actually these states are not totally independent from each other. An event can be in two states at the same time that is an event can be both "People Invited" and "Notification Added".

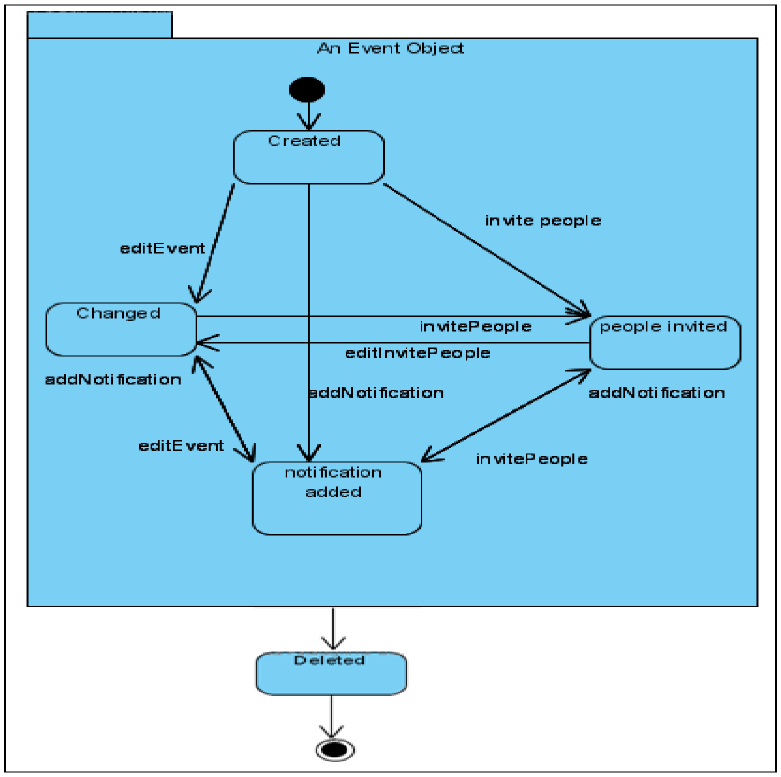


Figure 8 - State Chart Diagram for “An Event Object”

4.2.2 Sequence Diagrams

In Figure #9, sequence diagram for scenario “First Usage of Cagenda” is shown. In this diagram user object demands to create a new account which a username and password. To perform this system sends a saveAccount request to the database. Result of the response of the database verifies whether a new account created or not. After an account successfully created, same user tries to log in the system with her username and password. After system checks username and password by database information it enables user to benefit from the program. After User enters the program, she adds an event and notification to this event and makes some changes with this event later.

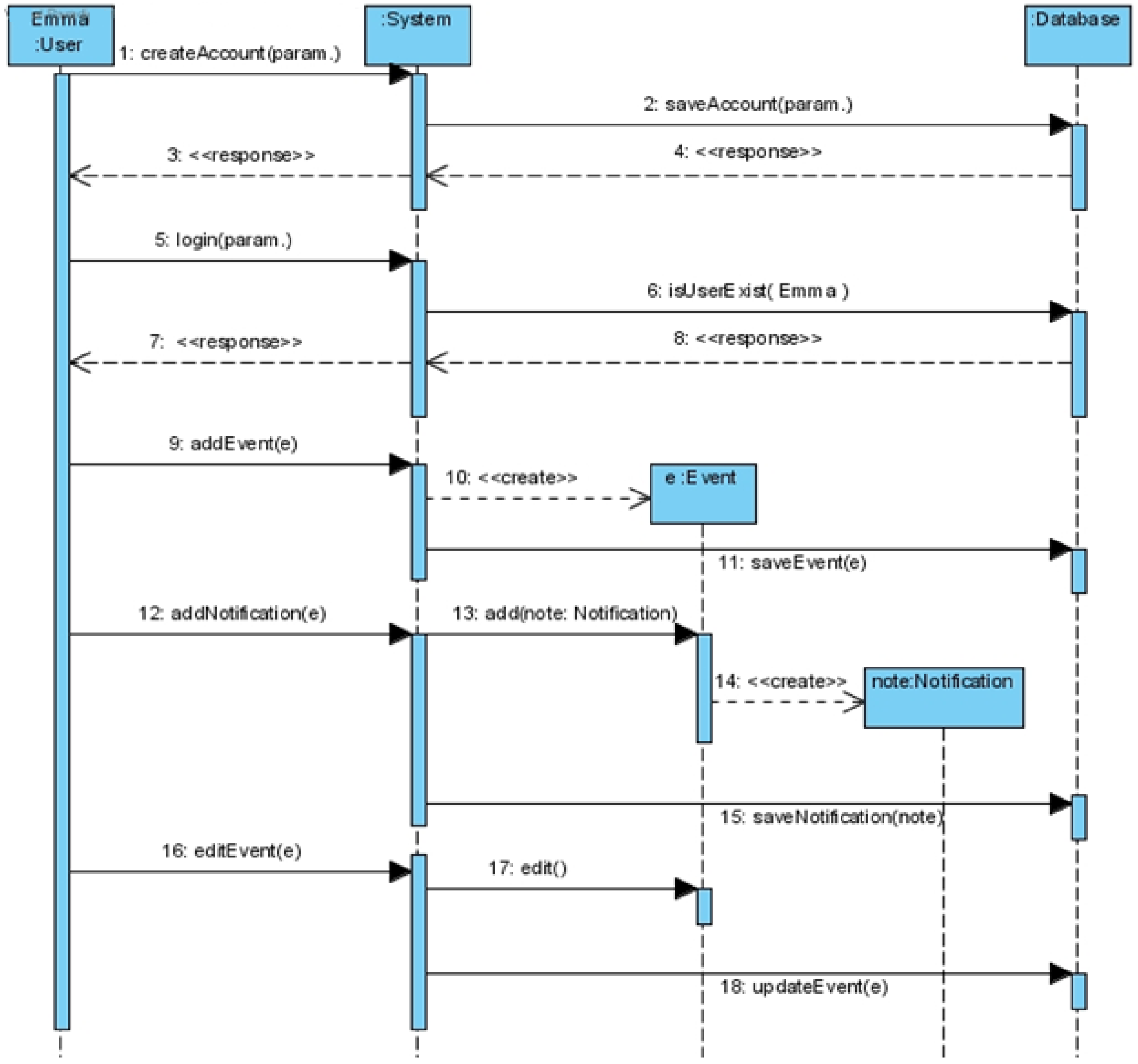


Figure 9 - Sequence Diagram for Scenario “First Usage of the Cagenda”

In Figure #10, sequence diagram for scenario “Arrangement of a Meeting” is shown. In this diagram, firstly user, Bilge, logins to the system. Then System calls the method isUserExist(Bilge) from Database. And next Database returns response to the System then it returns response to the user. Afterwards, seeOtherSchedule() methods is called. System checks privacy settings from database and according to result; other schedule is loaded to the screen. Afterward, inviteOthers() method is called with creating a notification. Database saves invited people from inviteOthers(). Lastly, addEvent() is called and this method creates a new event and this new event is saved into Database.

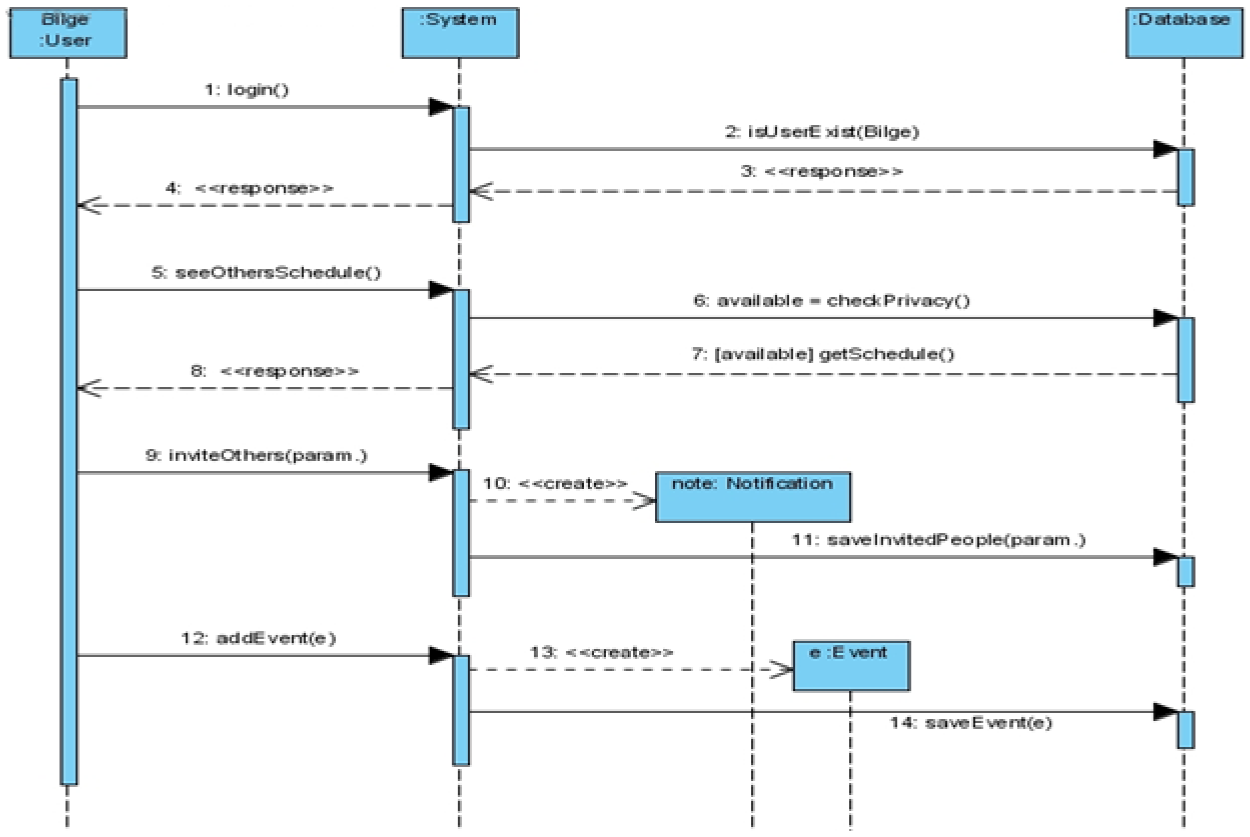


Figure 10 - Sequence Diagram for Scenario “Arrangement of a Meeting”

Figure #11 shows the diagram of the scenario “Rescue of a Day”. In this diagram, we can see that a user object sends a request to the system to login. After s/he successfully entered the program, system of the Cagenda automatically checks whether user has a notification or not and displays to the user. Later, same user invites some other users to his meeting by sending notification to them. Furthermore, he deletes an existing event and adds a new event by sending necessary commands to the system.

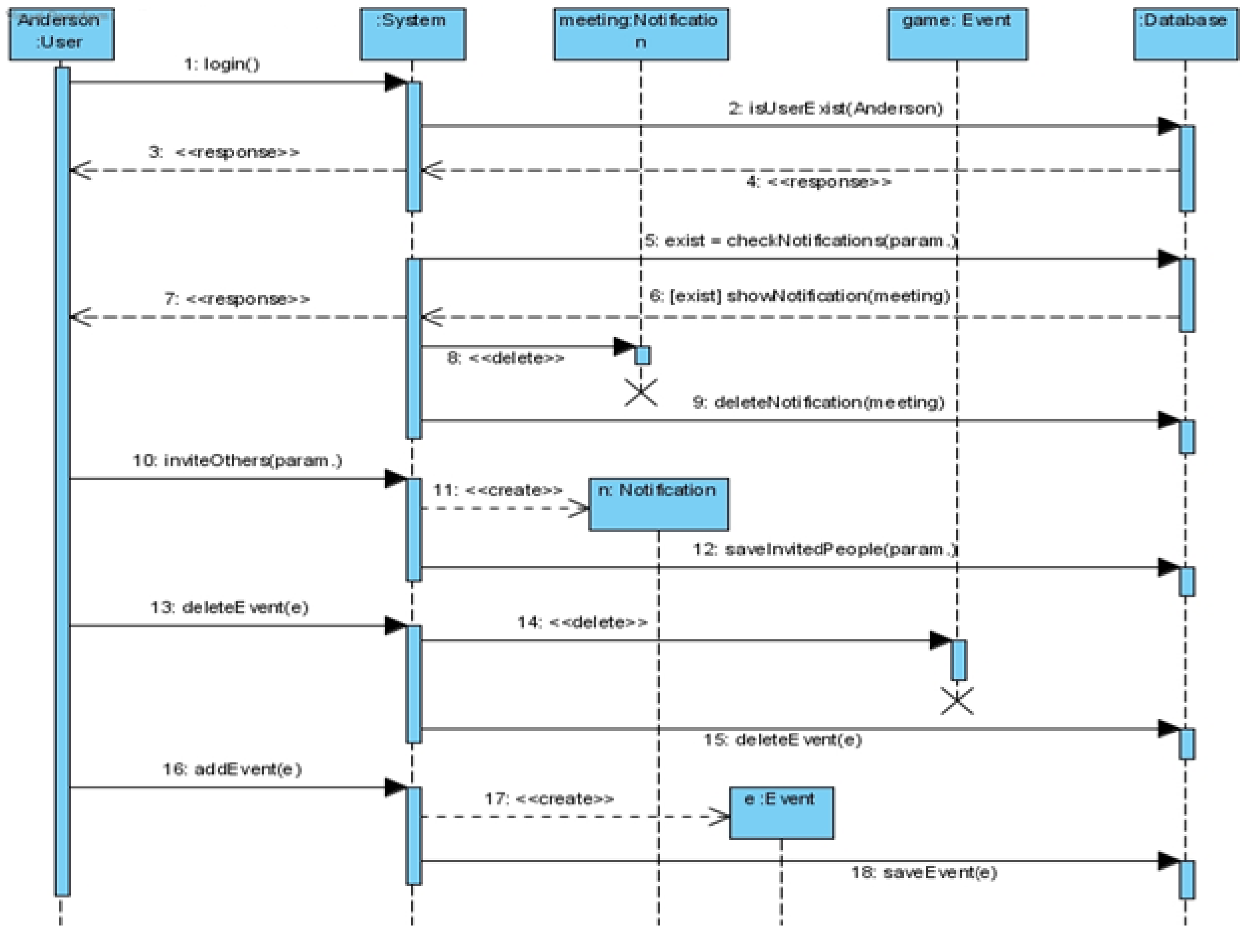


Figure 11 - Sequence Diagram for Scenario “Rescue of a Day”

5. System Design

In system design part, we explained our design goals and the architecture with the desired patterns.

5.1. Design Goals

We, as developers of the project, have some design goals to reach during the design part of the project and later in implementation part. In general, our main goals are low coupling and high cohesion. These two main design ideas are very important for all other sub goals. Moreover good documentation is very important for later progress. In order to update the project according to the given feedback, we have to know all information and we have to access everything easily. To achieve that, documentation is very important to remind the things we have forgotten. Briefly, clean, easily understandable documentation is very important for the improvement of the project. Our design goals are:

Reliability: Project should be reliable which means system should perform the expected actions and give expected results to the user during execution.

Reusability: We have a core project which can be improved with new functionalities later. In order to implement new functions, system should be reusable. Our design should be improvable for new functions with slight modifications.

Portability: Project should be portable for wide spread usage. Our system is an internet based project which will have wide spread usage. In order to distribute the project easily project should be portable. Also, user should easily use the program in any environment without spending time to installation.

Fault tolerance: Faults are the natural result of human factor. Of course there will be errors in project; so dealing with errors is very important. Program should be fault tolerance to deal with unexpected situations. If errors arise, user should not be disturbed, everything should continue as normally and system should cope with error without attracting the user. So there should be powerful fault handle system and project should be fault tolerance.

High-performance: Time is very important in daily life and therefore processes should not take much time. High performance is very important for in order not to lose users.

Easy interaction with user: User will spend long time with the program because he will always need to update his schedule and also check his current schedule. Therefore, user interface should be friendly for easy usage, and user shall not have any difficulty with program. He could easily do anything he wants. Moreover, user interface should be sympathetic to attract the user.

5.2. Sub-System Decomposition

Sub-system decomposition consists of separating the system into smaller parts. Each subsystem should focus on single concern. In this process, separation of concerns which are low coupling and high cohesion is very important. With these two principles, each subsystem should be independent of each other and each subsystem should have a single task. In this section, top-level decomposition of the system consisting of subsystems and package diagram will be explained.

We have three subsystems which are GUI, ApplicationLogic and Database. Each subsystem includes classes which are independent of each other. With this, low coupling principle is achieved. In GUI subsystem, classes that are related to user interface are added. ApplicationLogic includes classes which have the main functionalities and Database includes DatabaseManager. Each subsystem has a single concern to achieve high cohesion principle. Figure #12 shows the diagram of Subsystem Decomposition.

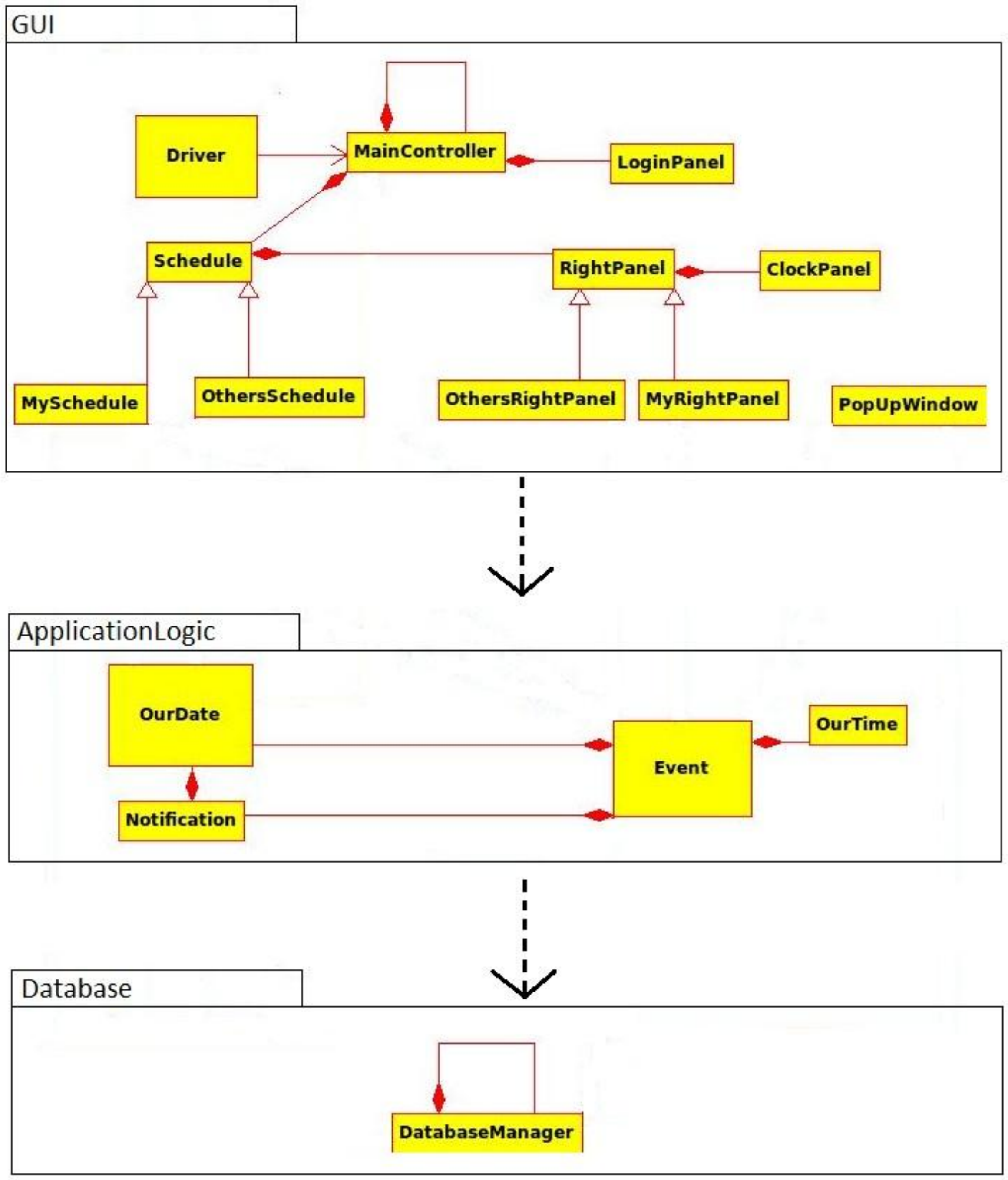


Figure 12 - Subsystem Decomposition Diagram

5.3. Architectural Patterns

We applied three Architectural Patterns, which are Client-Server, Layers and Model-View-Controller, to further structure the subsystems.

5.3.1. Client Server Pattern with Repository

In our system, there is a common WebServer for all clients. This WebServer has a MySQL Database which is used for storing all the information. Client is the requester and WebServer is the provider. Client knows the services of server and with these services, server performs some operations such as creating an event, saving a new event, deleting event or adding notification to the event and return results to the client. The system may have multiple clients but there is only one web server which they connect and use the MYSQL database.

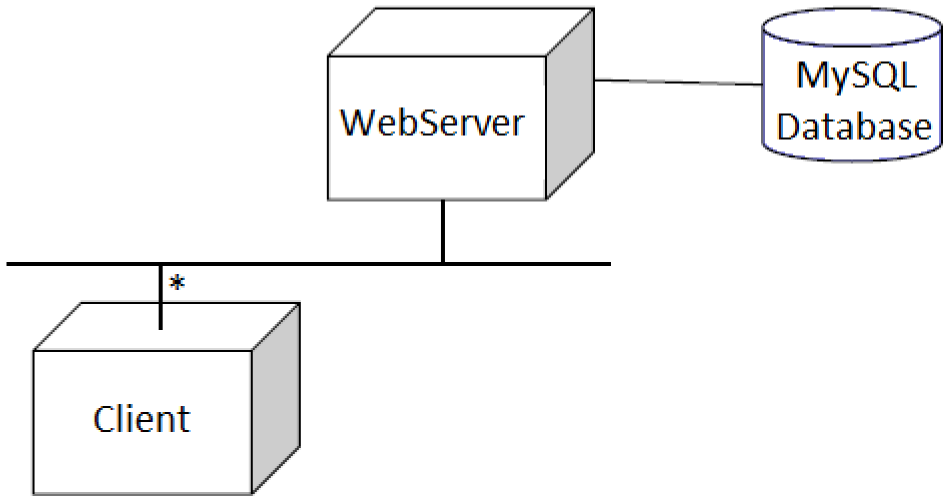


Figure 13 - Client-Server Pattern Deployment Diagram

5.3.2. Layers Pattern

Large Systems consist of many low and high level issues. In these systems high level issues depend on low level issues. In our system, we have three layers which are Presentation, Application Logic and Database. Application Logic is allowed to use Database. It uses Database to get all the information for user and store all the information that is entered by the user.

Presentation Layer does not know about Database, it uses ApplicationLogic to show necessary information to the user with attractive user interface.

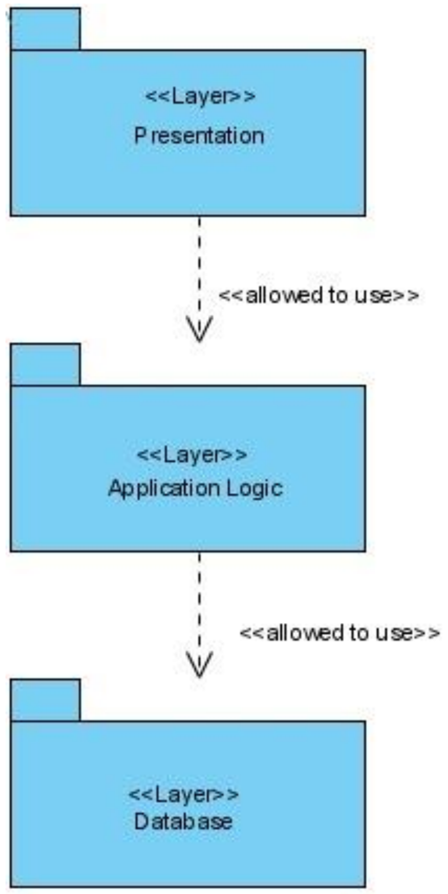


Figure 14 - Layers Pattern Diagram

5.3.3. Model View Controller Pattern

MVC Pattern is appropriate for interactive systems. In interactive systems, the interaction between user and program is very important. Also user interface needs to be changed easily if functionality changes. New functionalities should be added to system easily. Since main purpose of our Distributed Agenda System includes easy, sympathetic and usable interaction with user, we applied MVC Pattern to our system.

Our agenda program has MVC pattern for events to be displayed. While using this pattern, we take into consideration that “Event” class should be our model and “MainController” class should be the controller for the “MySchedule” and “OthersSchedule” view classes. In our program, there is an important property which is seeing schedule of other users. When a user wants to see other’s schedule, s/he simply clicks on the necessary button; then all view of the program will be updated according to information of the other users. To do this, there is an abstract class called “Schedule” in our program. This class arranges a template for “MySchedule” and “OthersSchedule” classes. “MainController” class takes required core information from “Event” class and uses this information to build new views using the view templates “MySchedule” and “OthersSchedule” classes. Figure15 shows the relationship between classes in MVC pattern.

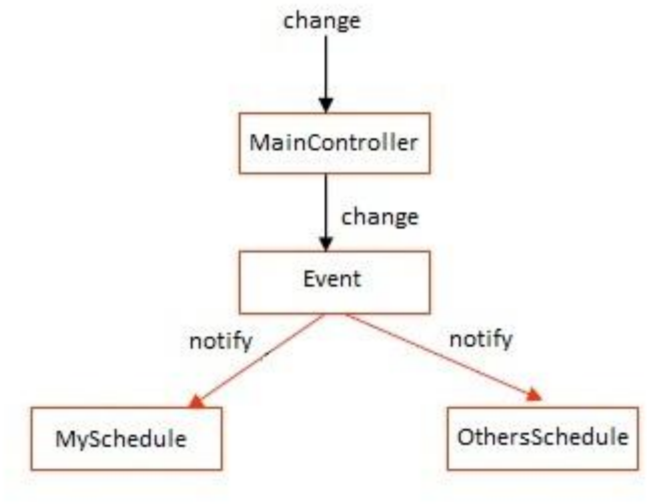


Figure 15 - MVC Pattern for our program

5.4. Hardware/Software Mapping

The programming language that we used for implementing our agenda project is JAVA. A user can run our application with any computer which has Java Runtime Environment (JRE). We have a web server which has a MYSQL database so; many users can use our program at the same time. Software can be executed with a single executable file. Deployment diagram of the system can be viewed below.

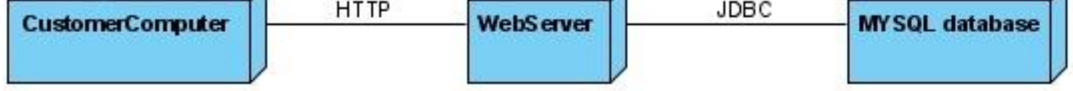


Figure 16 - Deployment Diagram of the System

5.5. Persistent Data Management

Our project needs to have a non-volatile memory because an agenda system requires information to be protected after execution of the program completed. We have chosen a distributed MYSQL database system to keep user files. Therefore, we are not bounded with local host of a computer and now any user can use our program if he has an internet connection. We have three database tables in our database. These are “accounts, events, and notifications”. Figure #17 has the general structure of the database. Accounts table has “username and password” columns and it is used while creating a new account and log in the system. Figure #18 shows the structure of accounts table. Events table has “eventid, username, eventtype, description, eventdate, eventtime, privacy and notification” columns. Figure #19 demonstrates events table. Notifications table has “calledUser and eventid” columns. Figure #20 has the details of the notification table. Eventid column in events table is same with eventid column in notification. We use eventid as primary key of the table so; each event has its own event id. A class named “DatabaseManager” has necessary SQL codes to manipulate database tables. Details of the database and tables inside of it can be viewed below.

Server IP: http://db.ahmetalpbalkan.com/

Database Name: alperb

Password: 123456

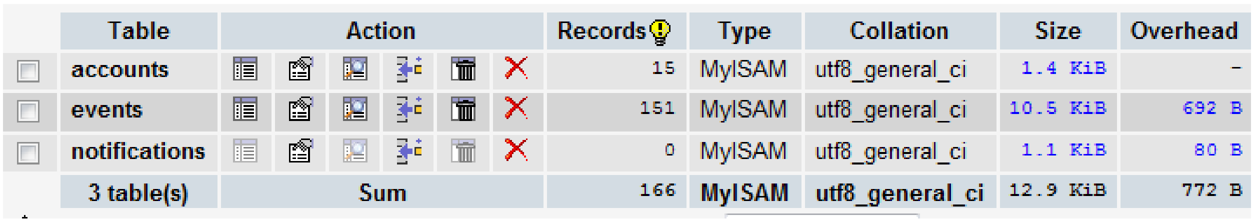
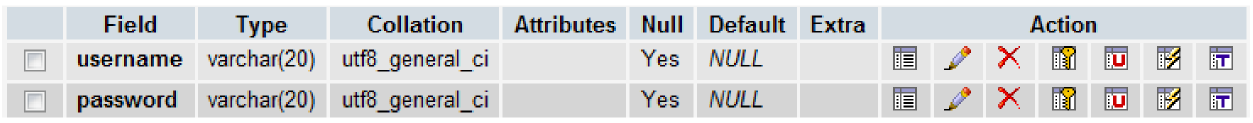


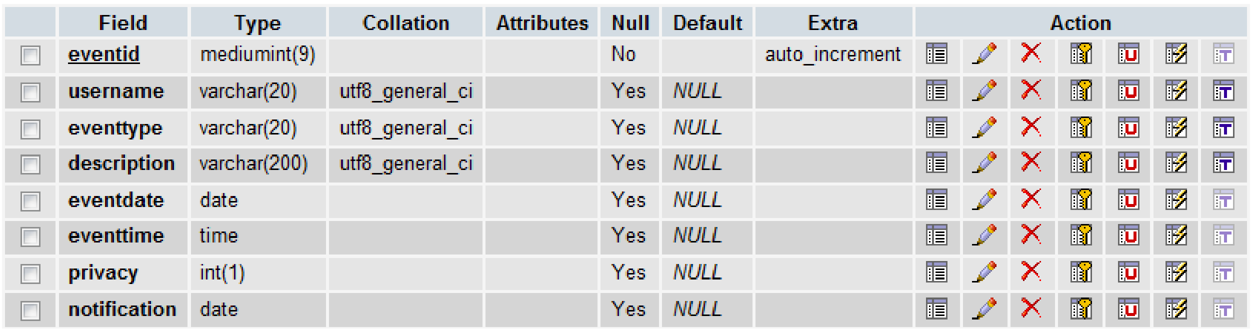
	Table	Action	Records	Type	Collation	Size	Overhead
☐	accounts		15	MyISAM	utf8_general_ci	1.4 KiB	-
☐	events		151	MyISAM	utf8_general_ci	10.5 KiB	692 B
☐	notifications		0	MyISAM	utf8_general_ci	1.1 KiB	80 B
	3 table(s)	Sum	166	MyISAM	utf8_general_ci	12.9 KiB	772 B

Figure 17: alperb Database General Structure



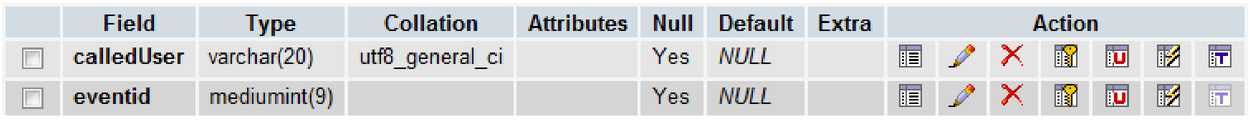
	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	**username**	varchar(20)	utf8_general_ci		Yes	*NULL*		
☐	**password**	varchar(20)	utf8_general_ci		Yes	*NULL*		

Figure 18: Accounts Table of the Database



	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	**eventid**	mediumint(9)			No		auto_increment	
☐	**username**	varchar(20)	utf8_general_ci		Yes	*NULL*		
☐	**eventtype**	varchar(20)	utf8_general_ci		Yes	*NULL*		
☐	**description**	varchar(200)	utf8_general_ci		Yes	*NULL*		
☐	**eventdate**	date			Yes	*NULL*		
☐	**eventtime**	time			Yes	*NULL*		
☐	**privacy**	int(1)			Yes	*NULL*		
☐	**notification**	date			Yes	*NULL*		

Figure 19: Events Table of the Database



	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	**calledUser**	varchar(20)	utf8_general_ci		Yes	*NULL*		
☐	**eventid**	mediumint(9)			Yes	*NULL*		

Figure 20: Notifications Table of the Database

5.6. Access Control and Security

Access control consists of different functionality and data which can be accessed by different actors. Since our project has only one actor which is the Cagenda user, there is no object guard system. If user can login to the system by correct user name and password, he has an access to add, edit or delete an event. Other users' schedule can be seen but events cannot be added, edited or deleted. Moreover, if events are private, they are seen as "busy event" with no detail.

6. Object Design

In object design part, we explained the design patterns that we applied and class interfaces which describe classes.

6.1. Design Patterns

We applied four Design Patterns, which are Singleton, State, Factory Method and Façade.

6.1.1. Singleton Pattern

Singleton pattern is a creational design pattern which is used to ensure that a class to has only one instance, this provides a global access to the instance. In this project, we used this pattern twice, one for “DatabaseManager” class and the other for “MainController” class.

1) “DatabaseManager” manages the database connection with the project's other classes by providing a set of methods to be used in the other classes. This database connection does not need to be created in every usage; it can connect only once otherwise it would take more time. Therefore, we should have only one instance of “DatabaseManager” class. We achieved this by applying Singleton Pattern here.

The code of the pattern:

```
private static DatabaseManager myDb; //the unique instance
public static DatabaseManager getInstance(){
     if (myDb == null)
          myDb = new DatabaseManager();
     return myDb; }
```

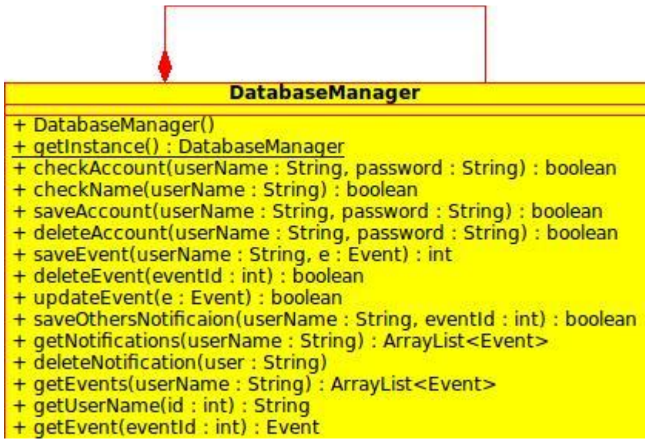

DatabaseManager
+ DatabaseManager()
+ getInstance() : DatabaseManager
+ checkAccount(userName : String, password : String) : boolean
+ checkName(userName : String) : boolean
+ saveAccount(userName : String, password : String) : boolean
+ deleteAccount(userName : String, password : String) : boolean
+ saveEvent(userName : String, e : Event) : int
+ deleteEvent(eventId : int) : boolean
+ updateEvent(e : Event) : boolean
+ saveOthersNotificaion(userName : String, eventId : int) : boolean
+ getNotifications(userName : String) : ArrayList<Event>
+ deleteNotification(user : String)
+ getEvents(userName : String) : ArrayList<Event>
+ getUserName(id : int) : String
+ getEvent(eventId : int) : Event

Figure 21 - Singleton Pattern of DatabaseManager

2) “MainController” class is the main controller class of the project which deals with controlling both GUI components and run-time behaviors so that it is needed to be accessed in most of the other classes in a global way with only one instance and we should ensure that it has only one instance so that we applied Singleton Pattern here.

The code of the pattern:

```
private static MainController project; //the unique instance
public static MainController getInstance(){
     if ( project == null)
          project = new MainController();
     return project;
}
```

When getting an instance of this class; the instance should be created with getInstance() method, the "new" operation should not be used.

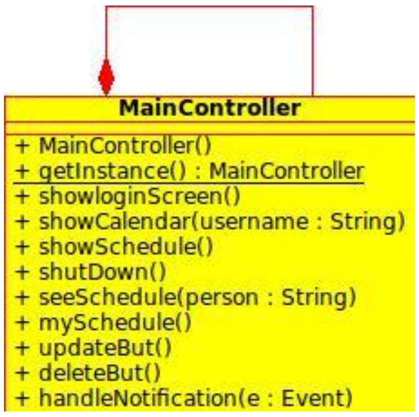

MainController
+ MainController()
+ getInstance() : MainController
+ showloginScreen()
+ showCalendar(username : String)
+ showSchedule()
+ shutDown()
+ seeSchedule(person : String)
+ mySchedule()
+ updateBut()
+ deleteBut()
+ handleNotification(e : Event)

Figure 22 - Singleton Pattern of MainController

6.1.2. State Pattern

State Pattern is a behavioral design pattern which is used when object's behavior depends on its state; when object's state has changed at run-time, its behavior should also change at that time.

We applied this pattern in our project, because the user is able to see both his schedule and others schedule, where “Schedule” class changes it's behavior when user wants to look at another person's schedule, or then when returning to his/her own schedule. Our project has a

'Schedule' class which deals with showing all a person's schedule, i.e. events. There are two different conditions; user sees his/her own schedule and user see another person's schedule. These are different because for example; user cannot add, change or delete and event in other people's schedule. Another example is that, if the events are private no detail about the event will show to the user and the event's icon should be busy icon. Dealing with these two different conditions, we have “MySchedule” and “OthersSchedule” classes which extend “Schedule” super class. “Schedule” class is an abstract class with two abstract methods:

```
public abstract void loadDate(OurDate d);
public abstract void loadAnEvent(Event e);
```

“MySchedule” and “OthersSchedule” classes implement these abstract methods in a different way. 'MainController' class has an instance of 'Schedule' class and it uses calls these abstract methods in a same way for both “MySchedule” and “OthersSchedule” classes. For example;

```
private Schedule s;
private OurDate current;
…
public void showSchedule(){
     s.loadDate(current);
}
```

If the 'See Schedule' button is pressed, this showSchedule() method of “MainController” is called; and then if “My Schedule” button is pressed, again this showSchedule() method of “MainController” is called. Same method is used but actually they make different things depending on the state of the “Schedule” object, depending on the implementation of the loadDate(OurDate d) method in sub-classes.

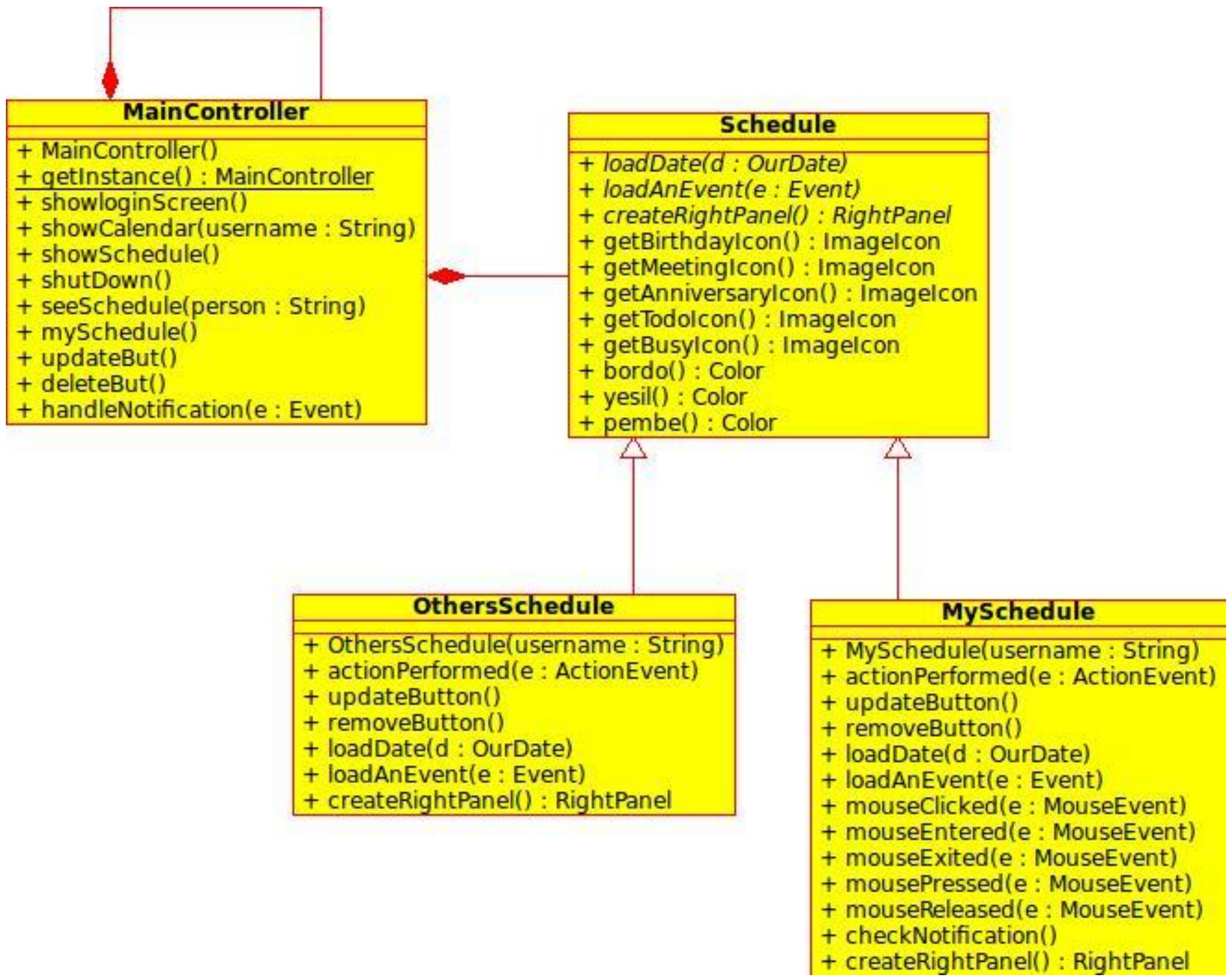

MainController
+ MainController()
+ getInstance() : MainController
+ showloginScreen()
+ showCalendar(username : String)
+ showSchedule()
+ shutDown()
+ seeSchedule(person : String)
+ mySchedule()
+ updateBut()
+ deleteBut()
+ handleNotification(e : Event)
Schedule
+ loadDate(d : OurDate)
+ loadAnEvent(e : Event)
+ createRightPanel() : RightPanel
+ getBirthdayIcon() : ImageIcon
+ getMeetingIcon() : ImageIcon
+ getAnniversaryIcon() : ImageIcon
+ getTodoIcon() : ImageIcon
+ getBusyIcon() : ImageIcon
+ bordo() : Color
+ yesil() : Color
+ pembe() : Color
OthersSchedule
+ OthersSchedule(username : String)
+ actionPerformed(e : ActionEvent)
+ updateButton()
+ removeButton()
+ loadDate(d : OurDate)
+ loadAnEvent(e : Event)
+ createRightPanel() : RightPanel
MySchedule
+ MySchedule(username : String)
+ actionPerformed(e : ActionEvent)
+ updateButton()
+ removeButton()
+ loadDate(d : OurDate)
+ loadAnEvent(e : Event)
+ mouseClicked(e : MouseEvent)
+ mouseEntered(e : MouseEvent)
+ mouseExited(e : MouseEvent)
+ mousePressed(e : MouseEvent)
+ mouseReleased(e : MouseEvent)
+ checkNotification()
+ createRightPanel() : RightPanel

Figure 23 - State Pattern Diagram

6.1.3. Factory Method Pattern

Factory Method is a creational design pattern which defines a model for creating an object where at creation time the super class can let its subclasses decide which class to instantiate. This method is all about the creation of an object, sub classes may want to create different classes' instances, and this pattern provides this functionality.

In the below Figure #24, the “ConcreteCreator” class, which is a sub class of “Creator” class, creates it's 'Product' which is a subclass of “ConcreteProduct” with the factoryMethod().

We applied this pattern in our project because the sub classes of the “Schedule” classes need to create the different instances of “RightPanel” super class. As it is explained in the previous subsection "State Pattern", “Schedule” class which deals with showing all a person's schedule, i.e. events. There are two different conditions, two different subclasses

“MySchedule” and “OthersSchedule”; dealing with users sees operation of his/her own schedule and another person's schedule respectively. These are different because for example; user cannot add, change or delete and event in other people's schedule. Moreover these subclasses create their own GUI components differently. For example; compare the Figure #3 and Figure #5, in the right side of the screen the panel components and their functionalities are slightly different as it can be seen clearly. In order to achieve this, ”MySchedule” class should create an instance of “MyRightPanel” and ”OthersSchedule” class creates an instance of “OthersRightPanel” class where “MyRightPanel” and “OthersRightPanel” are the sub classes of the “RightPanel” class.

In our usage of the pattern the factory method is createRightPanel() which is an abstract method in super class “Schedule”. This factory method is overridden in sub classes “OthersSchedule” and “MySchedule”, implementation example is given below. “Schedule” class's property “rightPanel” is created with our factory method createRightPanel(), not with "new" operation, or hard coded conditional statements.

The code example in super class Schedule:

```
//abstract FACTORY METHOD and RightPanel as property
protected RightPanel rightPanel = createRightPanel();
public abstract RightPanel createRightPanel(); //FACTORY METHOD
```

The implemantation of the factory method createRightPanel() in “OthersSchedule” class:

```
//FACTORY METHOD
@Override
public RightPanel createRightPanel() {
     return new OthersRightPanel();
}
```

The implemantation of the factory method createRightPanel() in “MySchedule” class:

```
//FACTORY METHOD
@Override
public RightPanel createRightPanel() {
     return new MyRightPanel(0);
}
```

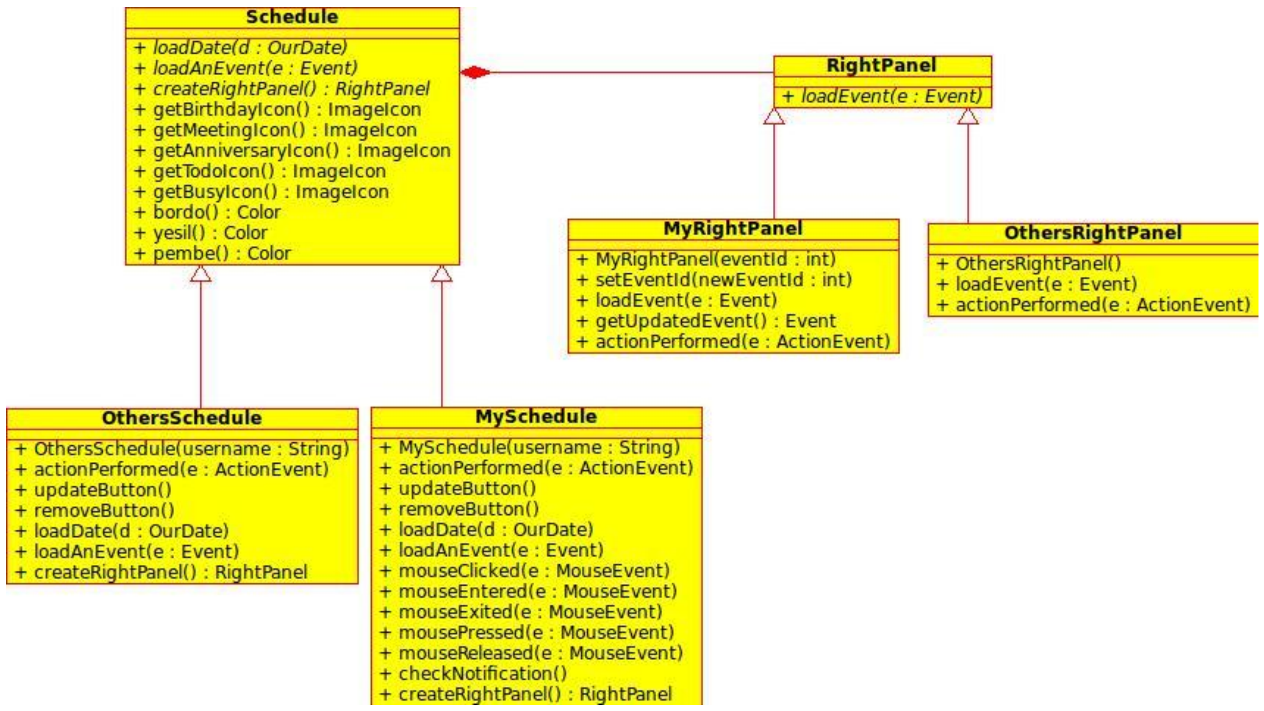

Schedule
+ loadDate(d : OurDate)
+ loadAnEvent(e : Event)
+ createRightPanel() : RightPanel
+ getBirthdayIcon() : ImageIcon
+ getMeetingIcon() : ImageIcon
+ getAnniversaryIcon() : ImageIcon
+ getTodoIcon() : ImageIcon
+ getBusyIcon() : ImageIcon
+ bordo() : Color
+ yesil() : Color
+ pembe() : Color
RightPanel
+ loadEvent(e : Event)
MyRightPanel
+ MyRightPanel(eventId : int)
+ setEventId(newEventId : int)
+ loadEvent(e : Event)
+ getUpdatedEvent() : Event
+ actionPerformed(e : ActionEvent)
OthersRightPanel
+ OthersRightPanel()
+ loadEvent(e : Event)
+ actionPerformed(e : ActionEvent)
OthersSchedule
+ OthersSchedule(username : String)
+ actionPerformed(e : ActionEvent)
+ updateButton()
+ removeButton()
+ loadDate(d : OurDate)
+ loadAnEvent(e : Event)
+ createRightPanel() : RightPanel
MySchedule
+ MySchedule(username : String)
+ actionPerformed(e : ActionEvent)
+ updateButton()
+ removeButton()
+ loadDate(d : OurDate)
+ loadAnEvent(e : Event)
+ mouseClicked(e : MouseEvent)
+ mouseEntered(e : MouseEvent)
+ mouseExited(e : MouseEvent)
+ mousePressed(e : MouseEvent)
+ mouseReleased(e : MouseEvent)
+ checkNotification()
+ createRightPanel() : RightPanel

Figure 24 - Factory Method Pattern Diagram

6.1.4. Façade Pattern

Facade pattern is a structural software engineering design pattern where “a facade is an object that provides a unified interface to a set of objects in a subsystem.”(1) It separates a subsystem from another subsystems or classes. “This provides a closed architecture.” (1-reference, slides 16)

In our project we applied this pattern in project’s storage management system. As stated before, we use database as data storage and many classes need to access the database. We have a “DatabaseManager” class which connects to database and make add, delete, update, and get operations from the database. Then, the other classes access the database through this “DatabaseManager” class. They do not directly try to access database directly as shown in the Figure #25. Instead when they need add, delete or get operation; they only call the public methods of the ‘DatabaseManager’ class, then “DatabaseManager” deals with other connection issues. The system with applied Facade Pattern is shown in the Figure #26.

Applying pattern gives the system flexibility to change the database access code without affecting the other system components; we only change the methods in “DatabaseManager” class. Moreover adding new operations also become easy, just adding a new method is enough; it can be used by other classes perfectly.

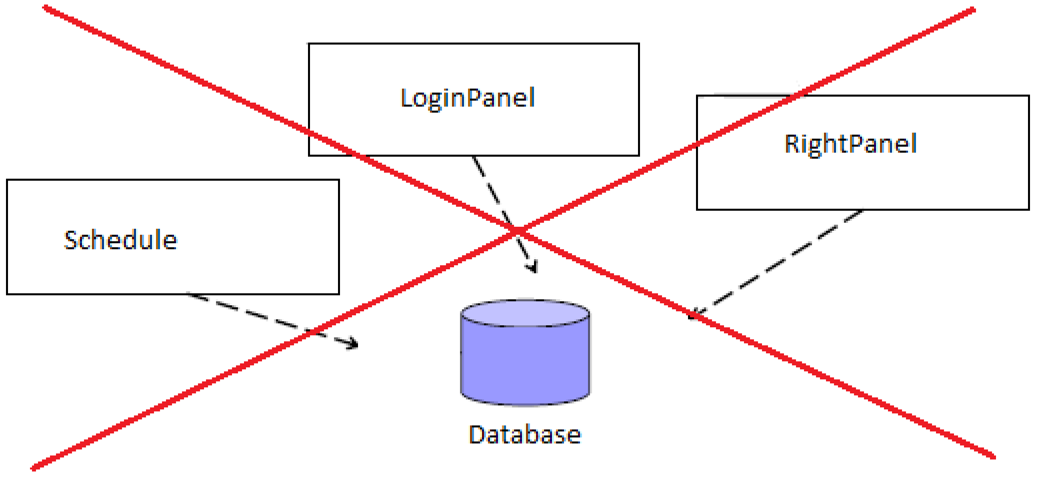


Figure 25 - The bad designed system

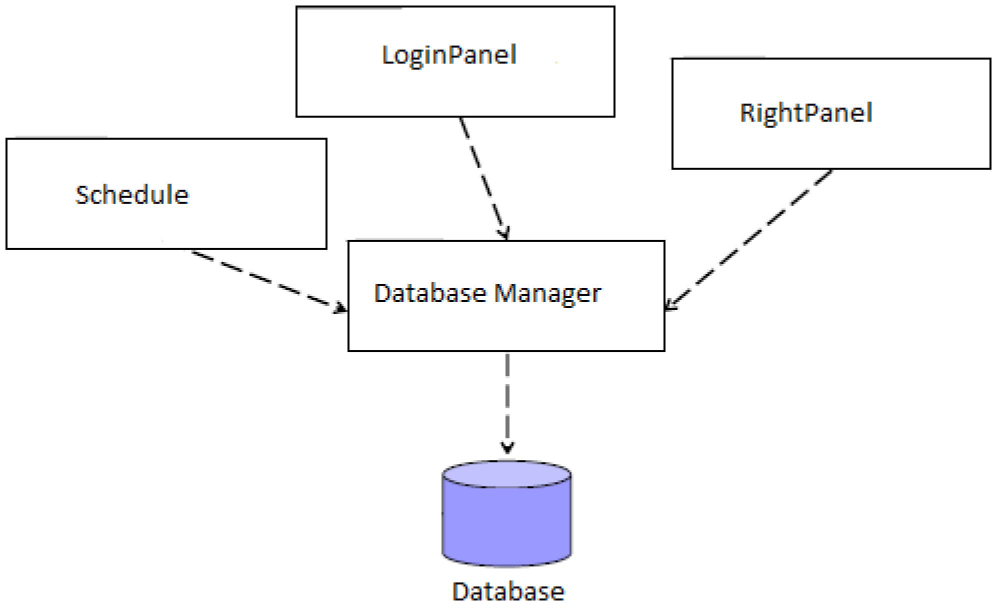


Figure 26 - Facade pattern applied system

6.2. Class Interface

This section includes the description of the classes and their public interfaces. Moreover; their visibility, type signatures, attributes and methods are explained in this section. The classes are explained in three subsections here; GUI package, Application Logic package and Storage Package.

6.2.1. GUI Package

1- ClockPanel Class:

public class ClockPanel extends JPanel{…}

This class is a JPanel as it extends JPanel and it shows the current time in "hh:mm" format by getting it from the Java's Calendar class.

Implements:

ActionListener: 'ClockPanel' has an inner class 'timeListener' which implements 'ActionListener', this is needed to update the label which shows the time.

Properties:

private Timer tmr: Needed for updating the time.

private JLabel clockLabel**:** Label which shows the time with 55 punto and bold font.

Constructor:

public ClockPanel(): It has a simple empty constructor which loads the current time.

2- LoginPanel Class

public class LoginPanel extends JPanel{…}

This class is a JPanel as it extends JPanel and it contains the first screen when you execute the project. User makes login, sign up and delete an account operations here.

Implements:

ActionListener: LoginPanel class has an inner class 'ButtonListener' which implements ActionListener. This deals with the button pressing actions of loginButton, signupButton and deleteButton with actionperformed(ActionEvent e) method.

<u>Properties</u>:

private JLabel usernameLabel,passwordLabel, welcome: their functionalities are explicit.

private JTextField usernameText: Field that user enters his name.

private JPasswordField passwordText: Field that user enters his password.

private JButton loginButton,signupButton, deleteButton: their functionalities are explicit.

private Color blue: Background color.

<u>Constructor:</u>

public LoginPanel(): Simple constructor which creates and adds the components and set their action listeners.

<u>Methods:</u>

private void refreshComponents(): just validate the panel

public void displayLoginError(): Shows a pop-up message dialog when user enters an invalid name or password.

private void emptyFields(): Shows a pop-up message if user presses any button when the fields are empty.

public void resetFields(): Resets the name and password fields.

public void deleteOk(): Shows a pop-up message when an existing account is deleted successfully.

public void signupError(): Shows a pop-up message when user tries to sign up with an existing or invalid name.

3- RightPanel Class

public abstract class RightPanel extends JPanel{…}

This is an abstract class which is super class of ‘MyRightPanel’ and ‘OthersRightPanel’ classes. This is the panel which is shown in the right part of the screen. (See Figure #3) This panel changes when user is seeing other person’s schedule. So we implemented this as an abstract class, then the subclasses can implement their preferences independently.

Properties:

-

Constructor:

-

Methods:

public abstract void loadEvent(Event e): This is an abstract class whose task is loading an 'Event' object's information to the GUI components such as buttons text fields etc. This process is done differently, differently overridden in subclasses.

--

4- MyRightPanel Class

public class MyRightPanel extends RightPanel implements ActionListener{…}

This class is a JPanel which extends 'RightPanel' class. As 'RightPanel' class extends JPanel, 'MyRightPanel' is also a JPanel. It overrides the loadEvent(Event e) method.

Implements:

ActionListener: This interface deals with all the button pressed actions of this class by overriding public void actionPerformed(ActionEvent e) method.

Properties:

private JLabel eventType, description, date, ndate, time, privacy, inviteOthers, notification, scheduleLabel: Simple labels.

private JComboBox typeCombo: Combo box where event type is chosen.

private JTextArea descriptionArea: The area where event's description is shown.

private JTextField dateField1 , dateField, dateField3, ndateField1, ndateField2, ndateField3,timeField1, timeField2, inviteField, scheduleField: Date and time fields.

private JRadioButton publicButton, privateButton, on, off: Buttons where event's privacy and notification is chosen.

private ButtonGroup privacyGroup: Groups the publicButton and privateButton, make them dependent.

private JButton delete, reset, ok, seeSchedule, mySchedule, logout, exit, add: Simple buttons whose functionality is explicit in their names.

private JScrollPane desScroll: Needed to make description area scrollable.

private ClockPanel clock: An instance of ClockPanel to show the clock.

private Event event: This is the loaded event of the panel, event's information is loaded to the GUI components.

<u>Constructor:</u>

public MyRightPanel(int eventId): The constructor takes the ID of the event, then gets the event info from the database with the ID, and loads the event info to the GUI components.

<u>Methods:</u>

public void setEventId(int newEventId): Sets the event's ID of this.

public void loadEvent(Event e): loads the all properties of the event to the right panel, fills the GUI components.

public Event getUpdatedEvent(): This method gets the event information from the GUI components, textFields, radio buttons, and returns this as an 'Event' object. This method is used when user updates the event information by clicking on the 'OK' button.

--

5- OthersRightPanel Class

public class OthersRightPanel extends RightPanel implements ActionListener{…}

<u>Implements</u>:

implements ActionListener: Just handles the button pressed actions.

<u>Properties</u>:

private static Color *blue*: Background color.

private JLabel eventType, description, date, time, scheduleLabel: Simple labels.

private JComboBox typeCombo: The combo box where event type is chosen.

private JTextArea descriptionArea: Area where the event description is shown.

private JTextField dateField1, dateField2, dateField3, timeField1, timeField2, scheduleField: Text fields where date and time is shown.

private JButton seeSchedule, mySchedule, logout, exit: Buttons whose functionality is explicit by their names.

private ClockPanel clock: Shows the clock.

private Event event: Holds the existing loaded event in the right panel

<u>Constructor:</u>

public OthersRightPanel(): Simple constructor which initializes all the components.

<u>Methods:</u>

public void loadEvent(Event e): Loads the Event object e to the GUI components.

public void actionPerformed(ActionEvent e): Handles the button pressed actions of seeSchedule, mySchedule, logout, exit buttons.

--

6- Schedule Class

public abstract class Schedule extends Container{…}

<u>Properties</u>:

protected RightPanel rightPanel = createRightPanel(): Abstract FACTORY METHOD + and RightPanel as property.

private ImageIcon birthday , meeting ,anniversary , todo, busy: Commonly used event type image icons.

private Color bordo, pembe, yesil: Commonly used colors.

<u>Constructor:</u>

-

<u>Methods:</u>

//ABSTRACT METHODS

public abstract void loadDate(OurDate d): //STATE PATTERN: for more info see subsection 6.1.2.

public abstract void loadAnEvent(Event e): Loads a single event to the proper place on the panels.

public abstract RightPanel createRightPanel(): //FACTORY METHOD. For more info about this method, see the subsection 6.1.3.

+public get methods for commonly used image icons and color objects

--

7 - MySchedule Class

public class MySchedule extends Schedule implements ActionListener, MouseListener{…}

Implements:

implements ActionListener: Handles button click events.

implements MouseListener: Handles mouse click events.

Properties:

private OurDate date, curDate: Date holds the date which is shown on the screen. curDate holds the current date.

private JLabel dateLabel, personLabel: Simple labels to display username and date.

private TitledBorder labels[]: Shows day numbers in a month.

private JPanel panels[]: This array has 42 panels each represents a day in a month.

private JButton nextButton, prevButton, lastClicked: Simple buttons to see other months.

private String username: Holds the current user name.

private JPanel topPanel: Top panel has some buttons (nextButton, prevButton etc.) and labels(dateLabel,personLabel etc.) which display and change the current month screen view.

public ArrayList<Event> events: Holds all the events of the user.

Constructor:

public MySchedule(String username): Initialize the components with username parameter. User name is necessary to perform loading events action.

Methods:

public void actionPerformed(ActionEvent e): Handles button clicked events which are clicking on an event button or change the current month on the screen.

public void updateButton(): When an event type changes this method is called to update the icon of the button.

public void removeButton(): When an event is deleted, this method is called to remove the button of the event.

public void loadDate(OurDate d): Loads the date object d to the screen.

public void loadAnEvent(Event e): This method loads a single event to the proper panel with proper button.

public void mouseClicked(MouseEvent e): When a mouse clicked on a panel, this method is called and an event is created with proper button and default info.

public void checkNotification(): This method is called when user opens Cagenda. If the user has notifications popup screen occurs.

public RightPanel createRightPanel(): This method creates an instance of MyRightPanel object and returns is. Factory method pattern is applied here and this is the factory method, for more information see the subsection 6.1.3.

----

8 - OthersSchedule Class

public class OthersSchedule extends Schedule implements ActionListener{…}

<u>Implements</u>:

implements ActionListener: Handles button click events.

<u>Properties</u>:

private OurDate date, curDate: Date holds the date which is shown on the screen. curDate holds the current date.

private JLabel dateLabel, personLabel: Simple labels to display username and date.

private TitledBorder labels[]: Shows day numbers in a month.

private JPanel panels[]: This array has 42 panels each represents a day in a month.

private JButton nextButton, prevButton, lastClicked: Simple buttons to see other months.

private String username: Holds the current user name.

private JPanel topPanel: Top panel has some buttons (nextButton, prevButton etc.) and labels(dateLabel,personLabel etc.) which display and change the current month screen view.

public ArrayList<Event> events: Holds all the events of the user.

Constructor:

public OthersSchedule(String username): Initialize the components with username parameter. User name is necessary to perform loading events action.

Methods:

public void actionPerformed(ActionEvent e): Handles button clicked events which are clicking on an event button or change the current month on the screen.

public void loadDate(OurDate d): Loads an OurDate object to the screen

public void loadAnEvent(Event e): This method loads a single event to the proper panel with proper button.

public RightPanel createRightPanel(): Creates a new 'OthersRightPanel' object and return it. Factory method pattern is applied here; this is the factory method, for more information see Section 6.1.3.

----

9 – PopUpWindow Class

public class PopUpWindow extends JFrame implements ActionListener{…}

This is a JFrame pop up window as its name clearly states. It has some properties other than showing simply a text message, so we make this a separate class. This occurs when a user sends other user an invitation of an event. The invitation pop-up occurs when user opens Cagenda if s/he has an invitation of an event. All the event properties, the invitatory's name are shown in the pop-up. User has two choices, accepting the event or ignoring the event. If s/he accepts the event, it will be saved to user's events.

Implements:

ActionListener: It implements ActionListener class to handle button pressed actions.

Properties:

private JButton add, cancel: Buttons to add the event or cancel the event.

private JTextArea description: Shows the description of the event here.

private DatabaseManager: An instance of the DatabaseManager class is needed to save the event if user accepts that event.

private Event popupEvent:This is the pop upped event.

Constructor:

public PopUpWindow(String name, Event e): This is the only single and default constructor. A pop-up cannot occur without a user name and a valid event.

Methods:

public void actionPerformed(ActionEvent e): Button handling operations are handled here. If the add button is clicked, the event is saved to database and popup frame is closed; if user clicks on the cancel button, simply frame is closed with no other action.

----

10 – MainController Class

public class MainController {…}

This is the main controller of the application.

Implements:

-

Properties:

private static JFrame frame: The only frame of the screen which keeps all panels.

private LoginPanel log: Login panel is shown when the application is opened.

private Schedule main: Schedule object which can be MySchedule or OthersSchedule object.

private static MainController pro: Single pattern applied here for more information see the subsection 6.1.1.

private OurDate current: Holds the date of today.

private String user: Current user name is stored here volatile.

Constructor:

public MainController(): It is like default constructor, but as this is the main controller of the program, no other constructors are needed. It creates and the attributes at first.

Methods:

public static MainController getInstance(): Returns a unique instance of this class, Singleton pattern is applied here, for detailed info see the subsection 6.1.1.

public void showloginScreen(): Loads the login screen to the frame.

public void showCalendar(String username): If the user is logged, this method is called and the schedule of the user with name ‘username’ is loaded to the screen here.

public void showSchedule(): State pattern is applied here, this is explained detailed in the subsection 6.1.2.

public void shutDown(): This method closes and exits the whole application.

public void seeSchedule(String person): This method is called when user wants to see another person’s schedule. It loads the other person’s schedule to the screen.

public void mySchedule(): This method is called when the user wants to return his/her own schedule. It loads the users schedule back to the screen.

public void updateBut(): This updates the event buttons on the screen when an event information is changed.

public void deleteBut(): This deletes the event buttons on the screen when an event information is deleted.

public void handleNotification(Event e): This methods handles the notifications by calling the Schedule classes method loadAnEvet(Event e).

6.2.2. APPLICATION LOGIC Package

1- OurTime Class

This is an simple public class OurTime which extend or implements no other class. This class represents the time with hours and minutes with 24 hour format. It is used in the 'Event' class where each event has a specific time, OurTime object is needed for each Event.

Implements:

-

Properties:

private int minute: Represents the minute as an integer between 0 and 59.

private int hour: Represents the hour as an integer between 0 and 23.

Constructors:

public OurTime(int minute, int hour): Simple constructor which initialize the properties minute and hour.

public OurTime(java.sql.Time convertTime): This constructor creates the instance with java.sql.Time class. This is needed because the time is stored as java.sql.Time in the database, we need to convert this to our time class 'OurTime'.

Methods:

public void setMinute(int m): set the private 'minute' property to m.

public void setHour(int h): sets the private 'hour' property to h.

public int getMinute(): returns the minute.

public int getHour(): returns the hour.

public String toString(): returns the string representation of the time.

2- OurDate Class

This is a simple public class OurDate which extend or implements no other class. This class represents the date by holding three integer properties; day, month and year. This date class is used in 'Event' class where each event has a date so that they have an instance of OurDate class

Implements:

-

Properties:

private int day: Represents the day of the month as an integer, starting with 1 up to 31, depending on the month the upper limit may change.

private int month: Represents the month of the year as integer, starting with 1 up to 12.

private int year: Represents the year as an integer again.

Constructors:

public OurDate(): Default constructor which sets properties day, month and year to 1.

public OurDate(int day, int month, int year): Initializes the properties day, month and year with methods signature variables.

public OurDate(java.sql.Date convertDate): This constructor creates the instance with java.sql.Date class. This is needed beacuse the date is stored as java.sql.Time in the database, we need to convert this to our date class 'OurDate'.

public OurDate(OurDate aDate): Copy constructor

Methods:

public int getDay(): Returns the day.

public int getMonth(): Returns the month

public void setDay(int d): Sets the day property to d.

public void setMonth(int m): Sets the month property to m.

public void setYear(int y): Sets year property to y.

public String toString(): Returns the string representation of Date object.

public int daysInMonth(): Returns the number of days in called date's month as an integer. This method is used when determining which day panels will be active and how should it be numbered (see Figure #3)

public void setCurrentDate(): set the current date in month, year and day by getting it from java's Calendar class.

public int getWeekOf1st(): returns the weekday of the first day of the month in called date object as an integer. Monday, Thursday ... Sunday is numbered as 1, 2, 3..7 respectively.

public OurDate getNextMonth(): Sets the date to the next month of the called date, also returns this updated date. This method is used to get and set the next month when user clicks on the left arrow button.

public OurDate getPreviousMonth(): Sets the date to the previous month of the called date, also returns this updated date. This method is used to get and set the previous month when user clicks on the right arrow button.

public boolean equals(OurDate d): Returns true if the dates are equal wholly, and false otherwise.

3- Notification Class

This is a public class which extends no other classes. Mainly this class is used in 'Event' class where each event has a notification specifying that it may be on or off (null or not). A Notification has only date, no time because the program may not be opened every time on the day, so whenever the user opens the program in that day, the notification pops up, it does not waits any time.

<u>Implements</u>:

-

<u>Properties</u>:

private OurDate date: Date of the notification.

<u>Constructor:</u>

public Notification(): Default constructor which calls the default constructor of OurDate.

public Notification(OurDate newDate): Sets the 'date' to 'newDate'.

<u>Methods:</u>

public OurDate getDate(): Returns the date of the Notification

public void setNotificationDate(OurDate newDate): Sets the date of the notification to newDate.

public void setDateToNULL(): Calls the default constructor of the OurDate class.

public java.sql.Date getDateSQl(): Returns the date in sql.Date format. This is essential because in database date is stored in that format.

public boolean isOn(): Returns true if the notification is on, and false otherwise.

4- Event Class

public class Event {…}

This public class 'Event' is an important base class of the project. This class is a composition of three other classes; 'OurDate', 'OurTime' and 'Notification' and other data members.

Implements:

-

Properties:

private int eventID: The unique id of the event object. One id only relates to one event.

private OurDate date: Represents the date of the event as an OurDate object.

private OurTime time: Represents the time of the event as an OurTime object.

private Notification notification: Represents the notification of the event as an Notification object.

private int privacy: It can be integer '0' or '1'. '0' means that the event is public and 1 means private.

private String description: Represents the description of the event.

private String type: Represents type of the event as a string where a type is one element of the constant array 'eventTypes':

private final String[] eventTypes = {"Meeting", "Birthday", "TO-DO", "Anniversary", "BUSY"};

Constructor:

public Event(int eventID, String description): Events are always first created with this default constructor, where eventId and description is enough to create it.

public Event(Event e): copy constructor

public Event(int eventID, OurDate date, OurTime time, Notification notification, int privacy, String description, String type): This is a full constructor which initializes all the properties, to signature variables.

Methods:

public boolean hasNotification(): Return true if the event has a notification, calls 'Notification' classes isOn() operation inside.

public OurDate getNotification(): Returns the notification date of the event as OurDate object.

public void setNotificationToNULL(): Calls the setDateToNULL() method of the 'Notification' class inside the method to make the notification off.

public java.sql.Date getNotificationSQL(): Returns notification date in sql.Date format.

public java.sql.Date getDateSQL(): Returns the date of the event in sql.Date format.

public OurDate getDate(): Returns the date of the event as OurDate object.

public java.sql.Time getTimeSQL(): Returns the time of the event in sql.Time format.

public OurTime getTime(): Returns the date of the event in OurTime format.

public int getPrivacy(): Returns the privacy, 1 or 0.

public String getDescription(): Returns the description of the event.

public String getType():Returns the type of the event.

public int getEventID():Returns the ID of the event.

public String[] getEventTypes(): Returns the string array of all event types.

public void setId(int id): Sets the ID of the event.

public void setDescription(String d): Sets the description of the event.

public void setType(String t): Sets the type of the event to a valid string t.

6.2.3. Storage Package

1- DatabaseManager Class

public class DatabaseManager {…}

DatabaseManager class is a public class which aims to provide connection between other classes of the program and the MySQL database in a web server. It includes lots of SQL codes to manipulate the database for necessary actions. With the help of this class any other classes can create an object of DatabaseManager class and request information from MySQL database or deliver information to the database.

Implements:

-

Properties:

public Statement stmt: This class in the library of Sql. Sql class should be imported to be able to create this property. Duty of this property is to create statement with using Sql commands and execute necessary actions on the database.

public ResultSet rs: This property is also in the Sql library. It is used to get data from database.

public Connection con: Again this property is in the Sql library. It is used to create a connection between database and our class. This property needs the connection url to be able to connect the database.

private static DatabaseManager myDb: Static DatabaseManager object provides us for creating only one object of the DatabaseManager class. All of the other classes which needs a DatabaseManager object can call a method which uses this property of the class. With the help of this property, there will be no garbage object of DatabaseManager class.

Constructor:

public DatabaseManager(): In the constructor of this class program tries to create a connection with the database with using its connection url and if it fails to create a connection, it caches an exception and displays it on the screen.

Methods:

public static DatabaseManager getInstance(): This method uses static myDb property and checks whether there is already an object of DatabaseManager. If not exist, it creates a new object and returns DatabaseManager object myDb. If exists, it returns existing myDb object.

public boolean checkAccount(String username, String password): While a user logging in to program the system calls this method to checks whether user has already have an account or not. Return value of this method is Boolean. Therefore, according to type of this Boolean value user can enter the system (= yes) or can get an error message while trying to enter the system (= no).

public boolean checkName(String username): Returns a Boolean value. According type of this method, when someone is trying to create a new account for the program, someone can create a new account (= yes) or get an error message which says this username is already exist.

public boolean saveAccount(String username, String password): This method firstly uses checkName method then saves the new account to the database and returns a Boolean value which describes whether operation is successfully completed or not.

public boolean deleteAccount(String username, String password): This method calls checkAccount method then according to result of this method it can deletes the account from database and returns an Boolean value which describes the result of the operation.

public int saveEvent(String username, Event e): This method saves an event to a user then returns an integer value which holds the unique event id to the program.

public boolean deleteEvent(int eventId): This method deletes an existing event and it uses its unique id number to locate it. After the operation, it returns the Boolean value which describes whether delete attempt succeeded or not.

public boolean updateEvent(Event e): This method updates and existing event with updated event “e” and returns the result of the operation as Boolean.

public boolean saveOthersNotification(String username, int eventId): This method save the notification created by other user.

public ArrayList<Event> getNotifications(String username): It returns all notification for a specific person in an event arraylist.

public void deleteNotification(String user): This method deletes all notification of a user.

public ArrayList<Event> getEvents(String username): This method gets all events of a user and returns them in an event arraylist.

public String getUserName(int id): This method returns a string value which is an owner of an existing event.

public Event getEvent(int eventId): This method returns an event object to the program.

7. Conclusion

Cagenda is a distributed agenda system which has functionalities such as adding, deleting and editing an event, seeing other users' schedules and inviting other users to events. Cagenda can be a good alternative to its opponents with its user friendly interfaces and powerful mechanism.

With this CS319 Project, we have gained a lot of experience with documentation, design, coding and group working. Firstly, we have learned a lot of ideas about object-oriented software engineering. While applying the object-oriented software ideas in a project, we have gained deeper perspectives. Critical and analytical thinking was very important while working on the project because nothing was linear. Furthermore, we have learned design is very important before the implementation because after the good design, implementation is very easy. Moreover, in implementation part it was our first time with online database. Before the project, we were afraid about online database but with hardworking, we handled all the difficulties and in the end everything is working almost perfectly. In addition, team working was very important during the project. Working as a team, communication between team members was the key point to achieve a good result. Lastly, we have learned documentation is very important beyond everything because of comprehensibility, reusability, maintenance and reliability.

There were some obstacles during the project. We divided to project to group members but while unifying all the work, there were some conflicts such as misunderstandings, different templates and errors. Moreover, we had difficulties while applying design patterns. We could not adopt our system to patterns directly.

Cagenda has a design and implementation which are very appropriate for improving the system in the future. As a future work, detailed daily view can be added to system. Daily view can make the system more usable and powerful. A lot of people want to see his current day with its details on screen. Moreover, program can execute faster while it accesses to database. Furthermore, web-based Cagenda, which works on a browser, will be a very good improvement. There can be more new functionalities that can be added to the Cagenda in future.

8. References

Instructor/TA help: Bedir Tekinerdogan / Bugra Mehmet Yildiz

Books: Object Oriented Software Engineering 3rd Edition

Other: Course Slides, http://en.wikipedia.org/wiki/Main_Page