Bilkent University

Department of Computer Engineering

Object-Oriented Software Engineering Project

CS 319 Project: Tempo

Design Report

Project Group: 1.G

Member Names: Mert Saraç, A. A. M. Jubaeid Hasan Chowdhury, Burak Erkılıç, Kaan Kıranbay

Course Instructor: Eray Tüzün

April 17, 2018

Table of Contents

1 Introduction 2

1.1 Purpose of the System 2

1.2 Design Goals 2

1.3 Definitions, acronyms, and abbreviations 3

2 High-Level Software Architecture 6

2.1 Design Patterns 8

2.1.1 MVC 8

2.1.2 Singleton 8

2.2 Subsystem Decomposition 9

2.3 Hardware/Software Mapping 11

2.4 Persistent Data Management 11

2.5 Access Control and Security 12

2.6 Boundary Conditions 12

2.7 Global Resources Handling 13

3 Subsystem Services 14

3.1 Event Management 14

3.2 Authorization 15

3.3 Dynamic Notification Management 15

3.4 Smart Event Management 16

3.5 Profile Management 16

3.6 Chat Management 2

3.7 Data Management 2

3.8 User Interface

4 Low-level Design 30

4.1 Object Design Trafe-Offs 15

4.2 Final Object Design 16

5 Improvement Summary 36

6 References 37

Design Report

CS 319 Project : Tempo

1. Introduction

This design report has been made to clarify the gap between our problem and our solution. We will be talking about our purpose, our design goals, our system's architecture and its' subsystem services.

1.1 Purpose of the System

Tempo is a smart scheduling application that combines to-do-list, schedular and event calendar. It is designed for helping users organize their daily events whether with friends or as an individual more effectively and efficiently. It has some similarities with other scheduling applications but it differs from the other applications by having the concept of Smart Event which allows users to arrange meetings easily by checking other participants' free time and to send meeting requests to them. The system's aim is also helping ones about self-improvement by having to-do list and habit tracker.

1.2 Design Goals

Like it was mentioned in Analysis Report, there are many non-functional requirements and these shape our design goals. The most crucial ones of our design goals are mentioned below.

End User Criteria:

Usability: It can be advocated that people do not want to waste too much time to organize their daily routine so, easiness of the system is one of the core features. To makes organizing process faster, Tempo provides some mouse actions such as moving events to another time, creating new event by dragging specific time period in the calendar. In addition,

having one main screen that contains many useful lists like To-Do-List and Friends List keeps all of the core functionality in the home page and allow quick access.

Ease of Learning: Since our program is designed for diverse group of age, ease of learning is so important. In order to achieve ease of learning goal, our design is similar with other popular calendar applications and also help buttons and tooltips are available to help users. In addition, creating, cancelling and rearrangement of events are handled by just mouse instead of complicated buttons so, it increases ease of learning.

User-Friendliness: To attract and encourage users, the program has many graphical features. All events have a color and they are changeable by the users. Various stickers can be attached to describe a event in a better way. Also, users can set a background wallpaper for calendar. Finally, night mode is available for users who wants to use the program if they don't like the brightness of the light colors; brightness of the program does not hurt their eyes by changing default color scheme from light to dark.

Maintenance Criteria:

Reliability: System should be purified from serious bugs and system crash problems. In order to achieve this, many testings will be done and while the implementation of program, all cases are and will be approached carefully. If the system crashes, system will sent a system crash report to inform developers. Another problem is disconnection from internet and by having an offline mode, this problem is prevented partially.

Performance Criteria:

Efficiency: Tempo's system is a responsive system which means it should be fast when users want to use or change anything in the system. Therefore, the program won't consume more than 10 MB of device's random access memory. The system is able to do this by not using big pictures such as high quality pictures but it will use picture formats like Joint Photographic Experts Group (.JPG). The responsive system is helpful while using the database too because the system will be getting information from the users, sending that information to database and saving it, and if needed the information will be received from the database by the users. These informations will not need more than 10KB of memory space to

be stored. Therefore, the responsive system and efficient system will make this program's internet usage as low as possible like 1 Kbps download and upload.

Response Time: The system of Tempo uses little memory and internet as explained in the efficiency part to run the program. This helps program to run fast and have small latency. Some core features like adding a personal event or changing the time of the events or registration to the system will be done in at worst 100 milliseconds. However, some things like adding a friend or adding a Smart Event and inviting people to it will be kind of a slow compared to other features like adding a personal event to the calendar; because when adding people or adding smart events, the system has to send information to database and from database the system will use that information to send invites to the invited people. However, these won't be that slow like 2 seconds but it will take the twice or the three times of 100 miliseconds.

Connection Time: As already mentioned on the previous part, the system uses little memory and internet. This helps the system to connect to database and gather information with low latency. However, there is two more like connecting for the first time and connecting after a disconnect.These two will be treated together as just connecting to the system. Connection time to the system will be at worst 500 milliseconds. If the program can't connect in that interval of time, the program will stop trying to connect and tell user to try again to connect later.

Trade-Offs:

Usability vs. Functionality: Generally, people do not prefer waste too much time on scheduling so, usability has higher priority in our design. In our design, to increase usability, it is avoided from too complex functions but just basic functions that user has used to since having many functionalities causes struggle for user. In addition, having user-friendly interface helps user.

Rapid Development - Functionality: Since we have limited time, rapid development has higher priority in our design. To reduce time for development, instead of other programming languages, JavaFX is prefered for implementation because it has some features that makes the development process easier such as helpful programs as Scene Builder.

Efficiency - Functionality: In our design, we prefer that efficiency has higher priority in our design since. It is decided that too many functionality might create hardship for users if user friendliness and ease of learning are considered. In this way, Tempo has limited features but our main goal is implement these features in a better way. In addition, we have limited time so, it prohibits us to work on numerous features.

1.3 Definitions, Acronyms, and Abbreviations

Since the project has many features, there are some terms that are should be explained:

Smart Scheduling: This feature makes people be able to use their time more efficiently with their friends or co-workers by showing the common time among people to the users.

Event: This feature is the core feature of Tempo. An event is simply an activity that can be whether with your friends (that is called SmartEvent) or without them (that is called Personal Event). In addition, it has some other categories like timeless event, timing event, temporary event and permanent event so, Tempo enables users to combine these categories for describing their event in the best way.

Smart Event: This feature is an event that is created by one of the users' requests to other users who are attending this meeting. For this, when one user is scheduling, he/she can see who is available during this period so he/she can easily change the time period for other individuals. When decided on the time, the user request for Smart Event and then, other participants are notified for this meeting, its explanation, participants, time period. Other participants can whether accept this request or reject. When one participant accepts it, the system waits for other participants response. If all of them accept it, this Smart Event is added to all participants schedule. If any participant does not accept it, system inform the one who creates the event for who did not accept and asks "Do you want to still arrange this event?" without the one who did not accept.

Personal Event: This feature is an event that user's friend list cannot see what user is doing on this time period but they can see he/she is not available during this time period.

Timeless Event: This event type is that is can happen anytime in a day. In simple words, there is no specific time period to do this event like drinking water.

Timing Event: This event type has specific time allocated time for doing this event. For schedule this, user use the weekly-time table.

Temporary Event: This event type is one-time event unless user specifies it will repeat.

Permanent Event: This event type is repeated so when user adds this event to his / her schedule, it will remain until user decides to this event will not be repeated anymore. For

this feature, user can adjust how often this event will repeat like every day, every other day, weekly etc.

Abbreviations:

JRE : Java Runtime Environment. It's a software package that contains what is required to run a Java program.[1] Therefore, JRE needs t o be installed on your computer in order to run Java applications.[2]

2. High-Level Software Architecture

In this section, the subsystem decomposition, hard/software mapping, persistent data management, access control and security, and boundary conditions will be explained in detail. Main goal is here to reduce the coupling between the different subsystems of the system, while increasing the cohesion of the subsystem components.

2.1 Design Patterns

2.1.1 MVC

Our design contains 3 main components that are designed for various features of the system. While designing our subsystems, we decided that our layer system should be a three-layer closed architecture because we wanted our system to be easier to develop and maintain. Therefore, we went with MVC (Model View Controller) architectural style.Our subsystems are classified into 3 different types:

- Model subsystems: Responsible for application domain knowledge such as events, friends, profile etc.
- View subsystems: Responsible for displaying application domain objects to the user such as Calendar,
- Controller subsystems: Responsible for sequence of interactions with the user and notifying views of changes in the model

2.1.2 Singleton

In our project, we going to use Singleton Design Pattern. Mostly likely to use for static data holders and helper classes such as CommuncationHelper class and Storage class for saved data. We decided to use this pattern because it makes easier to use helper and data holder classes while developing various features.

2.2 Subsystem Decomposition

Our design contains 8 main components that are designed for various features of the system. These are User Interface, Event management, Authorization, Dynamic Notification Management, Smart Event Management, Profile Management, Chat Management, and Data Management due to it contains many features of other components as it is shown in Figure 1.

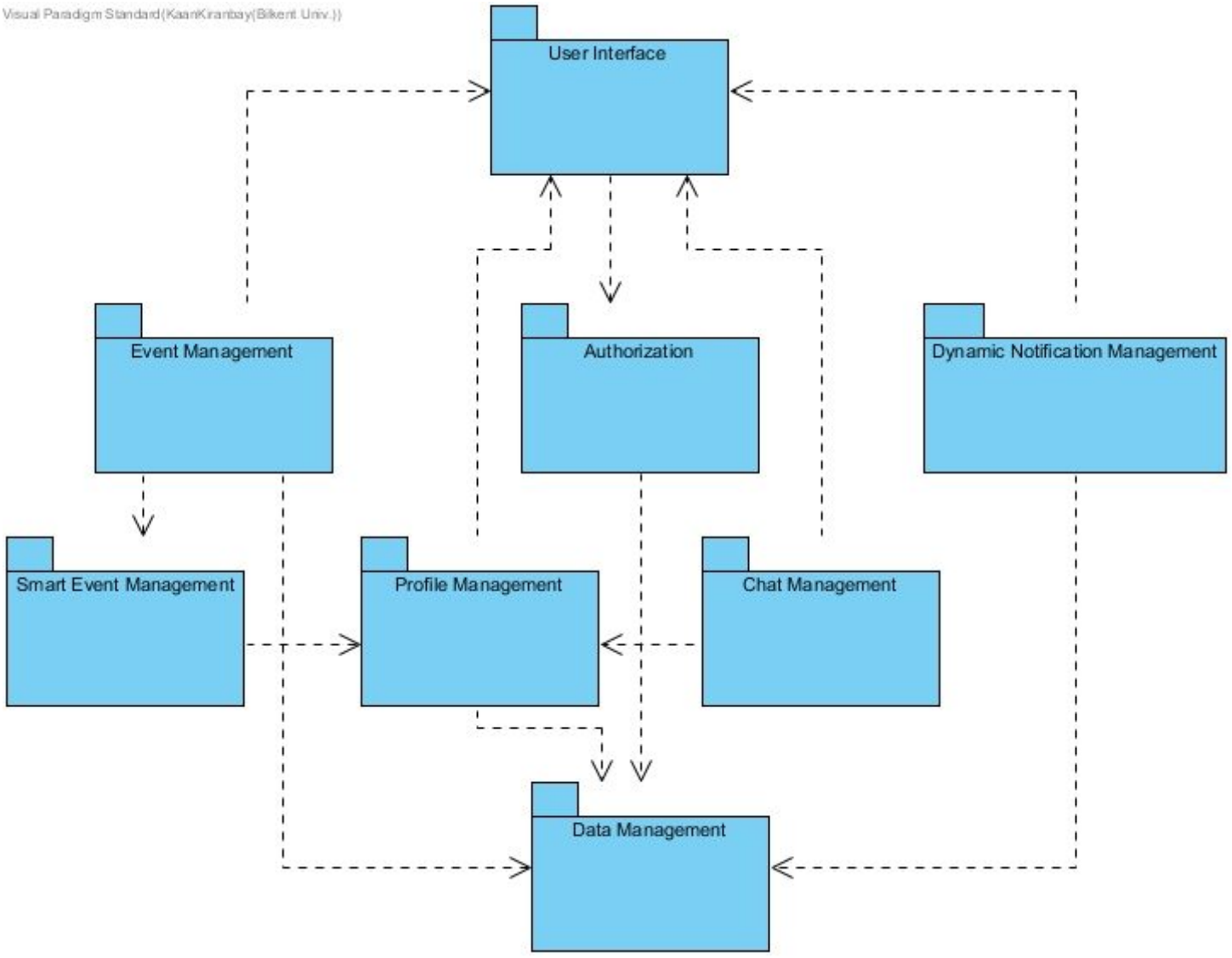

Figure 1 - Subsystem Decomposition

The authorization subsystem collects user info which is then checked against a central dataholder in the database to match the infos and allow the user to access his own profile.

The data management section is responsible for storing all the model objects and interacting with the database. It contains data holder classes for storing event objects, profile objects and notification objects.

The Smart Event management is a subsubsystem to event manager which is responsible for creating and controlling Smart Events. However, the displaying of this smart events is still done by user interface subsystem.

The profile management subsystem contains classes that controls user profile information and friend information, and makes friend lists and creates new friends.

Dynamic Notification Management subsystem which has an active connection with the database and communicates with it about friends requests and Smart Event requests of users. If any friend request gets accepted then this subsystem will give info to user interface to make it show a pop-up notification and send message to profile management subsystems to add that user to friend list.

Chat management subsystem helps interaction between user and his friends via user interface and profile management subsystems.

The user interface subsystem contains a javaFX API calendar and GUI material that are used to display events and to-do-list. The event management subsystem is the controller section for creating and displaying all types of events.

2.3 Hardware / Software Mapping

Since our system is developed by Java, it needs Java Runtime Environment to be executed. For hardware requirements, the system needs a keyboard and a mouse to be controlled by user and also for interaction between them. For graphical part, Java graphics do not need too much GPU and it is compatible for majority of the personal computer systems.

- Control Objects -> Processor
- Entity Objects -> Memory and Database
- Boundary Objects -> Keyboard and mouse for input, and Monitor for output

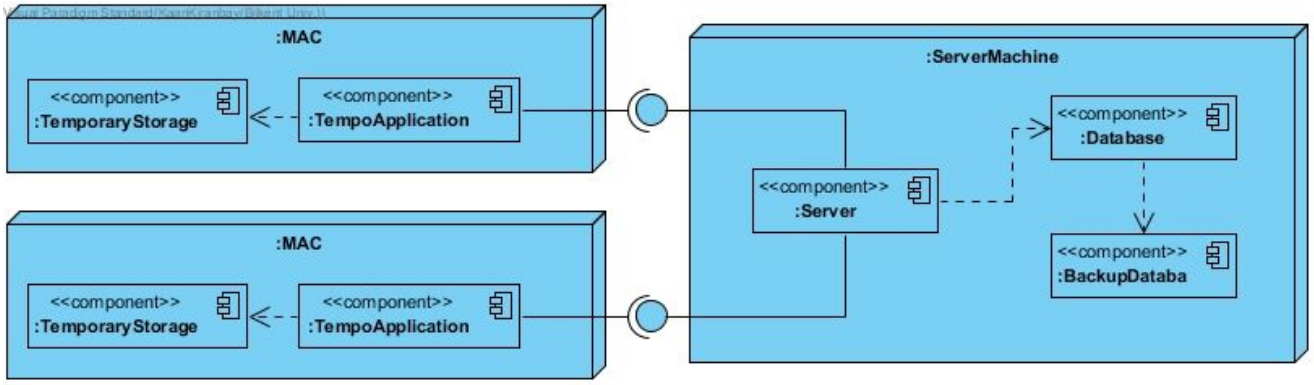

Figure 2 - Deployment Diagram

2.4 Persistent Data Management

Most of the data is stored in the database. However, if the user is already logged in by using internet connection from a device, the user will be able to use the application's some of features like adding personal event offline. To acquire this feature, the system uses data management subsystem to store the data in device memory temporarily. When the user is back online, the data is updated to the database and the device storage data is deleted. For the database, user data is stored upto one year. Once a data passes one year it gets automatically deleted.

2.5 Access Control and Security

Users needs to register or login for the usage of the system. By asking two times for users' new password eliminates the problem of creating an password that a user does not know. Also a user needs to enter name, surname and valid email address to enroll to the system correctly. For login part, unique username and password are needed to access the profile and personal calendar. Users' passwords will be matched by md5 encoding to be secured.

2.6 Boundary Conditions

Initialization

Tempo is a click and run application for the personal computers that have java runtime environment. Therefore, there won't be a install file for Tempo. Tempe is a application that is an executable .jar file.

Disconnection

Tempo is a system that works with internet connection. In case of disconnection, many features of the system becomes not usable due to not connecting with the database.

Termination

Tempo can be terminated or closed by clicking the exit button on the user interface all the times. When a users clicks exit(X) button, the system will disconnect itself from the database. However, this termination will not log the user out of the system. The user will be able to use the system if there is no internet connection as an offline user.

Error

The system will give an error if the file or the saved data in the database are corrupted and will delete its contents.

2.7 Global Resource Handling

Access Matrix

	Calendar	**Profile**	**Event**	**Smart Event**
User	view() change() drag()	view() edit() changeSettings()	<<create>> edit() relocate()	<<create>> approve() ignore()
Friend	view()	view()		<<create>> approve() ignore()

3. Subsystem Services

3.1 Event Management

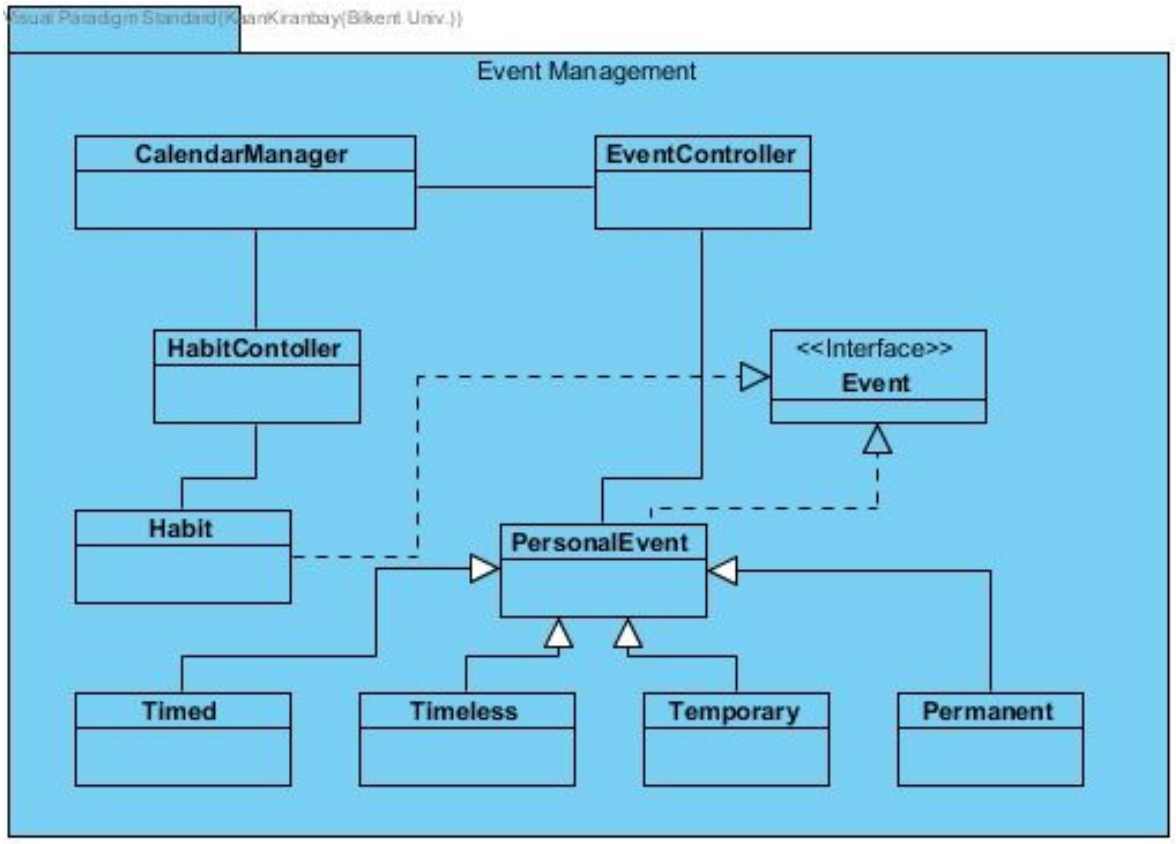

Figure 3 - Event Management Subsystem

Event Manager has classes to create, update and remove events. Event Manager interacts with User Interface to show events on the calendar or To-Do List. It has also connection with Smart Event Manager and Data Management. Since Event Manager can add, remove and update events, it needs to create changes in Data Management that keeps track of the data. For Smart Event Manager, they have connection since a smart event is also an event and it uses some features in Event Manager.

3.2 Authorization

Figure 4 - Authorization Subsystem

Authorization subsystem has Authorization package. Authorization Services contains classes for sign up and login. It has connection with User Interface and Data Management. Interaction with Data Management allows to login a user and it saves the profile of a new user that signed up into the database. Other interaction with User Interface allows to show Login / Sign Up page.

3.3 Dynamic Notification Management

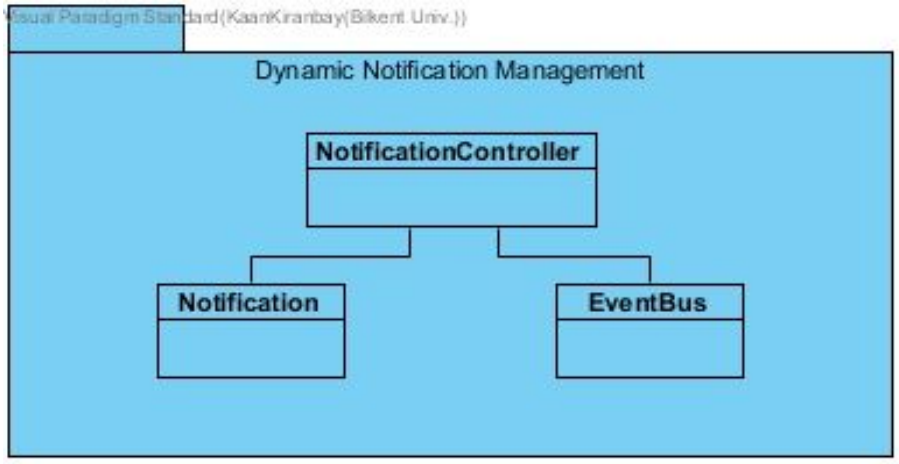

Figure 5 - Dynamic Notification Management Subsystem

Dynamic Notification Management always works on background and notifies changes to user. According to that, it has connection with Data Management to identify changes for notifying it. Also, it has connection with User Interface to notify changes to user as a message.

3.4 Smart Event Management

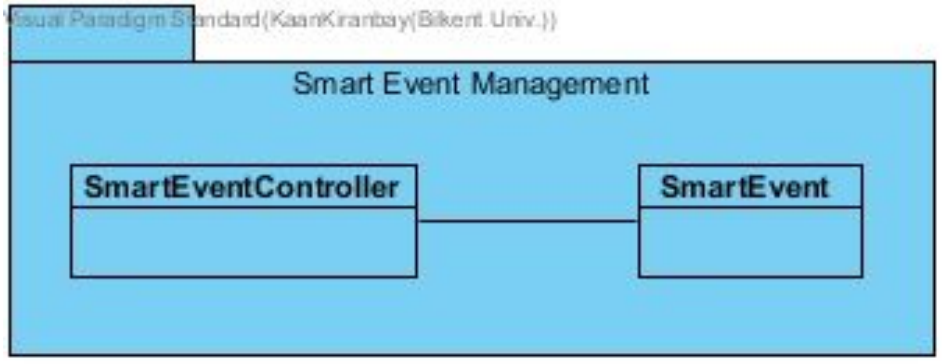

Figure 6 - Smart Event Management Subsystem

Smart Event Manager contains classes for Smart Event's needs and operations. It has interaction with Event Management and Profile Management. Interaction with Event Manager was mentioned above. For interaction with Profile Management, Smart Event Management needs one's friends and their availability.

3.5 Profile Management

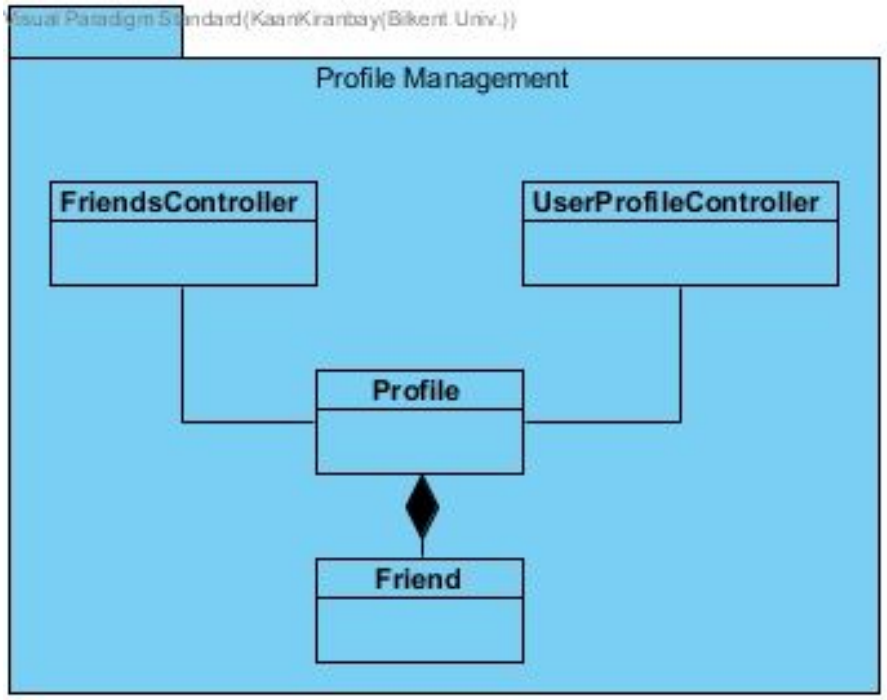

Figure 7 - Profile Management Subsystem

Profile Management has interaction with Chat Management, Data Management and Smart Event Management. Due to the fact that Profile Management contains classes for add and remove friend and possible changes should be also done in Data Management, it has connection with Data Management. To chat someone, that person should be user's friend and to check it, there is a connection between Chat Management and Profile Management.

3.6 Chat Management

Figure 8 - Chat Management Subsystem

Chat Management has 2 connections. One of them is with Profile Management which is mentioned in 3.6. Other one is with User Interface to display the chat in the application.

3.7 Data Management

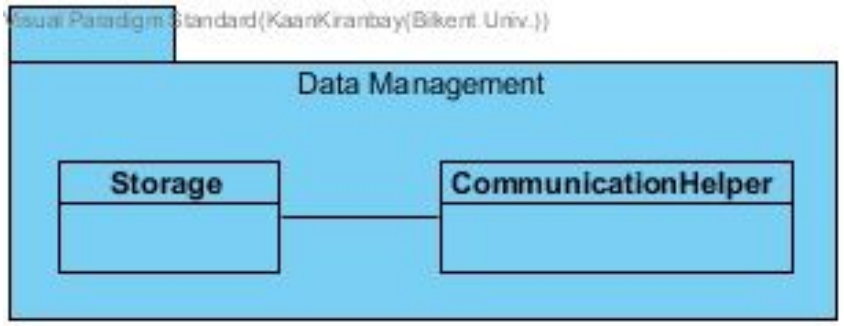

Figure 9 - Data Management Subsystem

Data Management retrieves data and changes data from the database when it is requested or needed, and it contains classes for that. Its connections were mentioned in 3.1, 3.2, 3.3 and 3.5.

3.8 User Interface

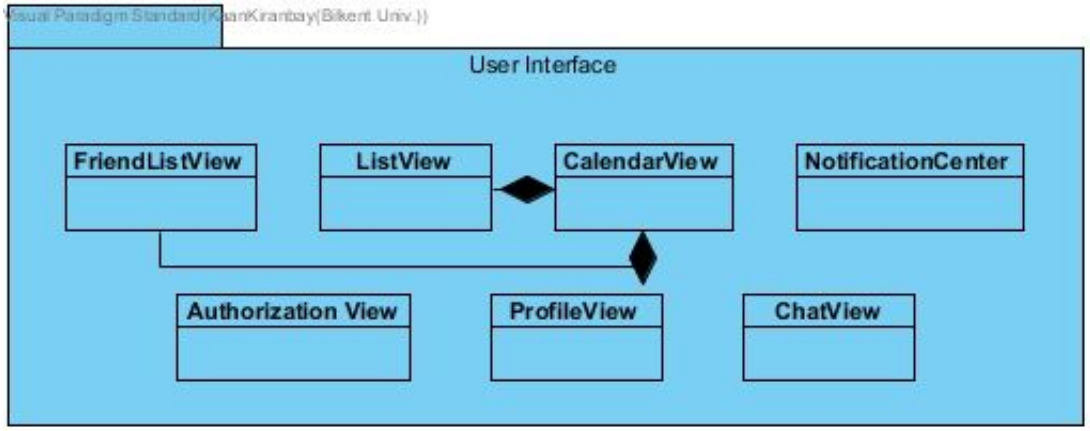

Figure 10 - User Interface Subsystem

User Interface contains the classes for pages like buttons, frames, labels etc. Its connections were mentioned in 3.1, 3.2, 3.3, 3.5 and 3.6.

4. Low-Level Design

4.1 Object Design Trade-offs

Complexity vs Maintanance and Speficity: We have a lot of classes for event types. We choose this so that each type of event has its own specific functionality and keeps everything in a object oriented manner. However doing so requires adds more lines of code and the functions of each event has to be designed differently. However because it's object-orientated it is easier to add other types of event as necessary in the future.

4.2 Final Object Design

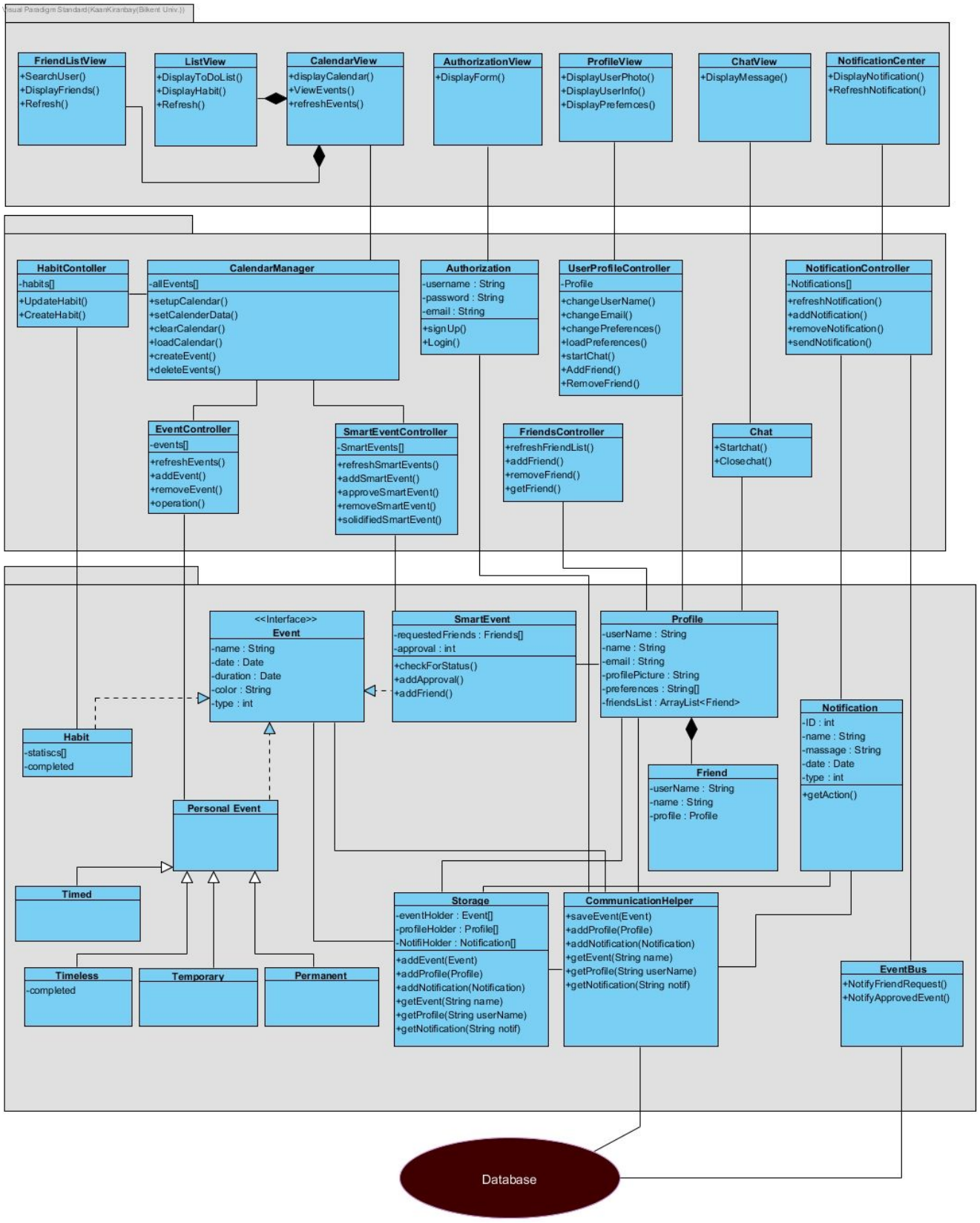

Figure 11 - Final Object Design

4.2.1 Event Management

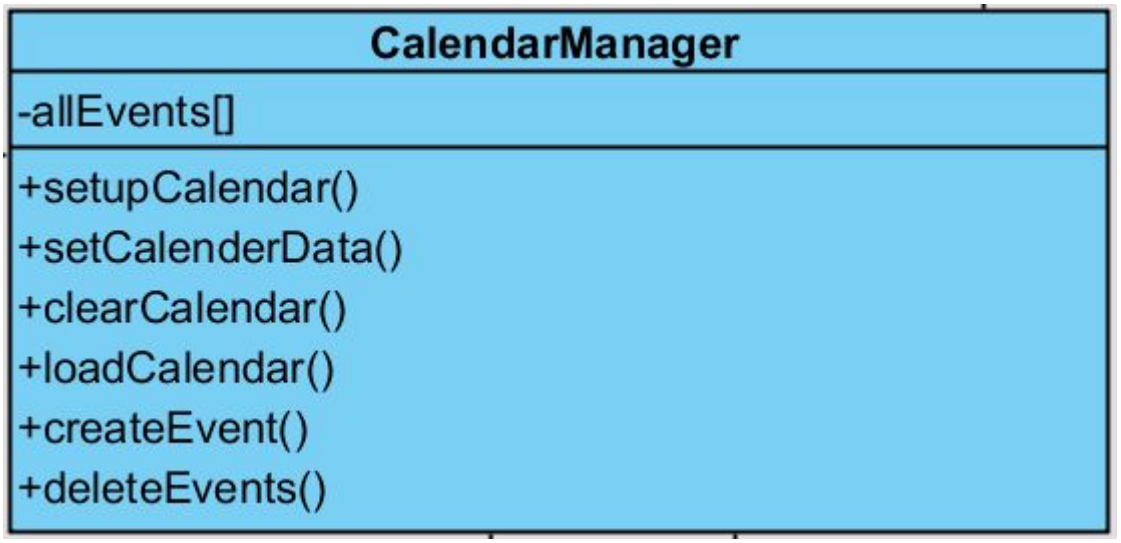

Figure 12 - CalendarManager Class

CalendarManager has an array of all events and as functions, it has createEvent() and deleteEvent() that creates and deletes an event, clearCalendar() that is for clearing all events in the calendar, setCalendarData() for for finalizing that changes that can be whether creating an event or removing it. , loadCalendar() for updating recent calendar and setupCalendar() for updating date.

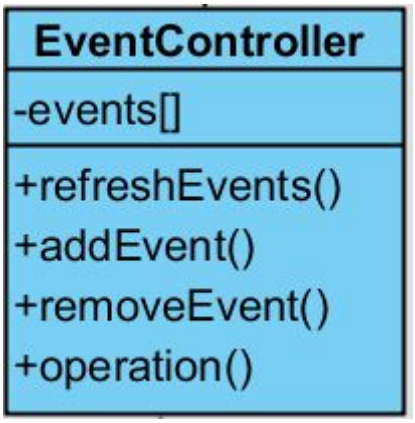

Figure 13 - EventController Class

EventController has an array of events and 4 methods. refreshEvent(), addEvent(), removeEvent(), operation(). refreshEvent() refreshes events to check whether there is a change in an event. addEvent() creates an event and removeEvent() deletes it. operation() …

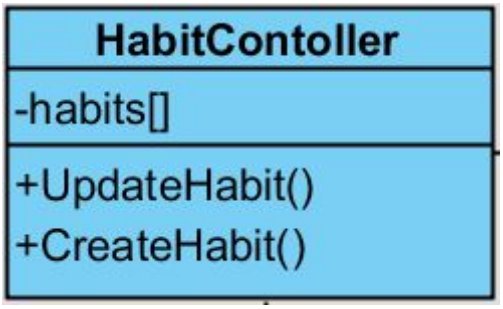

Figure 14 - HabitController Class

HabitController class has an array of habits and 2 methods. UpdateHabit() refreshes the habits' instances. CreateHabit() creates new habits according to user's wantings.

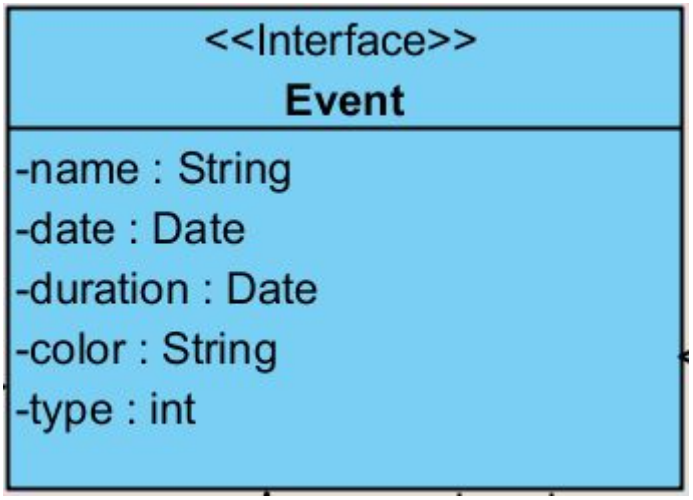

Figure 15 - Event Interface

Event interface has only attributes of 5. Those identfy the events names, dates, durations, colors, and event types.

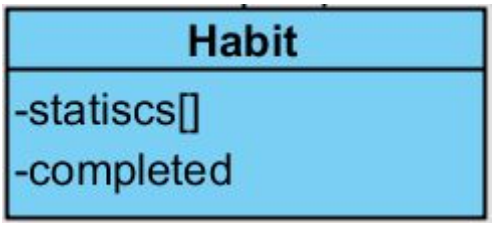

Figure 16 - Habit Class

Habit class has only 2 attributes except the attributes from event interface. Those 2 attributes are to keep the statistics and whether the habit is completed or not.

Figure 17 - PersonalEvent Class

Personal Event has no attributes but it uses event interface.

Figure 18 - Timed Class

Timed has no attributes but it uses event interface.

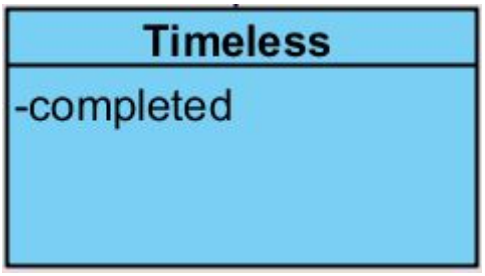

Figure 19 - Timeless Class

Timeless has one attribute to specify whether the timeless event is completed or not also it uses event interface.

Figure 20 - Temporary Class

Temporary has no attributes but it uses event interface.

Figure 21 - Permanent Class

Temporary has no attributes but it uses event interface.

4.2.2 Authorization

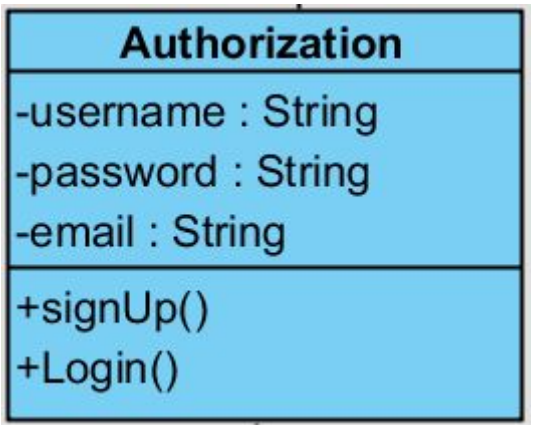

Figure 22 - Authorization Class

It contains user's username, password and email. It has 2 functions that are signUp() that is for sign up operation and login() that is for login operation. signUp() takes username, email and password but login needs just username and password.

4.2.3 Dynamic Notification Management

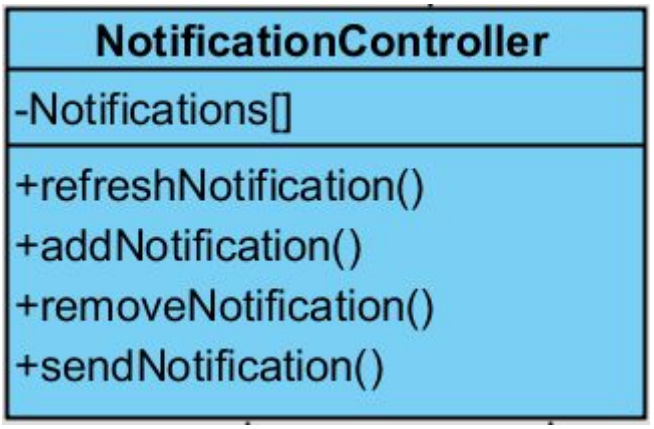

Figure 23 - NotificationController Class

NotificationController has an array of notifications and 4 methods which are refreshNotification(), addNotification(), removeNotification(), sendNotification(). refreshNotification() checks whether there is a new notification, addNotification() creates new notification, removeNotification() deletes current notification after it is read and sendNotification() sends to notification.

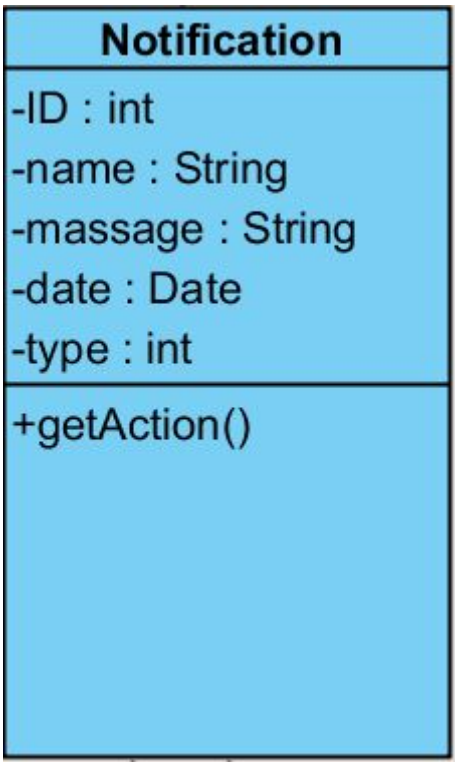

Figure 24 - Notification Class

Notification contains ID, message, name, data and type for a notification since notifications are unique.

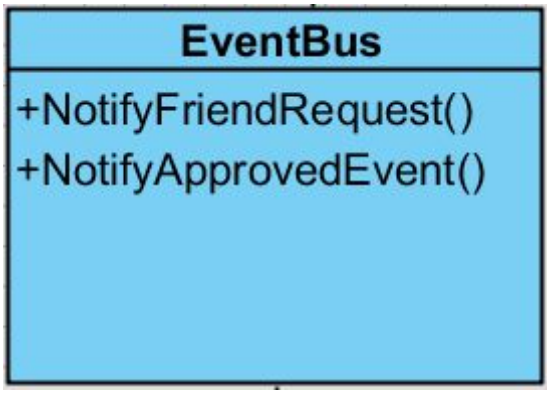

Figure 25 - EventBus Class

Event Bus has 2 methods for friend list and Smart Event which are notifyFriendRequest() and notifyApprovedEvent(). notifyFriendRequest() displays message when there is a friend request and informations about it like who send it. notifyApprovedEvent() displays a message for owner of the Smart Event when it is approved.

4.2.4 Smart Event Management

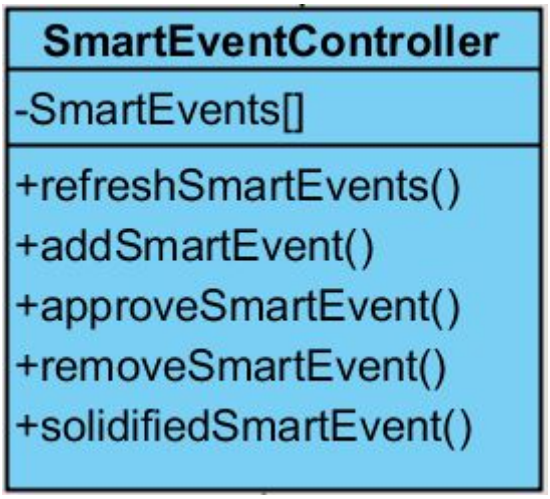

Figure 26 - SmartEventController

SmartEventController has SmartEvent array and 4 methods. refreshSmartEvents() checks whether there is a change in smart events. addSamrtEvent() creates a smart event and add it to calendar. approveSmartEvent() accepts the request of a smart event and adds to the calendar. removeSmartEvent() removes existed smart event from calendar and send a message to owner of the event.

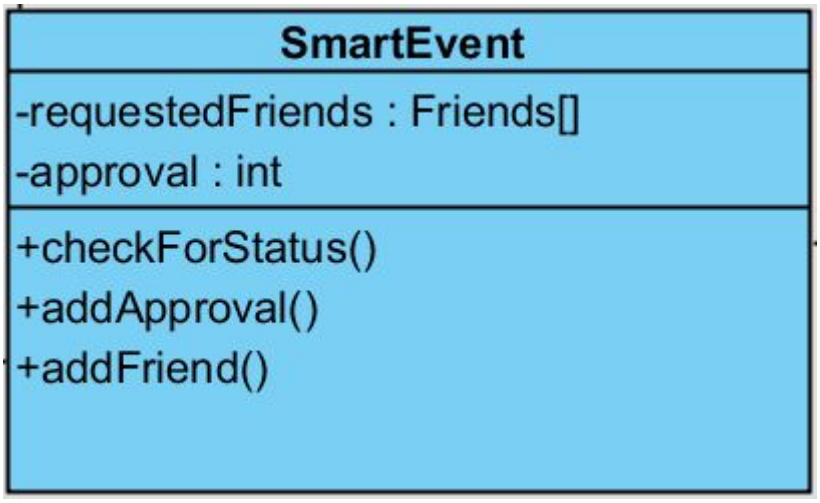

Figure 27 - SmartEvent Class

checkForStatus() checks smart event condition that means whether it is approved or not. addApproval() shows that one participant of the event accepts it and addFriend() adds a friend to user's list.

4.2.5 Profile Management

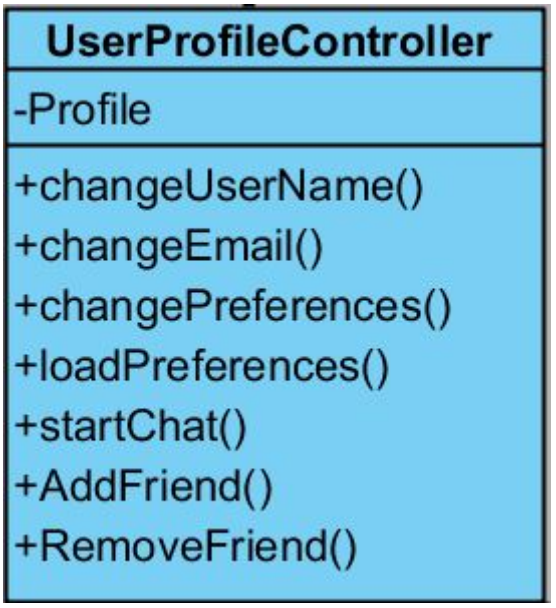

Figure 28 - UserProfileController Class

UserProfileController has a profile and 7 methods. changeUserName() is for changing existing username. Similarly, changeEmail() and changePreferences() are for changes in corresponding domain. starChat() starts chat with particular friend in friend list. addFriend() and removeFriend() is used for adding and removing a friend.

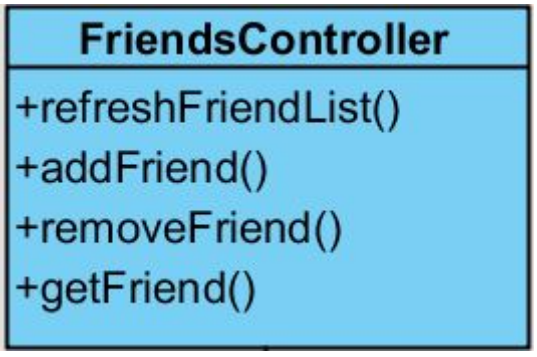

Figure 29 - FriendsController Class

refreshFriendList() refreshes friend list and shows if there is any change. addFriend() is used for adding a friend. Similarly, removeFriend() is used for removing one from a friend list. getFriend() is used for obtaining information about that friend like his/her free-time.

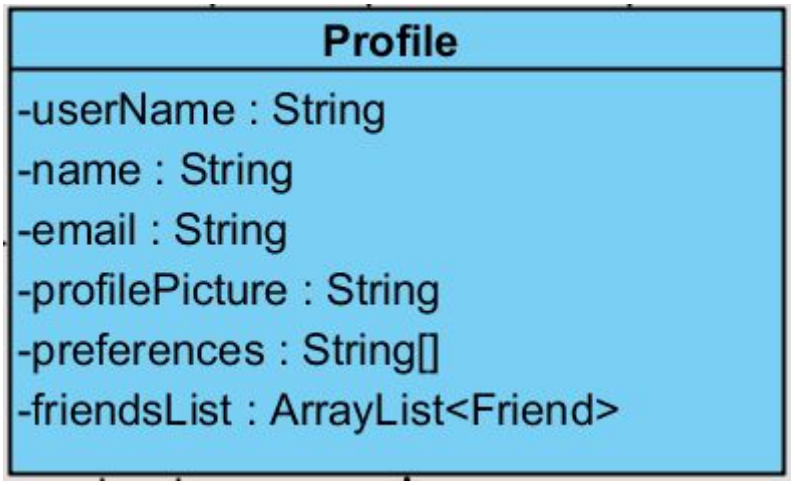

Figure 30 - Profile Class

In profile, there is username, name, email, profilePicture, preferences of that user and list of friends that is an array list called is friendList.

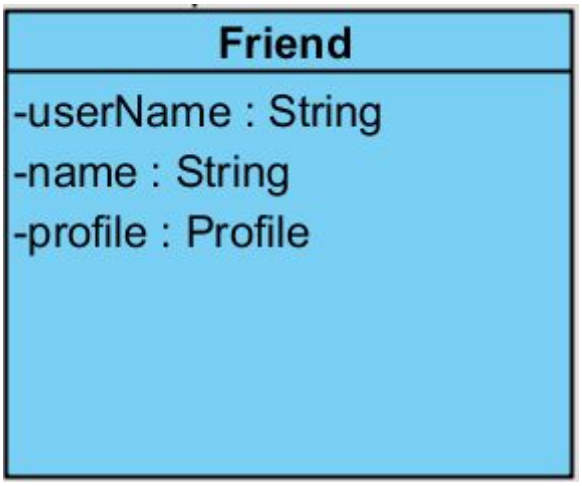

Figure 31 - Friend Class

A friend has username, name and a profile.

4.2.6 Chat Management

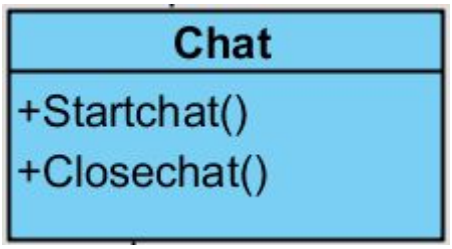

Figure 32 - Chat Class

It has 2 functions that are startChat() and closeChat(). startChat() is activated whenever user sends a message and closeChat() ends that conversation.

4.2.7 Data Management

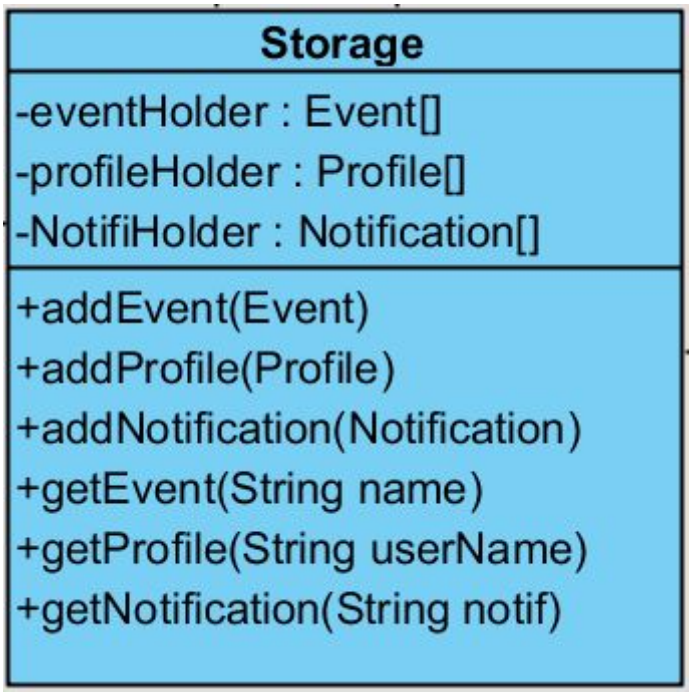

Figure 33 - Storage Class

Storage class is main repository for saved and loaded data which is from database and used in various part of the program.

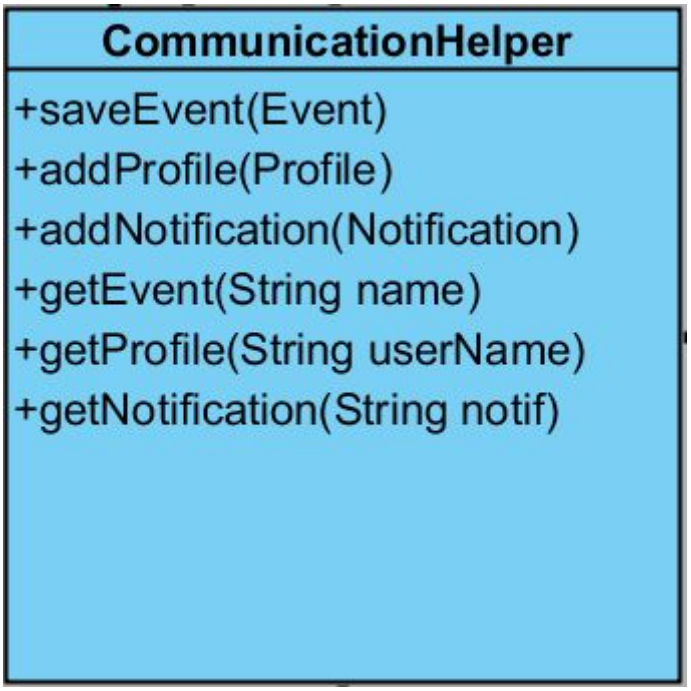

Figure 34 - CommunicationHelper Class

4.2.8 User Interface

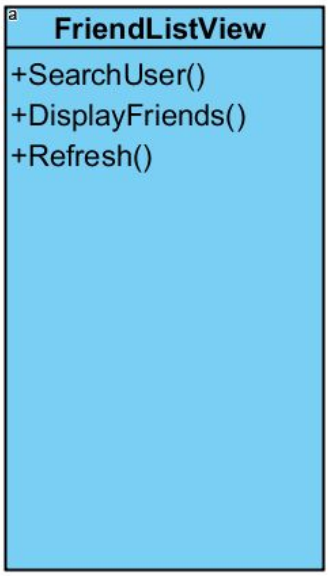

Figure 35 - FriendListView Class

FriendListView class has 3 methods. SearchUser() displays a search window for users to search people. DisplayFriends() displays user's friends. Refresh() refreshes the friend list.

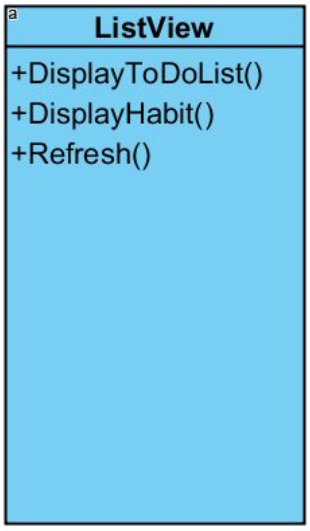

Figure 36 - ListView Class

ListView class has 3 methods. DisplayToDoList() displays the to-do list. DisplayHabit() displays the habits. Refresh() refreshes both of them.

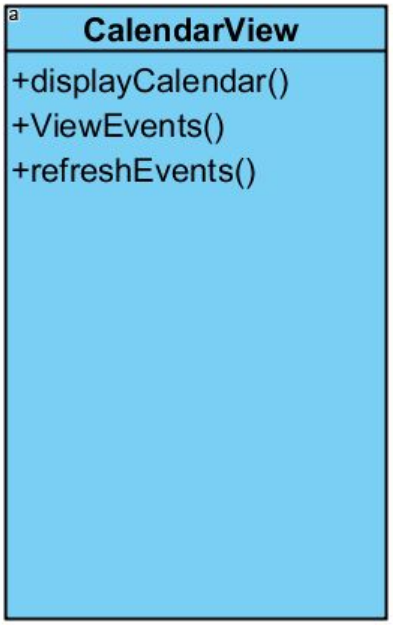

Figure 37 - CalendarView Class

CalendarView class has 3 methods. displayCalendar() displays the calendar according to users choice of which calendar will be shown like a weekly or a monthly one. ViewEvent() displays the events on the calendar. refreshEvents() refreshes the calendar.

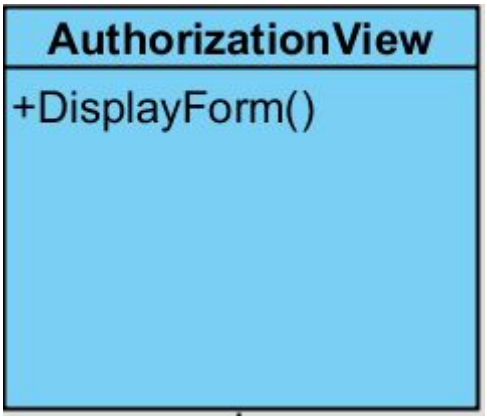

Figure 38 - AuthorizationView Class

DisplayForm() displays the login and register page.

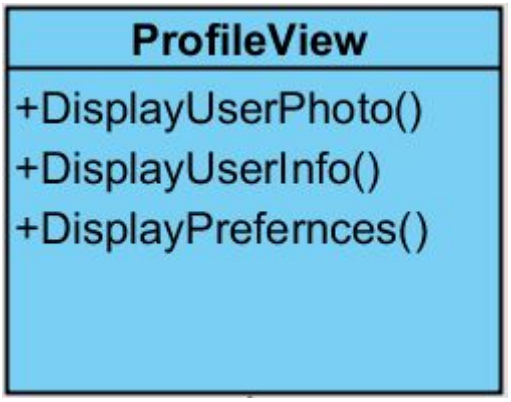

Figure 39 - ProfileView Class

ProfileView class uses 3 methods. DisplayUserPhoto() displays the .JPG photo. DisplayUserInfo() displays the user's information such as username, passowrd, e-mail. DisplayPreferences() displays the settings of the program like switching on or off the night mode.

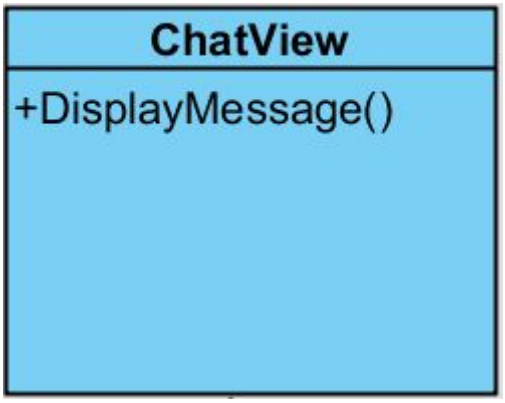

Figure 40 - ChatView Class

displayMessage() displays message from a user and it works whenever a message is sent.

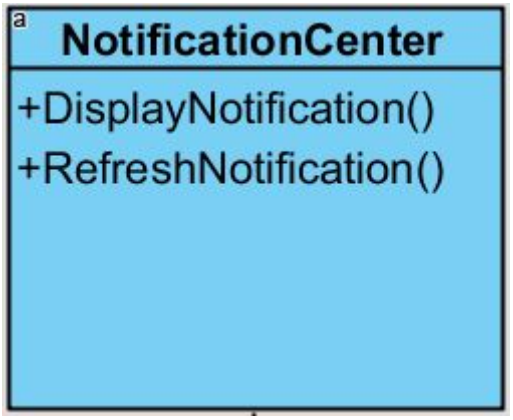

Figure 41 - NotificationCenter Class

displayNotificaiton() displays the notification and refreshNotifiaction() refreshes for check whether there is any change.

5. Improvement Summary

In previous version, the system has authorization page, calendar page with just monthly view and preferences page. Also, user can add an event. However, after demo, we decided to add some features and improve the ones that we have. In this way, we add habit event as independent from timeless event. Previous version's timeless event contains habits but after our instructor's advise, we changed it. Then, we also add a page that contains statistics about all works user did or did not. The system keeps track of the actions that user did and shows some plots about the improvement. Another issue that we have discussed is about chat feature. If we have chat feature, it allows user to communicate easier. In implementation part, database is created by MongoDB and is connected for authorization. In some classes, more detailed attributes and operations are provided and connection between classes and subsystems are revised. Authorization page's design is changed.

6. References

[1] https://en.wikipedia.org/wiki/Java_virtual_machine [2]

https://techterms.com/definition/jre