Bring ideas to life
VIA University College

First Semester Project

Software Engineering

Maria Asenova 239533

Ioan-Claudiu Hornet 297113

Ionut-Florentin Grosu 297111

Cezary Korenczuk 299118

Supervisors:

Mona Andersen - SSE

Allan Henriksen - SDJ

Line Egsgaard - RWD

VIA University College

Software Engineering

First Semester

4th of June 2020

Table of content

List of Appendices **i**

List Of Figures

Abstract

[all group members]

VIAFit fitness center needs a single user system with the purpose of managing data and a website which can display their schedule. The aim of the project is to deliver a system handling the data of the VIAFit. The purpose of this report is to document the different stages of development.

The development was initialized with the phase of analysis. To plan the technical performance of the software, requirements, use case diagrams, descriptions and activity diagrams were created. The findings of this stage were used for the design phase. Class and sequence diagrams were made setting up the stage for implementation which was done in Java. The GUI was created by using the javafx library and Scene Builder. The implementation was followed by a test phase.

The result is a software allowing the management of data, stored in binary files. The functionality of the system enables the user to add, edit and remove data. The schedule information, exported as an XML file, is displayed on the newly designed website. Hence, the aim of the project is fulfilled and all VIAFit's requirements are met.

1. Introduction

[all group members]

Digital transformation helps businesses develop faster, improve their efficiency and enhance customer experience.The days when employees used to visit an office with a pile of paper are long gone. Operating digitally reduces paper waste, improves employees satisfaction and supports business processes (Shane Barker, 2019).

This is the case for VIAFit, a small fitness center. The owner wants to take the company into the digital age and overhaul the archive system and website. Currently, the data in the fitness center is stored in a physical way, using paper and a white board. The center keeps information regarding instructors, the classes they teach and the members registered (Appendix A). As the amount of data kept at the gym increases, it gets more difficult to manage the data (The Local, 2018).

The fitness center is currently facing problems with keeping data saved and updated. Storing data on a piece of paper can raise a lot of issues such as loss and maintenance of data. A few years back the center experienced an accident where they lost half of their data because of a lit candle.

Moreover, the monthly schedule for classes, used to inform both instructors and members, is kept on a white board at the gym. There was an accident where the data on the white board was erased. The manager could not fully recover the information which led to confused instructors and members.

Due to the previously experienced accidents, VIAFit needs a digital way of storing and managing data. As the fitness center keeps a large amount of data, it should be accessible and easy to maintain.

The owner also needs to upgrade the current website of VIAFit. The new website should include the monthly schedule and inform the visitors about the fitness center. For more details on the background description, look at Appendix A.

Based on the needs presented above analysis, design, implementation and test will follow.

2. Analysis

[all group members]

To help VIAFit store data digitally a system will be developed. The first step of the development process is to understand the needs of the user.

The extracted keywords from the interview have been used to create functional requirements which represent what the system should be able to do. These requirements have been categorized by importance.

2.1. Functional Requirements

Critical Priority

2.1.1. As an employee, I want to register a new member with a full name, address, e-mail, type of membership and a unique phone number, so the member can be identified in the system by a phone number.

2.1.2. As an employee, I want to add a new class in the system with its name and maximum participants, so that it is safely stored and accessible.

2.1.3. As an employee, I want to assign a date, time and a relevant instructor to a class, so that it can be added to the schedule.

2.1.4. As an employee, I want to identify a class in the schedule within a certain timeframe, so that I can remove it.

2.1.5. As an employee, I want to be able to upgrade the membership, so that a member is allowed to attend classes.

2.1.6. As an employee, I want to register an instructor with a full name, address, e-mail, the class taught by the instructor and a unique phone number, so that the instructor can be identified by phone number.

2.1.7. As an employee, I want to export the schedule for a specific month, so that it can be displayed on the website.

High Priority

2.1.8. As an employee, I want to identify a member and display the personal data, so I can edit it.

2.1.9. As an employee, I want to identify and display the information about a class, so I can edit it.

2.1.10. As an employee, I want to be able to downgrade the membership, so that a member is not allowed to attend classes.

2.1.11. As an employee, I want to identify an instructor and display the personal data, so I can edit it.

2.1.12. As an employee, I want to sign up a member for a class, so that the member is added as a participant for the class, if the class is not already full.

2.1.13. As an employee, I want to cancel a member's attendance from a class, so that the member is removed from the list with participants for the class.

Low Priority

2.1.14. As an employee, I want to identify a member, so that it can be deleted.

2.1.15. As an employee, I want to identify a class by its name, so that I can remove it from the system.

2.1.16. As an employee, I want to preview a schedule in a specific month, so that I can get an overview.

2.1.17. As an employee, I want to identify an instructor, so that it can be deleted.

2.2. Non-Functional Requirements

2.2.1. The system needs to be implemented in Java programming language and running on a single computer.

2.2.2. Every update in the system includes writing to a file

2.3. Use Case Diagram

A use case diagram has been made based on the previously presented functional requirements. This diagram represents the user's actions in relation with the system.

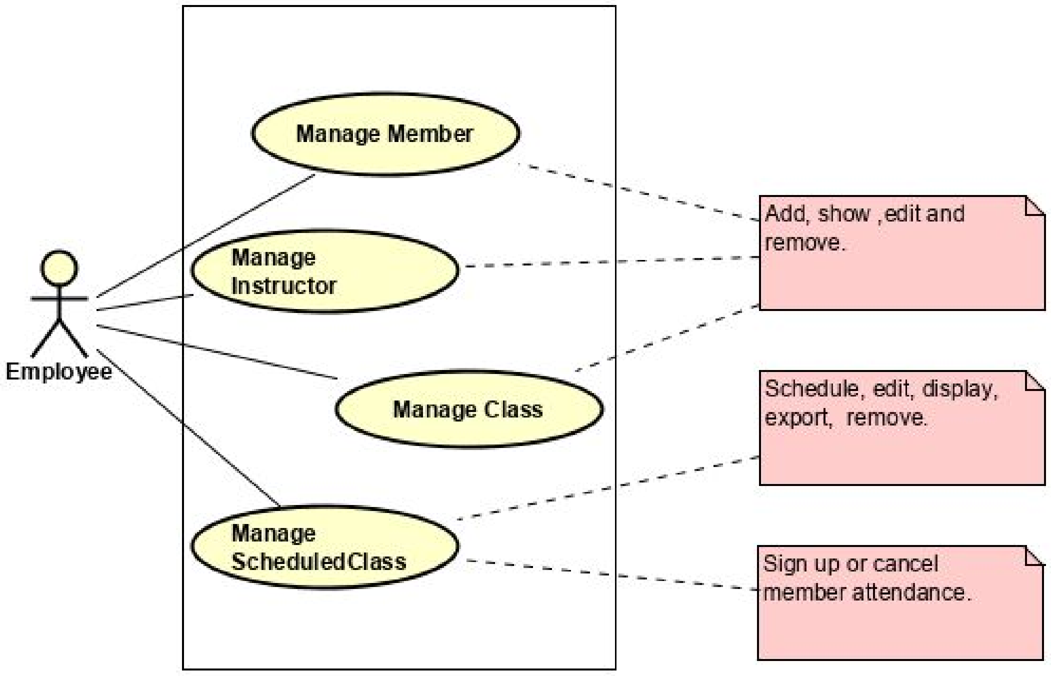

Manage Member
Manage Instructor
Manage Class
Manage ScheduledClass
Employee
Add, show ,edit and remove.
Schedule, edit, display, export, remove.
Sign up or cancel member attendance.

Figure 1. Use case diagram

2.4. Use Case Description

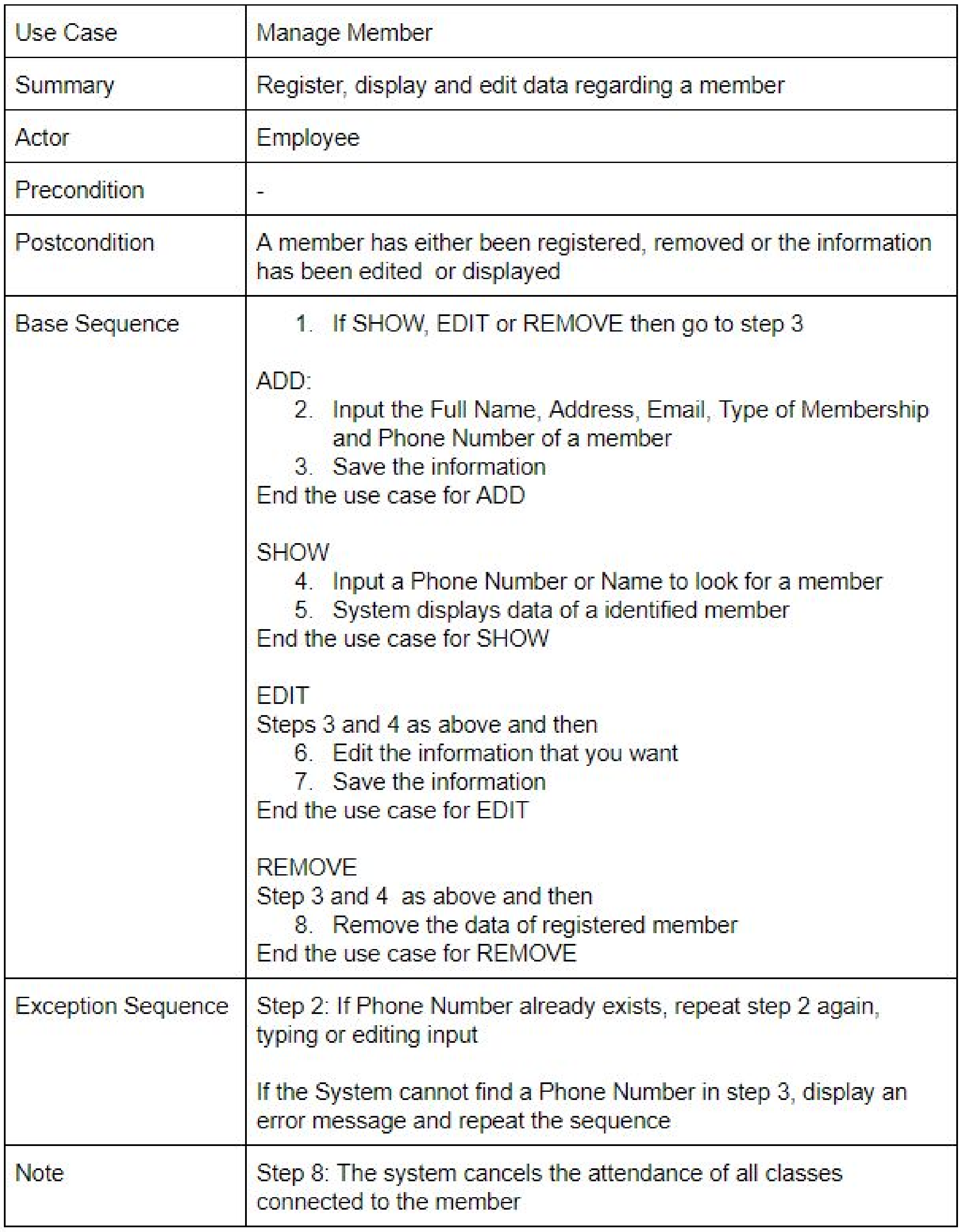
As an extension of the use case diagram four use case descriptions have been made for each action - manage member, manage instructor, manage class and manage scheduled class.

The use case for managing a member consists of registering, displaying, editing and removing data about members.

Based on the requirements, Manage Member is an essential use case since it serves as a base for other use cases, such as Manage ScheduledClass.

For adding a new member, the process consists of adding personal information and saving it. For everything else, the data will have to be displayed first. Afterwards, it can be edited or removed.

The action of managing information about members is presented in a step by step description in the following use case. To see all of the use case descriptions, look at Appendix B.

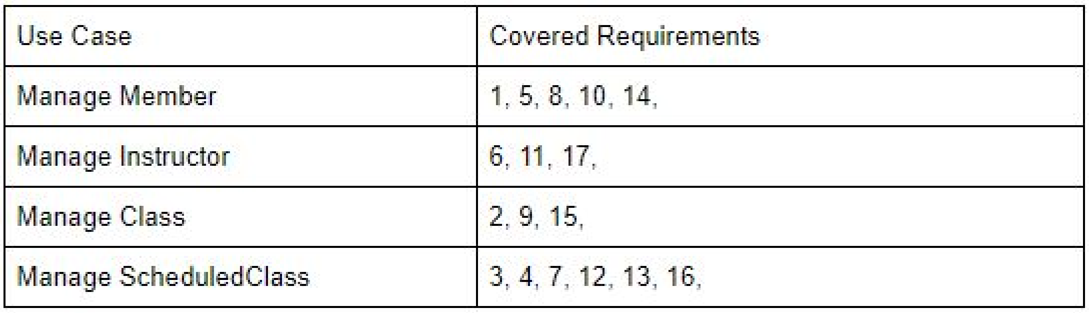


Use Case	Manage Member
Summary	Register, display and edit data regarding a member
Actor	Employee
Precondition	-
Postcondition	A member has either been registered, removed or the information has been edited or displayed
Base Sequence	1. If SHOW, EDIT or REMOVE then go to step 3 ADD: 2. Input the Full Name, Address, Email, Type of Membership and Phone Number of a member 3. Save the information End the use case for ADD SHOW 4. Input a Phone Number or Name to look for a member 5. System displays data of a identified member End the use case for SHOW EDIT Steps 3 and 4 as above and then 6. Edit the information that you want 7. Save the information End the use case for EDIT REMOVE Step 3 and 4 as above and then 8. Remove the data of registered member End the use case for REMOVE
Exception Sequence	Step 2: If Phone Number already exists, repeat step 2 again, typing or editing input If the System cannot find a Phone Number in step 3, display an error message and repeat the sequence
Note	Step 8: The system cancels the attendance of all classes connected to the member

Figure 2. Use case description for manage member

2.5. Link Between Requirements and Use Cases

The connections between the Use Cases and the Requirements they cover can be observed in the following table. For more details on the link between the requirements and use cases steps, look at Appendix C.



Use Case	Covered Requirements
Manage Member	1, 5, 8, 10, 14,
Manage Instructor	6, 11, 17,
Manage Class	2, 9, 15,
Manage ScheduledClass	3, 4, 7, 12, 13, 16,

Figure 3. Link between requirements and use cases

2.6. Activity Diagram for manage member

The activity diagram shows how the process of managing a member is handled. The first step for the user is to choose between adding a new member or displaying information about an existing one.

The process of registering a member consists of adding personal information. One of the key identifiers for a member within the system is phone number. Creating a new member includes verification if the phone number inserted by the user already exists in the system. In case the phone number is not identified within the system, a new member will be saved. Otherwise, the user will need to input a new set of data.

The process of displaying data about existing members starts by searching either by full name or a phone number. Based on the input the system searches for matching member. In case the member is not found, a warning message is displayed and the user has to input a new set of data.

Once the user has inserted valid information, the member's information will be displayed with the options of editing or removing it. Deleting a member from the system will also cancel the member's attendance from the classes. For more activity diagrams, look at Appendix D.

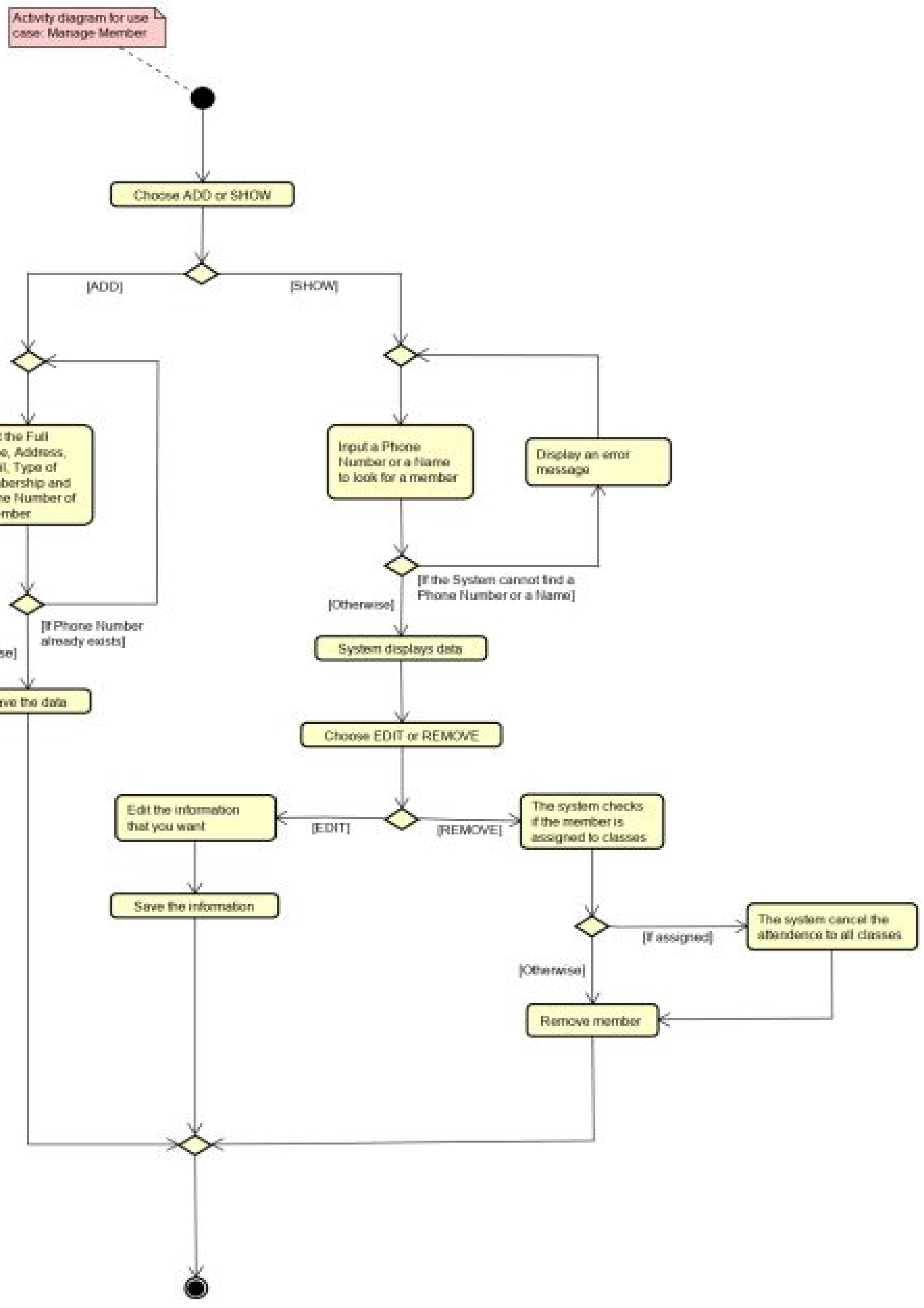

Activity diagram for use case: Manage Member
Choose ADD or SHOW
[ADD]
[SHOW]
Input the Full Name, Address, Email, Type of Membership and Phone Number of a member
[If Phone Number already exists]
[Otherwise]
Save the data
Input a Phone Number or a Name to look for a member
Display an error message
[If the System cannot find a Phone Number or a Name]
[Otherwise]
System displays data
Choose EDIT or REMOVE
[EDIT]
[REMOVE]
Edit the information that you want
Save the information
The system checks if the member is assigned to classes
[If assigned]
[Otherwise]
The system cancel the attendence to all classes
Remove member

Figure 4. Activity diagram for Manage Member

2.7. Domain Model

A domain model has been created based on the functional requirements representing the relationship and multiplicity between classes.

This represents the relations between the classes discovered in the previous analysis steps. The FitnessCenter can exist with or without members, instructors, classes and scheduled classes. For a scheduled class to be created a date and time should be provided.

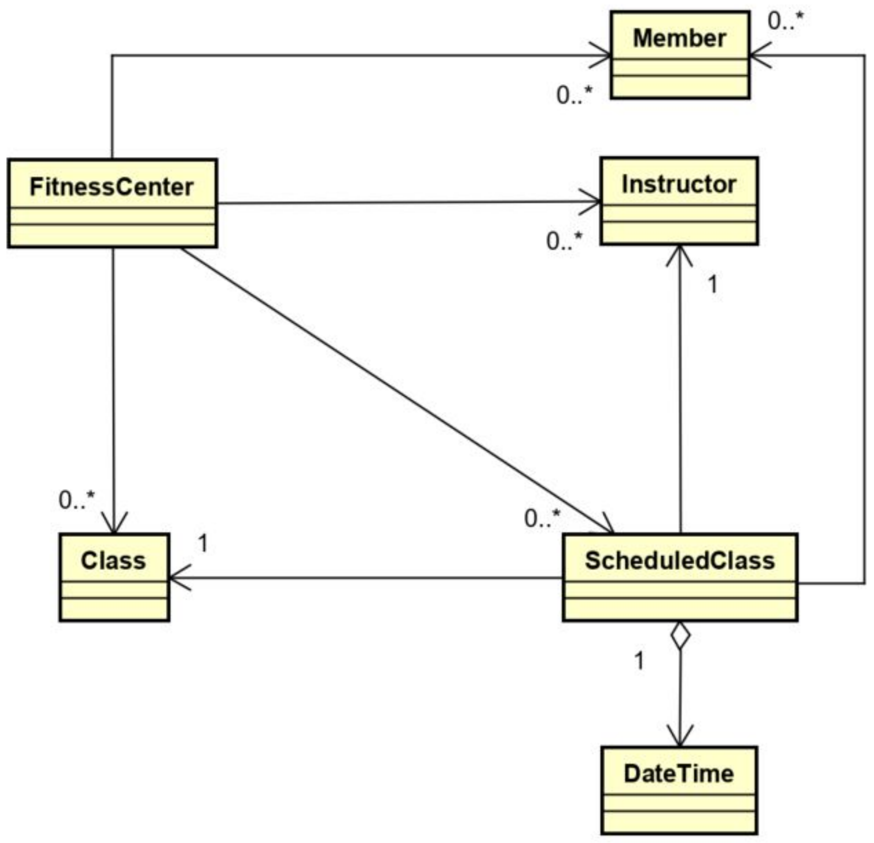


Figure 5. Domain model

3. Design

[all group members]

3.1. Class Diagram

The class diagram was created based on the Domain Model and acts like a blueprint for the implementation process.

As the instructor and member classes have common fields and methods, an abstract class called Person was created. The class was defined as abstract since it is used only for inheritance and not for creating instances of the class.

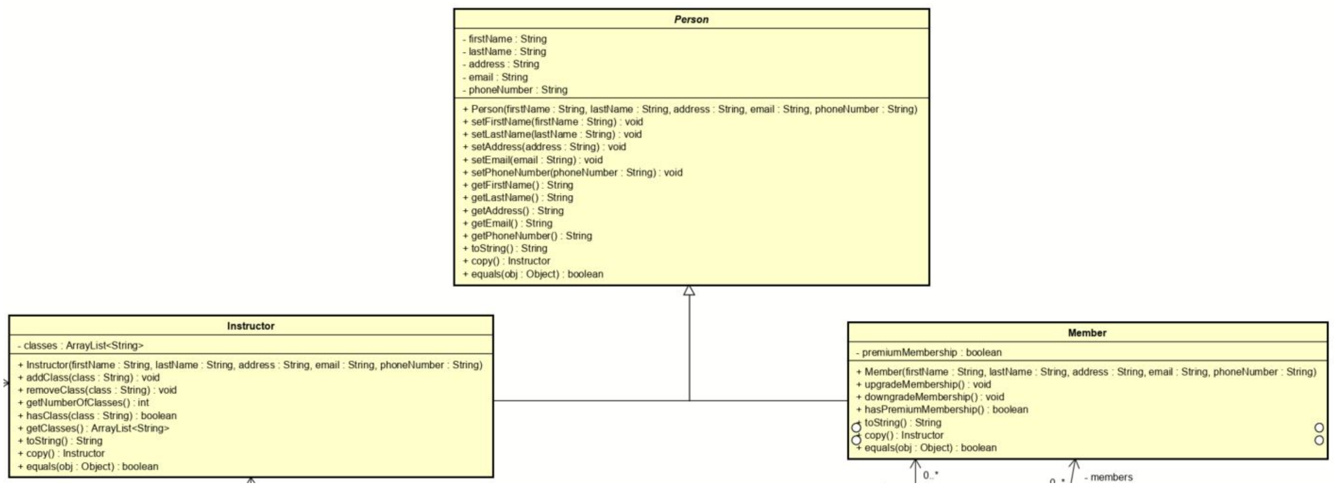

Person
- firstName : String
- lastName : String
- address : String
- email : String
- phoneNumber : String
+ Person(firstName : String, lastName : String, address : String, email : String, phoneNumber : String)
+ setFirstName(firstName : String) : void
+ setLastName(lastName : String) : void
+ setAddress(address : String) : void
+ setEmail(email : String) : void
+ setPhoneNumber(phoneNumber : String) : void
+ getFirstName() : String
+ getLastName() : String
+ getAddress() : String
+ getEmail() : String
+ getPhoneNumber() : String
+ toString() : String
+ copy() : Instructor
+ equals(obj : Object) : boolean
Instructor
- classes : ArrayList<String>
+ Instructor(firstName : String, lastName : String, address : String, email : String, phoneNumber : String)
+ addClass(class : String) : void
+ removeClass(class : String) : void
+ getNumberOfClasses() : int
+ hasClass(class : String) : boolean
+ getClasses() : ArrayList<String>
+ toString() : String
+ copy() : Instructor
+ equals(obj : Object) : boolean
Member
- premiumMembership : boolean
+ Member(firstName : String, lastName : String, address : String, email : String, phoneNumber : String)
+ upgradeMembership() : void
+ downgradeMembership() : void
+ hasPremiumMembership() : boolean
+ toString() : String
+ copy() : Instructor
+ equals(obj : Object) : boolean
0..*
- members

Figure 6. Inheritance

The File Adapter extracts functionality from the GUI and the IO classes and makes it easier to be used by the FitnessCenter class. To see the class diagram, look at Appendix E.

The class diagram has two IO classes, MyfileIO and MyTextFileIO.

MyFileIO is used for reading and writing data to a binary file, such as instructors, members, classes and scheduled classes. This method of storing data was used since binary files are more efficient and compact in comparison with text files (Tony Gaddis, 2015).

In order to export a month of scheduled classes, the MyFileTextIO class was created. The class is responsible for reading and writing to an XML file. It has methods for both writing and appending to a file.

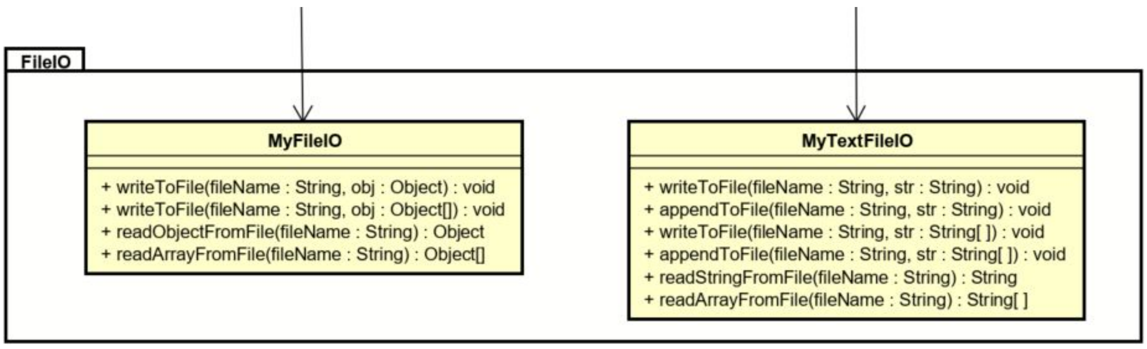

FileIO
MyFileIO
+ writeToFile(fileName : String, obj : Object) : void
+ writeToFile(fileName : String, obj : Object[]) : void
+ readObjectFromFile(fileName : String) : Object
+ readArrayFromFile(fileName : String) : Object[]
MyTextFileIO
+ writeToFile(fileName : String, str : String) : void
+ appendToFile(fileName : String, str : String) : void
+ writeToFile(fileName : String, str : String[ ]) : void
+ appendToFile(fileName : String, str : String[ ]) : void
+ readStringFromFile(fileName : String) : String
+ readArrayFromFile(fileName : String) : String[ ]

Figure 7. FileIO class diagram

3.2. Sequence Diagram

The register member sequence diagram describes the flow of registering a member (registerMember()).

Firstly, the data about the member is inputted by the user. The method registerMemeber() is called in the GUIController which triggers the getMember() method to the FitnessCenterFileAdapter class.

From the adapter the method readObjectFromFile() is called to the MyFileIO class with the purpose of reading the already existing members from the binary file. After the object is received by the adapter, an ArrayList<Member> is created with the returned object. The adapter loops through the received ArrayList<Member>, comparing each member's phone number with the phone number previously entered by the user. Afterwards, the adapter returns the member, an object of type Member.

In the GUIController the returned object is being compared. If its value equals null, a new Member object is created. The newly created Member object is added to the binary file by calling the saveMembers() method from GUIController which triggers the writeToFile() method from the MyFileIO class.

Finally, a confirmation message is returned to the GUIController and displayed to the user. To see the sequence diagram for registering a member, look at Appendix F.

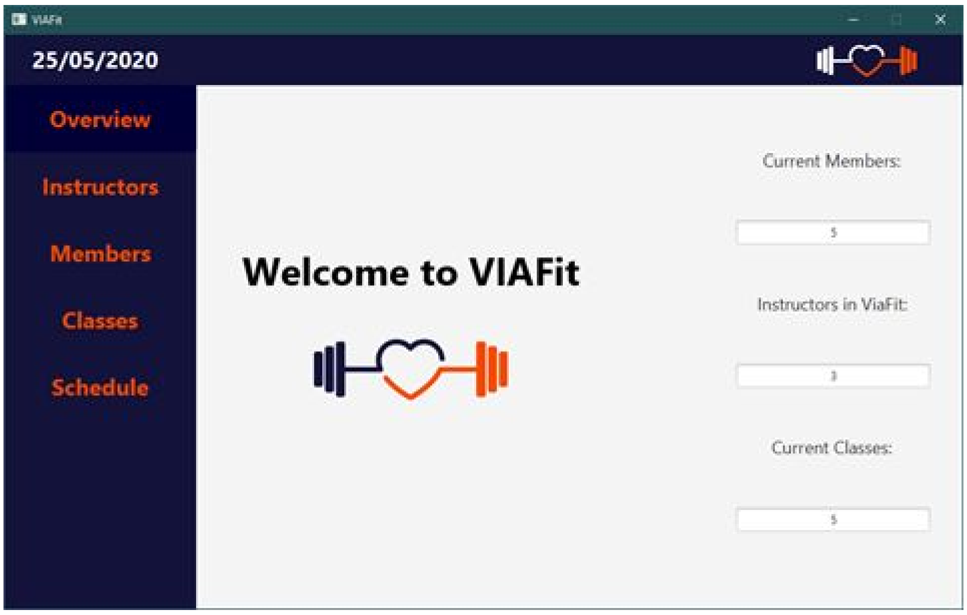
To see the original files of the diagrams, look at Appendix G.

3.3. GUI

The first page presented to the user consists of a top bar, side navigation menu and a main section. The top bar displays the current date and the logo of VIAFit. The side menu bar includes different categories - overview, instructor, members, classes and schedule.

By default the main section displays the number of currently added members, instructors and classes in the gym. As the user navigates through the sidebar, the content of the main section changes accordingly.

The instructors, members and classes panes consist of similar layouts.



Current Members: 5

Instructors in ViaFit: 3

Current Classes: 5

Figure 8. GUI overview screen

These three panes have two sections. Register section, opened by default, and a Find and edit section. The Find and edit section also fulfills the requirement of removing.

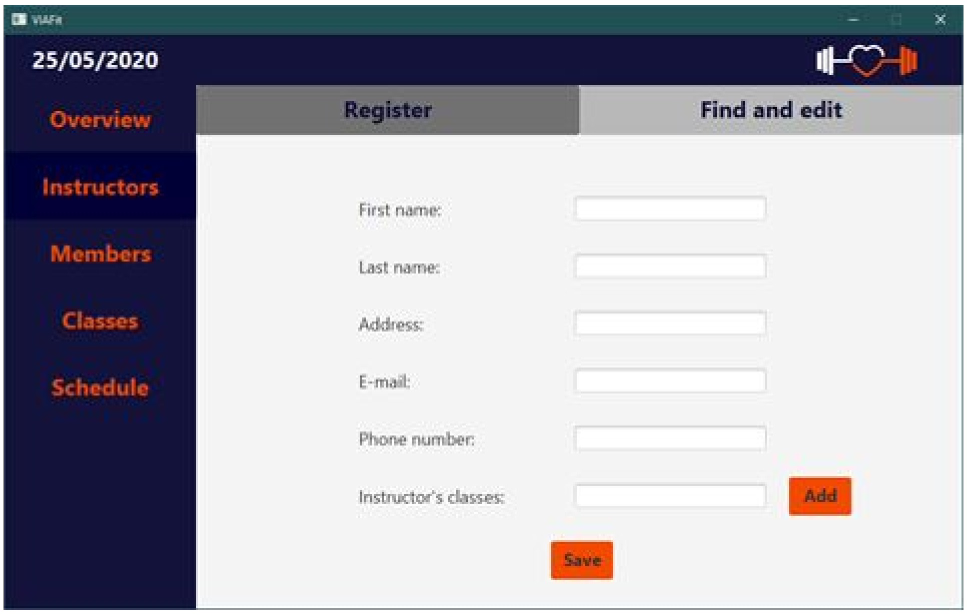


Figure 9. GUI register instructor screen

The schedule pane includes two sections, Schedule a class, displayed by default and the Display, edit and export section.

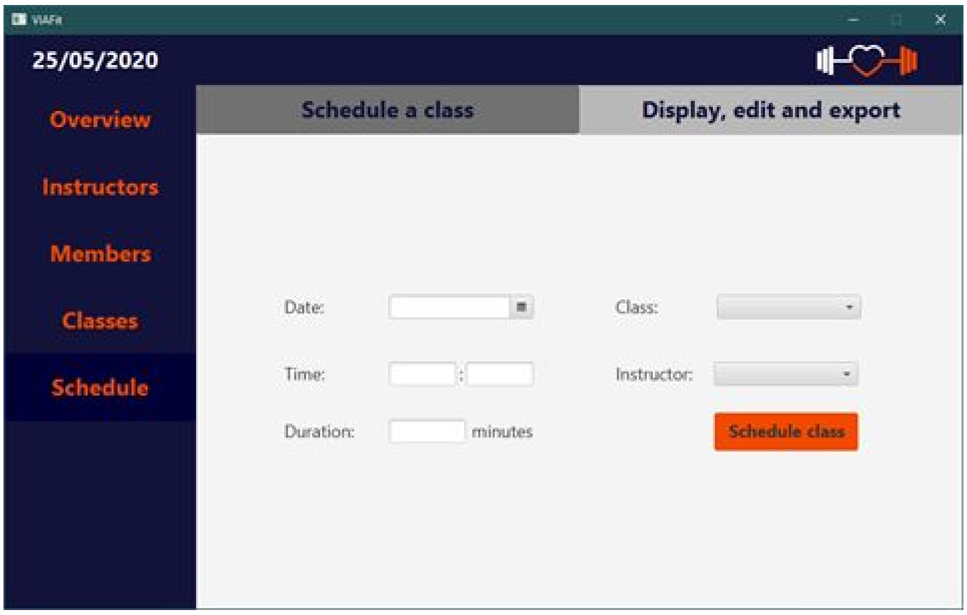


Figure 10. GUI schedule a class screen

The Display, edit and export section contains five different tabs representing the five interactions with the schedule - Edit, Remove, Sign up a Member, Cancel a Member’s Attendance and Export.

After using the first pane for finding a scheduled class, the user can go to the five previously mentioned tabs and use them. For further information on the graphical user interface, see the Appendix H.

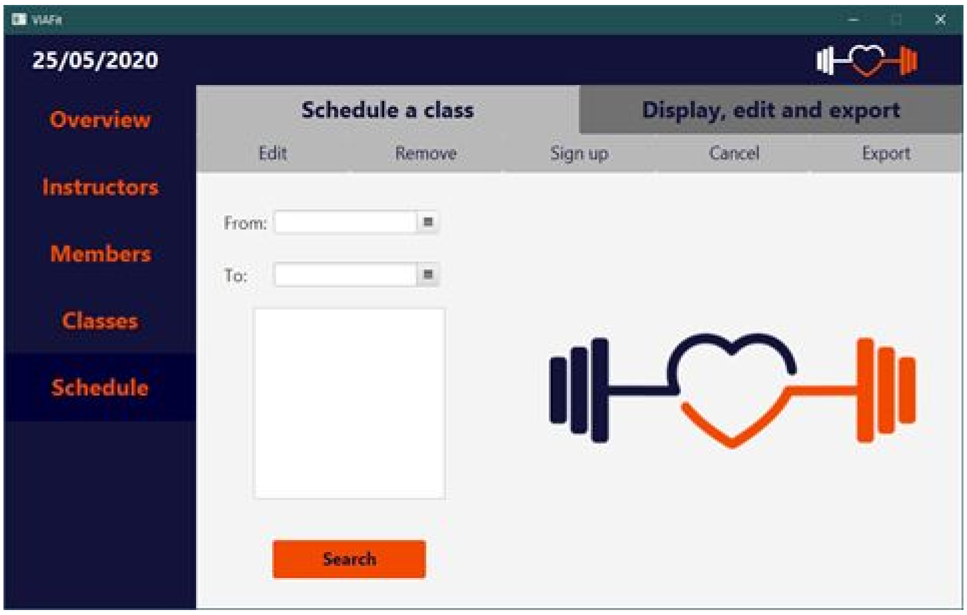


Figure 11. GUI find scheduled class screen

3.4. Website

The design process of the website started by creating wireframes, which were used as a base for putting together the layout of the website. To see the wireframes for the website, look at Appendix I.

After the wireframes have been finished, a colour palette was chosen.

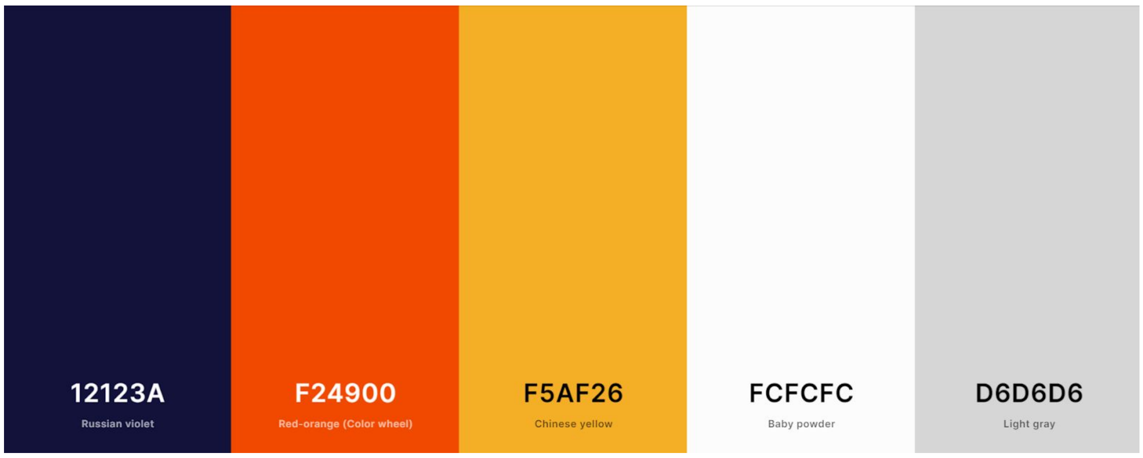


Figure 12. Color palette

The desktop layout consists of a navigation bar placed at the top of the document, main section and a footer. The navigation bar is split into five different categories, providing the user with easy access to the different pages.

The division of categories has been done, based on their content. Each of them has its own purpose and functionality. The third category, Schedule, has been specifically designed, in order to be compatible with importing external data. The navigation bar has been made fixed to provide the user with access to the different pages at all times.

The principals of the desktop layout were kept in the mobile version. The components of the website have been resized and adjusted according to given resolution. The same method will be implemented when it comes to creating a schedule.

The font family used for the website is the default Helvetica Neue from the Bootstrap framework. The sans-serif font was chosen due to legibility and readability (Harshite Arora, 2018).

4. Implementation

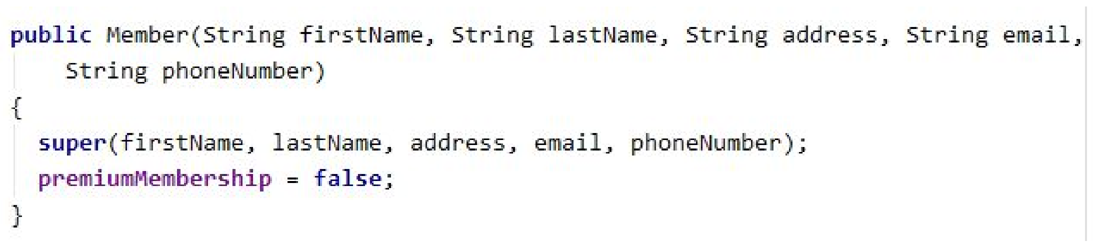
[all group members]

Based on the design outputs, an implementation process followed. The implementation process began by implementing the basic classes from the class diagram.

4.1. GUI

As the instructor and member classes have common fields and methods, an abstract class called Person was created.

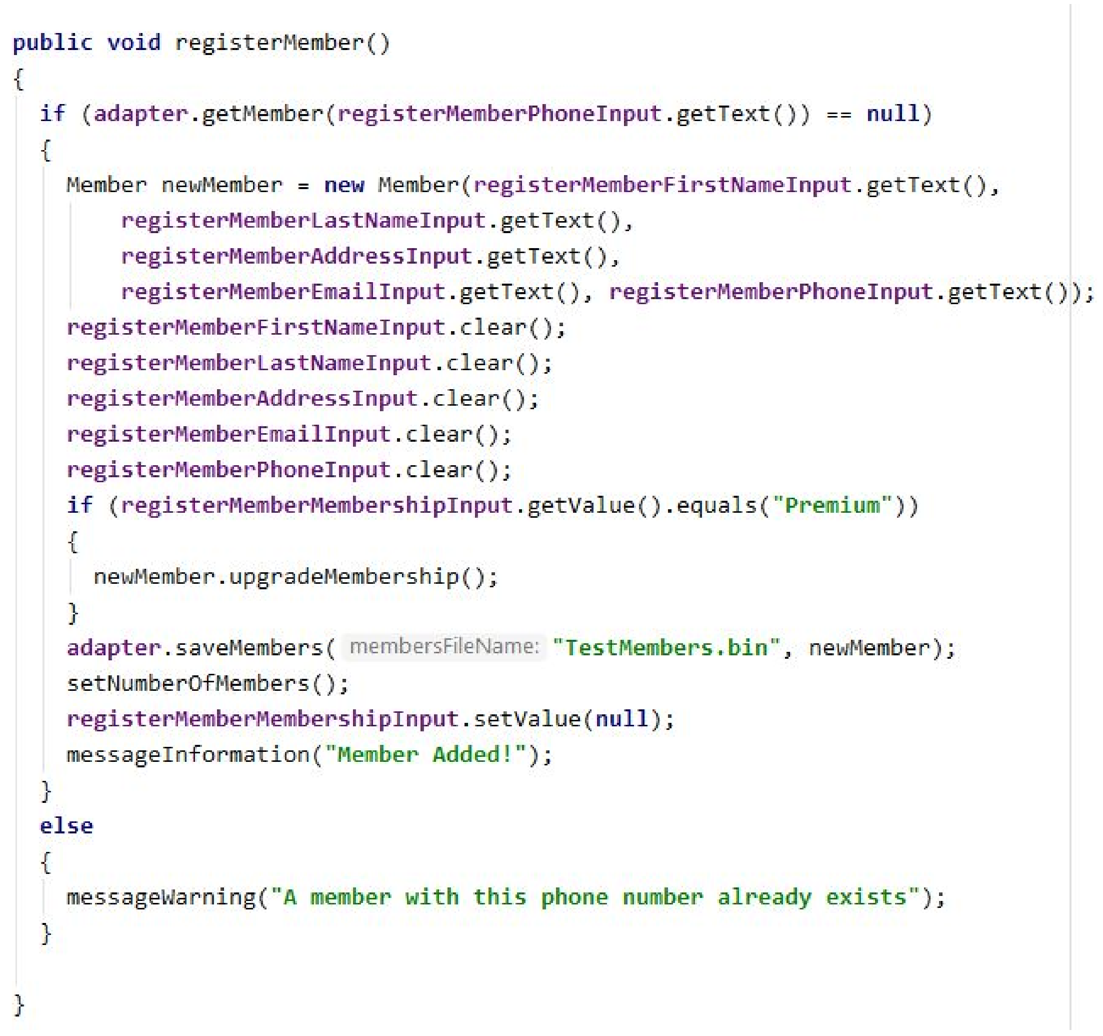
4.1.1. Registering Member



```
public Member(String firstName, String lastName, String address, String email,
    String phoneNumber)
{
  super(firstName, lastName, address, email, phoneNumber);
  premiumMembership = false;
}
```

Figure 13. Constructor for Member class

The creation of Member instances includes proving a phone number which serves as a unique identifier. Therefore, the registerMember() method in the GUIController class needs to perform a check if the member which is about to be created has a unique phone number. A new member is registered in the system if the condition in the decision structure is fulfilled.

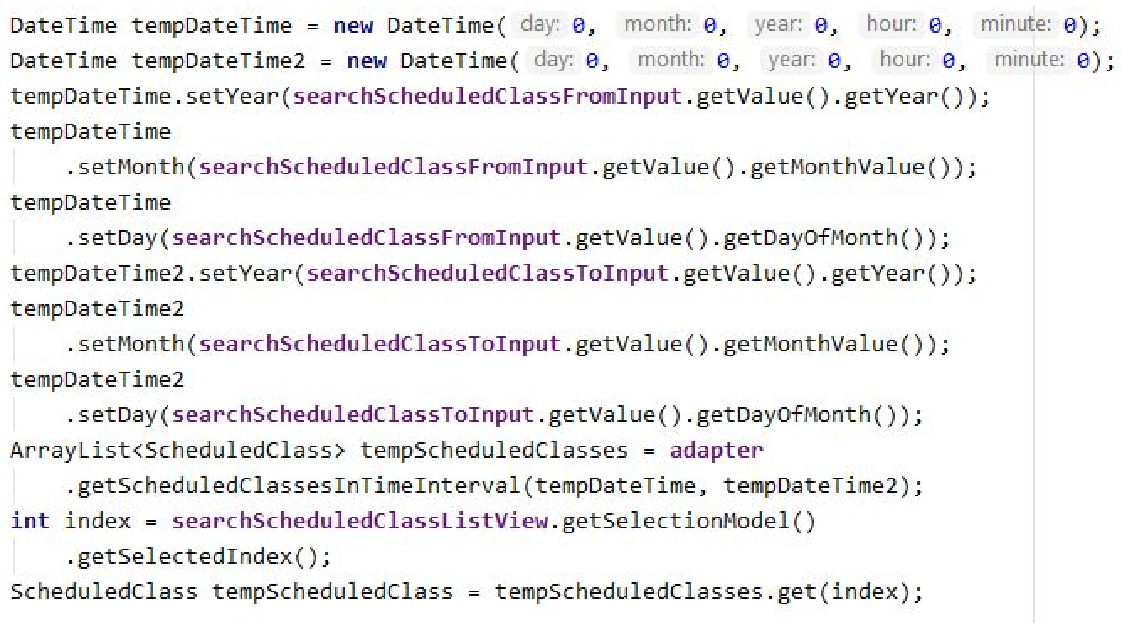


```java
public void registerMember()
{
  if (adapter.getMember(registerMemberPhoneInput.getText()) == null)
  {
    Member newMember = new Member(registerMemberFirstNameInput.getText(),
        registerMemberLastNameInput.getText(),
        registerMemberAddressInput.getText(),
        registerMemberEmailInput.getText(), registerMemberPhoneInput.getText());
    registerMemberFirstNameInput.clear();
    registerMemberLastNameInput.clear();
    registerMemberAddressInput.clear();
    registerMemberEmailInput.clear();
    registerMemberPhoneInput.clear();
    if (registerMemberMembershipInput.getValue().equals("Premium"))
    {
      newMember.upgradeMembership();
    }
    adapter.saveMembers( membersFileName: "TestMembers.bin", newMember);
    setNumberOfMembers();
    registerMemberMembershipInput.setValue(null);
    messageInformation("Member Added!");
  }
  else
  {
    messageWarning("A member with this phone number already exists");
  }

}
```

Figure 14. Register member method in GUIController

4.1.2. Searching for scheduled classes

Two objects of type DateTime with all fields initialized with zero and then overwritten by the user’s input. An ArrayList<ScheduledClass> has been extracted from the binary file based on the two DateTime objects which represent the time interval. The index variable is used to keep the position of the scheduled class selected from the preview list and retrieve the object from the ArrayList<ScheduledClass>.

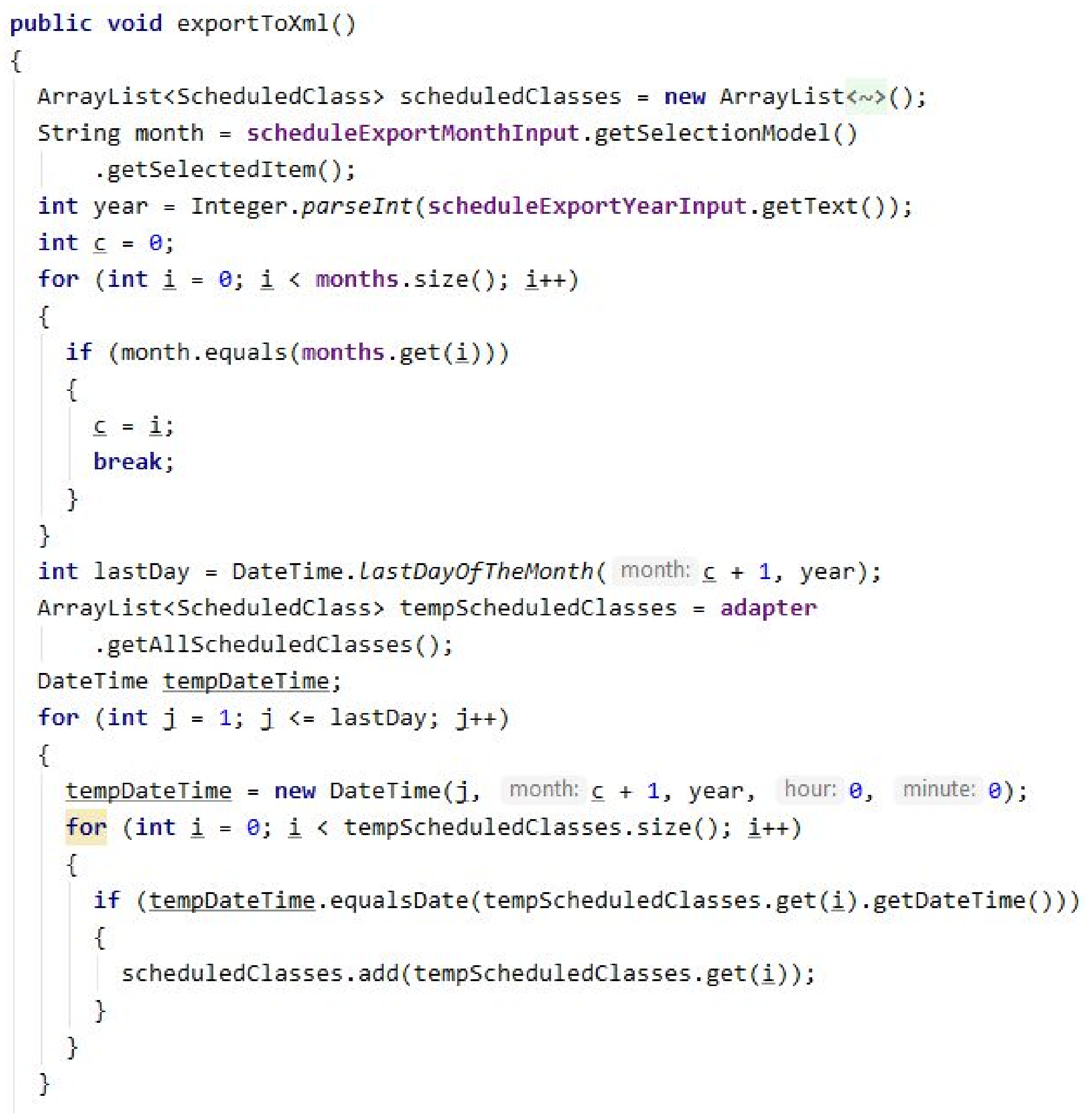


```java
DateTime tempDateTime = new DateTime( day: 0,  month: 0,  year: 0,  hour: 0,  minute: 0);
DateTime tempDateTime2 = new DateTime( day: 0,  month: 0,  year: 0,  hour: 0,  minute: 0);
tempDateTime.setYear(searchScheduledClassFromInput.getValue().getYear());
tempDateTime
    .setMonth(searchScheduledClassFromInput.getValue().getMonthValue());
tempDateTime
    .setDay(searchScheduledClassFromInput.getValue().getDayOfMonth());
tempDateTime2.setYear(searchScheduledClassToInput.getValue().getYear());
tempDateTime2
    .setMonth(searchScheduledClassToInput.getValue().getMonthValue());
tempDateTime2
    .setDay(searchScheduledClassToInput.getValue().getDayOfMonth());
ArrayList<ScheduledClass> tempScheduledClasses = adapter
    .getScheduledClassesInTimeInterval(tempDateTime, tempDateTime2);
int index = searchScheduledClassListView.getSelectionModel()
    .getSelectedIndex();
ScheduledClass tempScheduledClass = tempScheduledClasses.get(index);
```

Figure 15. Code block for searching scheduled classes

4.1.3. Exporting to XML

The method exportToXml() consists of two parts. The first part includes the functionality for getting the input of month and year from the user interface. Once the value of the month is retrieved from the ArrayList of months, the last day of the month is calculated by calling the lastDayOfTheMonth() method. Afterwards, the first and last day of the month is used to create two DateTime objects representing a time interval. All the scheduled classes from that interval are collected.

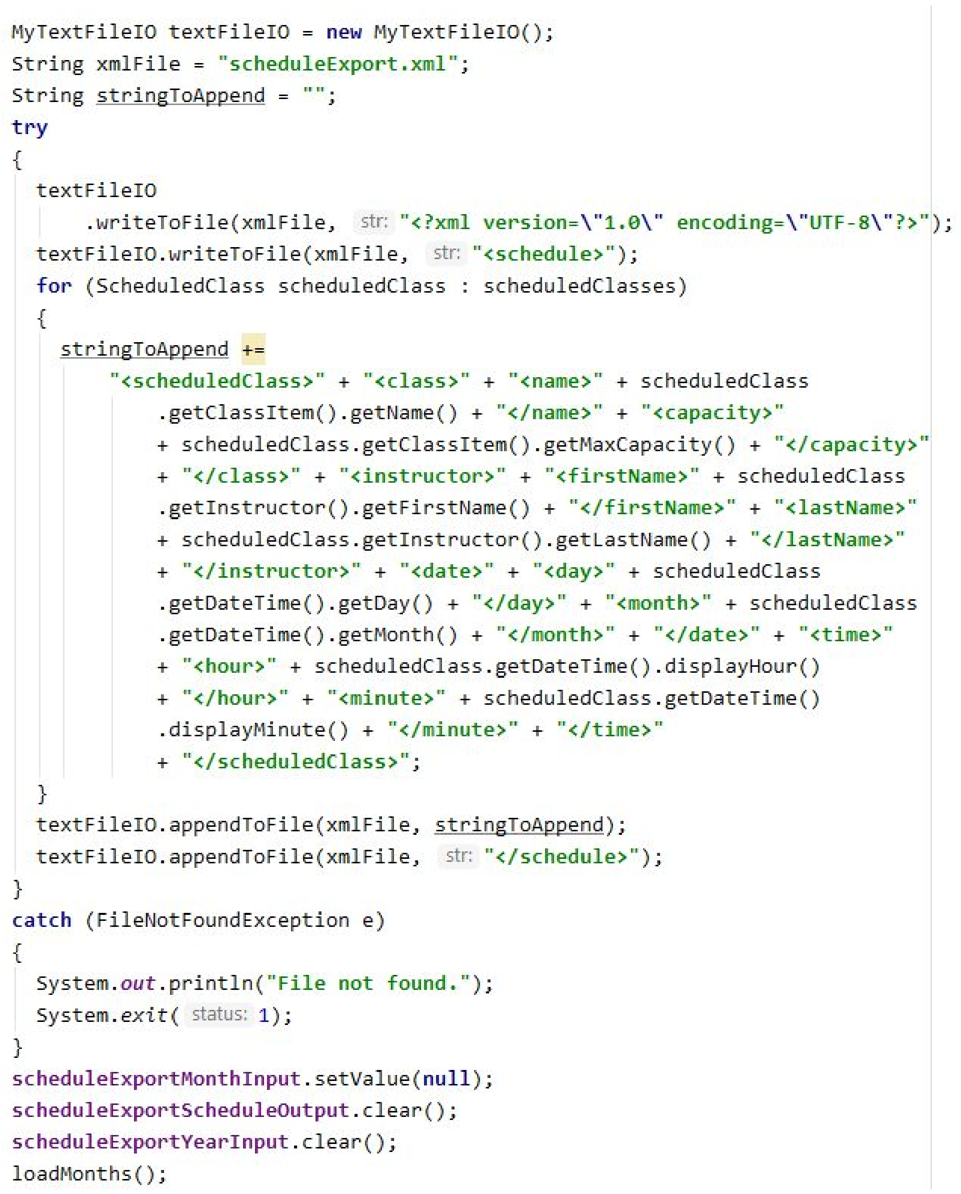


```
public void exportToXml()
{
  ArrayList<ScheduledClass> scheduledClasses = new ArrayList<~>();
  String month = scheduleExportMonthInput.getSelectionModel()
      .getSelectedItem();
  int year = Integer.parseInt(scheduleExportYearInput.getText());
  int c = 0;
  for (int i = 0; i < months.size(); i++)
  {
    if (month.equals(months.get(i)))
    {
      c = i;
      break;
    }
  }
  int lastDay = DateTime.LastDayOfTheMonth( month: c + 1, year);
  ArrayList<ScheduledClass> tempScheduledClasses = adapter
      .getAllScheduledClasses();
  DateTime tempDateTime;
  for (int j = 1; j <= lastDay; j++)
  {
    tempDateTime = new DateTime(j,  month: c + 1, year,  hour: 0,  minute: 0);
    for (int i = 0; i < tempScheduledClasses.size(); i++)
    {
      if (tempDateTime.equalsDate(tempScheduledClasses.get(i).getDateTime()))
      {
        scheduledClasses.add(tempScheduledClasses.get(i));
      }
    }
  }
```

Figure 16. Export to XML method in GUIController class, part 1

The second part of the exportToXml() method handles the functionality of writing the ArrayList of scheduled classes to a string structured as an XML format. This is possible by using the methods from MyTextFileIO. Finally, the string is written to an

XML file which is later used for displaying the scheduled classes on the VIAFit website. To see the source code for the software and the javadocs generated, look at Appendix J and K.



```
MyTextFileIO textFileIO = new MyTextFileIO();
String xmlFile = "scheduleExport.xml";
String stringToAppend = "";
try
{
  textFileIO
      .writeToFile(xmlFile,  str: "<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
  textFileIO.writeToFile(xmlFile,  str: "<schedule>");
  for (ScheduledClass scheduledClass : scheduledClasses)
  {
    stringToAppend +=
        "<scheduledClass>" + "<class>" + "<name>" + scheduledClass
            .getClassItem().getName() + "</name>" + "<capacity>"
            + scheduledClass.getClassItem().getMaxCapacity() + "</capacity>"
            + "</class>" + "<instructor>" + "<firstName>" + scheduledClass
            .getInstructor().getFirstName() + "</firstName>" + "<lastName>"
            + scheduledClass.getInstructor().getLastName() + "</lastName>"
            + "</instructor>" + "<date>" + "<day>" + scheduledClass
            .getDateTime().getDay() + "</day>" + "<month>" + scheduledClass
            .getDateTime().getMonth() + "</month>" + "</date>" + "<time>"
            + "<hour>" + scheduledClass.getDateTime().displayHour()
            + "</hour>" + "<minute>" + scheduledClass.getDateTime()
            .displayMinute() + "</minute>" + "</time>"
            + "</scheduledClass>";
  }
  textFileIO.appendToFile(xmlFile, stringToAppend);
  textFileIO.appendToFile(xmlFile,  str: "</schedule>");
}
catch (FileNotFoundException e)
{
  System.out.println("File not found.");
  System.exit( status: 1);
}
scheduleExportMonthInput.setValue(null);
scheduleExportScheduleOutput.clear();
scheduleExportYearInput.clear();
loadMonths();
```

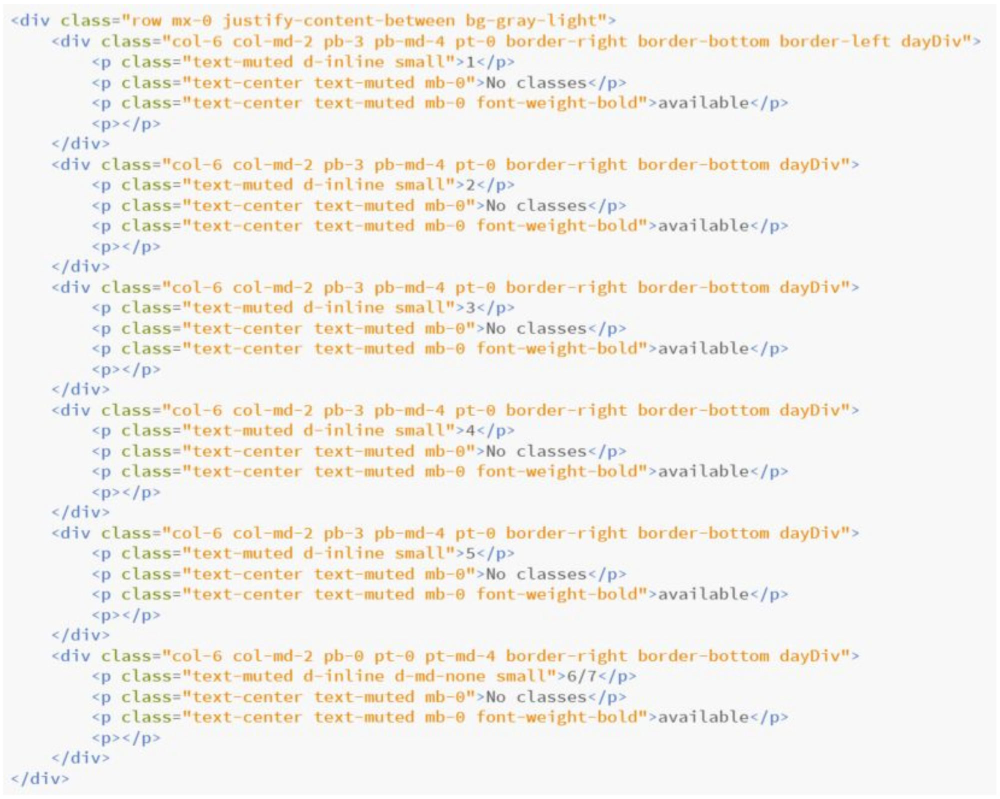
Figure 17. Export to XML method in GUIController class, part 2

4.2. Website

4.2.1. Content and presentation layer

To display the scheduled classes stored in the XML file, a calendar table has been created using HTML5 and Bootstrap 4. Divisions with a predefined layout classes of ‘row’ and ‘column’ have been used to build the layout for the monthly schedule.

As shown on the figure below, figure 18, each week of the month is being displayed by using division with ‘row’ class. The division also has the bootstrap utility class ‘justify-content-between’ used to to evenly distribute the items of the row on the main axis (Chris Coyier, 2020).



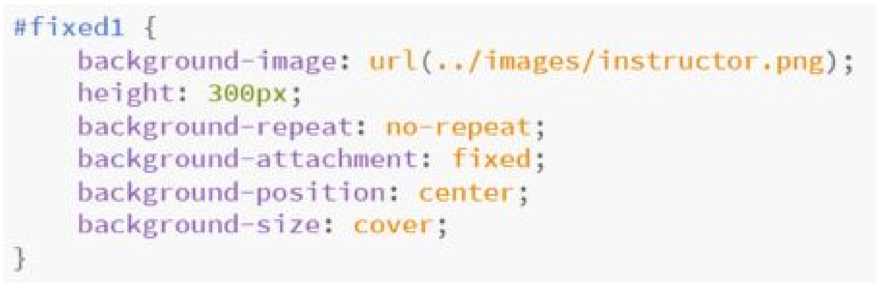
```
<div class="row mx-0 justify-content-between bg-gray-light">
    <div class="col-6 col-md-2 pb-3 pb-md-4 pt-0 border-right border-bottom border-left dayDiv">
        <p class="text-muted d-inline small">1</p>
        <p class="text-center text-muted mb-0">No classes</p>
        <p class="text-center text-muted mb-0 font-weight-bold">available</p>
        <p></p>
    </div>
    <div class="col-6 col-md-2 pb-3 pb-md-4 pt-0 border-right border-bottom dayDiv">
        <p class="text-muted d-inline small">2</p>
        <p class="text-center text-muted mb-0">No classes</p>
        <p class="text-center text-muted mb-0 font-weight-bold">available</p>
        <p></p>
    </div>
    <div class="col-6 col-md-2 pb-3 pb-md-4 pt-0 border-right border-bottom dayDiv">
        <p class="text-muted d-inline small">3</p>
        <p class="text-center text-muted mb-0">No classes</p>
        <p class="text-center text-muted mb-0 font-weight-bold">available</p>
        <p></p>
    </div>
    <div class="col-6 col-md-2 pb-3 pb-md-4 pt-0 border-right border-bottom dayDiv">
        <p class="text-muted d-inline small">4</p>
        <p class="text-center text-muted mb-0">No classes</p>
        <p class="text-center text-muted mb-0 font-weight-bold">available</p>
        <p></p>
    </div>
    <div class="col-6 col-md-2 pb-3 pb-md-4 pt-0 border-right border-bottom dayDiv">
        <p class="text-muted d-inline small">5</p>
        <p class="text-center text-muted mb-0">No classes</p>
        <p class="text-center text-muted mb-0 font-weight-bold">available</p>
        <p></p>
    </div>
    <div class="col-6 col-md-2 pb-0 pt-0 pt-md-4 border-right border-bottom dayDiv">
        <p class="text-muted d-inline d-md-none small">6/7</p>
        <p class="text-center text-muted mb-0">No classes</p>
        <p class="text-center text-muted mb-0 font-weight-bold">available</p>
        <p></p>
    </div>
</div>
```

Figure 18. HTML for row in calendar

Afterwards, each day of the week is represented as a column using the ‘col’ layout class. The layout of the columns and the space they take is controlled by the layout breakpoint classes provided by Bootstrap 4 (getbootstrap, 2020). This makes it possible to easily bring the layout of the columns to mobile.

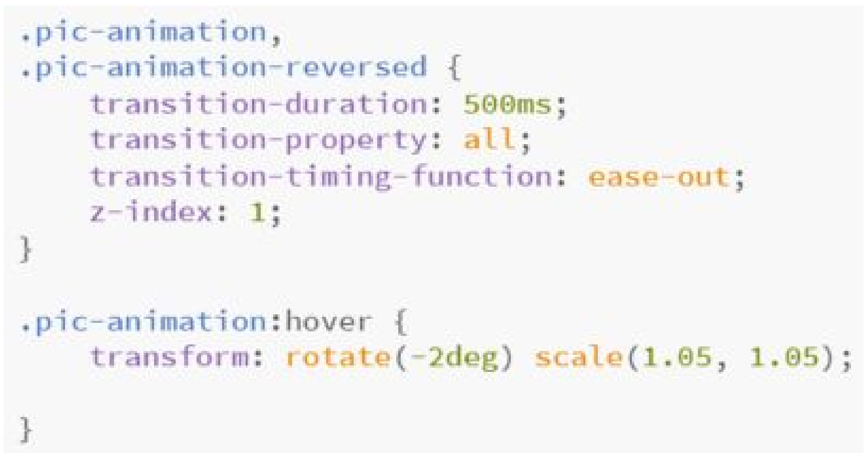
On mobile, each of the columns takes the space of 6 out of 12 possible per row. While on medium screen and above each column takes the space of 1 out of 12, aligning the days of an entire week on one row.

Effects such as animations and fixed background images are achieved by writing plain CSS since the Bootstrap 4 framework does not provide utility classes for them (getbootstrap, 2020).



```
#fixed1 {
    background-image: url(../images/instructor.png);
    height: 300px;
    background-repeat: no-repeat;
    background-attachment: fixed;
    background-position: center;
    background-size: cover;
}
```

Figure 19. CSS styles for fixed background

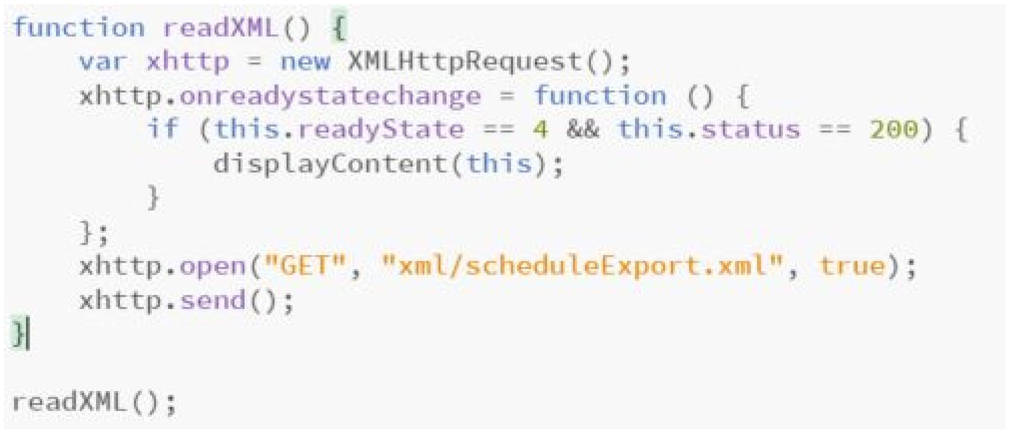


```
.pic-animation,
.pic-animation-reversed {
    transition-duration: 500ms;
    transition-property: all;
    transition-timing-function: ease-out;
    z-index: 1;
}

.pic-animation:hover {
    transform: rotate(-2deg) scale(1.05, 1.05);

}
```

Figure 20. CSS styles animation

4.2.2. Behavior layer

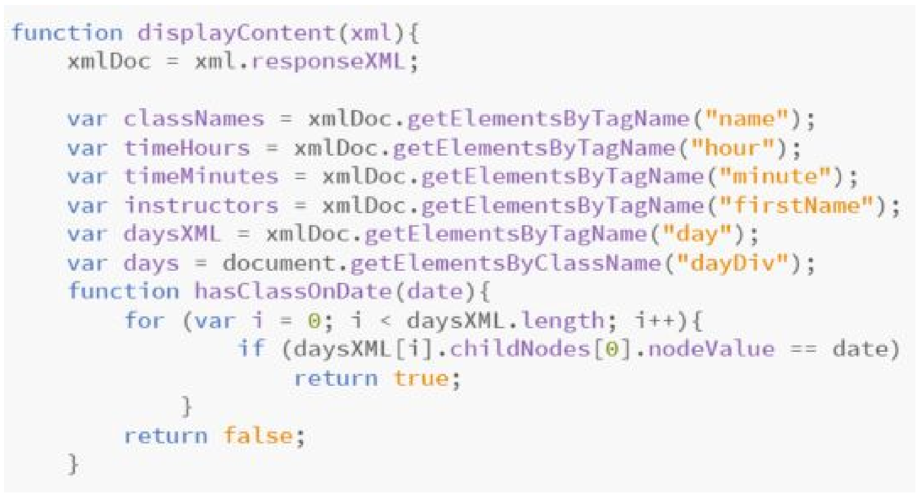
Reading content from an external XML file generated by the GUI when the user exports the monthly schedule starts by sending an xml http request.



```
function readXML() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readyState == 4 && this.status == 200) {
            displayContent(this);
        }
    };
    xhttp.open("GET", "xml/scheduleExport.xml", true);
    xhttp.send();
}

readXML();
```

Figure 21. JavaScript for reading XML

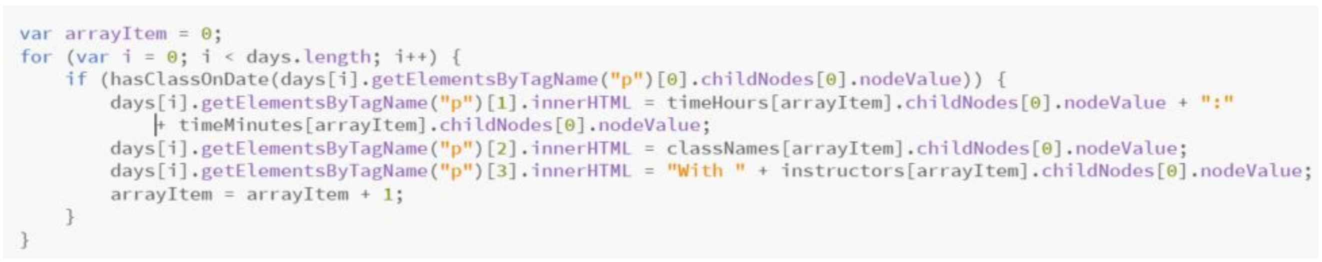
The function displayContent goes through the xml response retrieving the information from the tags - name, hour, minute, firstName and day. The method hasClassOnDate checks if there is correlation between the day value xml file and the displayed calendar.



```
function displayContent(xml){
    xmlDoc = xml.responseXML;

    var classNames = xmlDoc.getElementsByTagName("name");
    var timeHours = xmlDoc.getElementsByTagName("hour");
    var timeMinutes = xmlDoc.getElementsByTagName("minute");
    var instructors = xmlDoc.getElementsByTagName("firstName");
    var daysXML = xmlDoc.getElementsByTagName("day");
    var days = document.getElementsByClassName("dayDiv");
    function hasClassOnDate(date){
        for (var i = 0; i < daysXML.length; i++){
                if (daysXML[i].childNodes[0].nodeValue == date)
                    return true;
            }
        return false;
    }
```

Figure 22. JavaScript for displaying XML content, part 1

The following loop iterates through the days in the calendar, checking if there is a scheduled class on the particular date. To see the source code for the website, look at Appendix L.



```
var arrayItem = 0;
for (var i = 0; i < days.length; i++) {
    if (hasClassOnDate(days[i].getElementsByTagName("p")[0].childNodes[0].nodeValue)) {
        days[i].getElementsByTagName("p")[1].innerHTML = timeHours[arrayItem].childNodes[0].nodeValue + ":"
            + timeMinutes[arrayItem].childNodes[0].nodeValue;
        days[i].getElementsByTagName("p")[2].innerHTML = classNames[arrayItem].childNodes[0].nodeValue;
        days[i].getElementsByTagName("p")[3].innerHTML = "With " + instructors[arrayItem].childNodes[0].nodeValue;
        arrayItem = arrayItem + 1;
    }
}
```

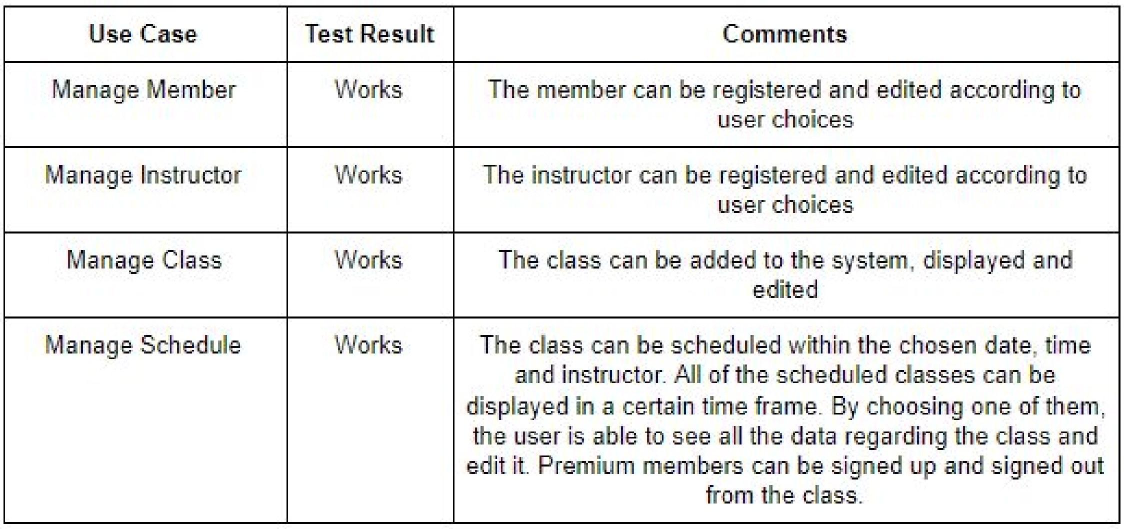
Figure 23. JavaScript for displaying XML content, part 2

5. Test

[all group members]

Soon after the implementation of the website and GUI was finished, a testing on the system was performed to ensure all of the use case descriptions are fulfilled.

The system was tested by comparing the current functionality of the developed software with the previously written use case descriptions. The outcome of the testing is summarized in the following table.



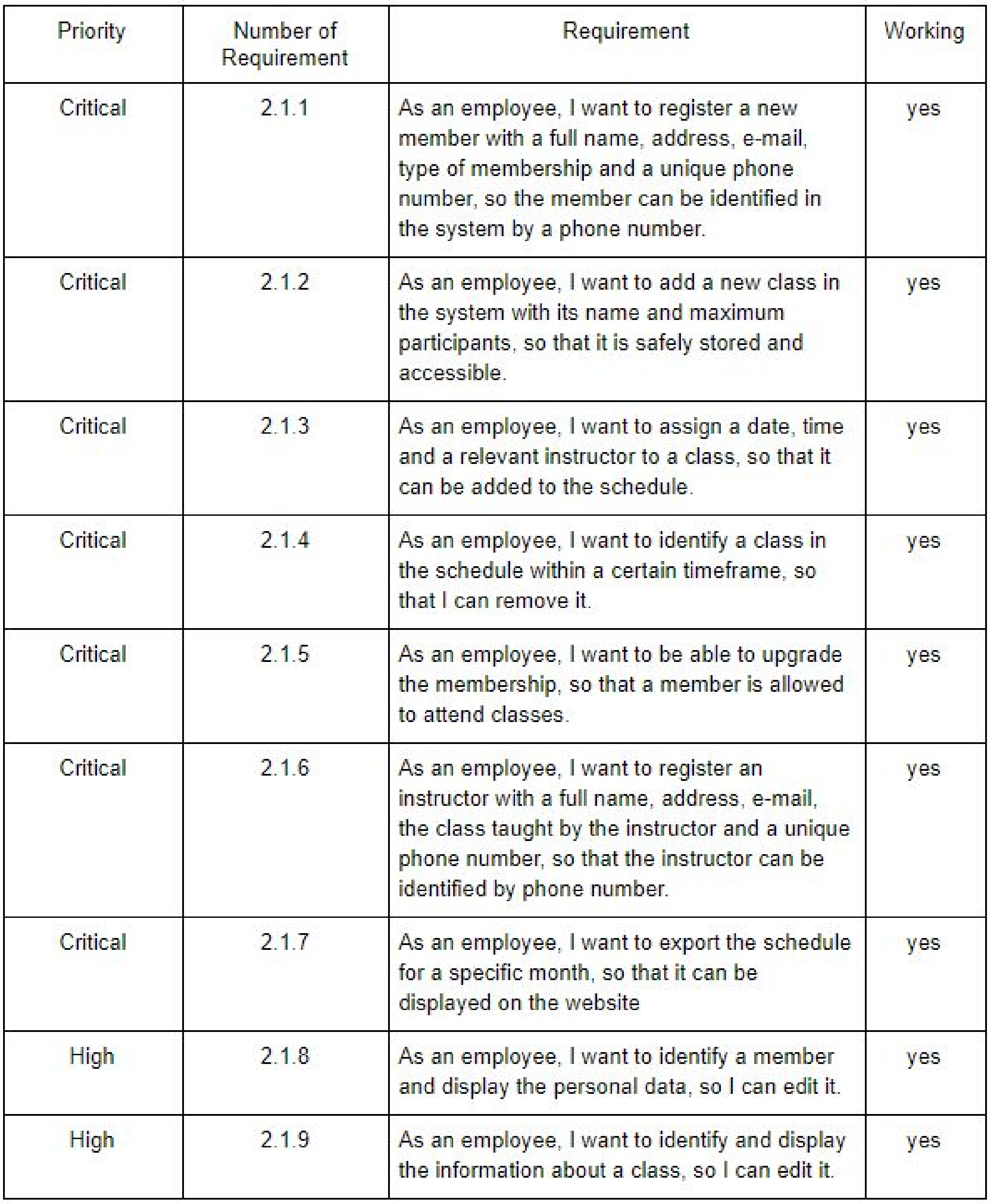
Use Case	Test Result	Comments
Manage Member	Works	The member can be registered and edited according to user choices
Manage Instructor	Works	The instructor can be registered and edited according to user choices
Manage Class	Works	The class can be added to the system, displayed and edited
Manage Schedule	Works	The class can be scheduled within the chosen date, time and instructor. All of the scheduled classes can be displayed in a certain time frame. By choosing one of them, the user is able to see all the data regarding the class and edit it. Premium members can be signed up and signed out from the class.

Figure 24. Test Table

6. Result

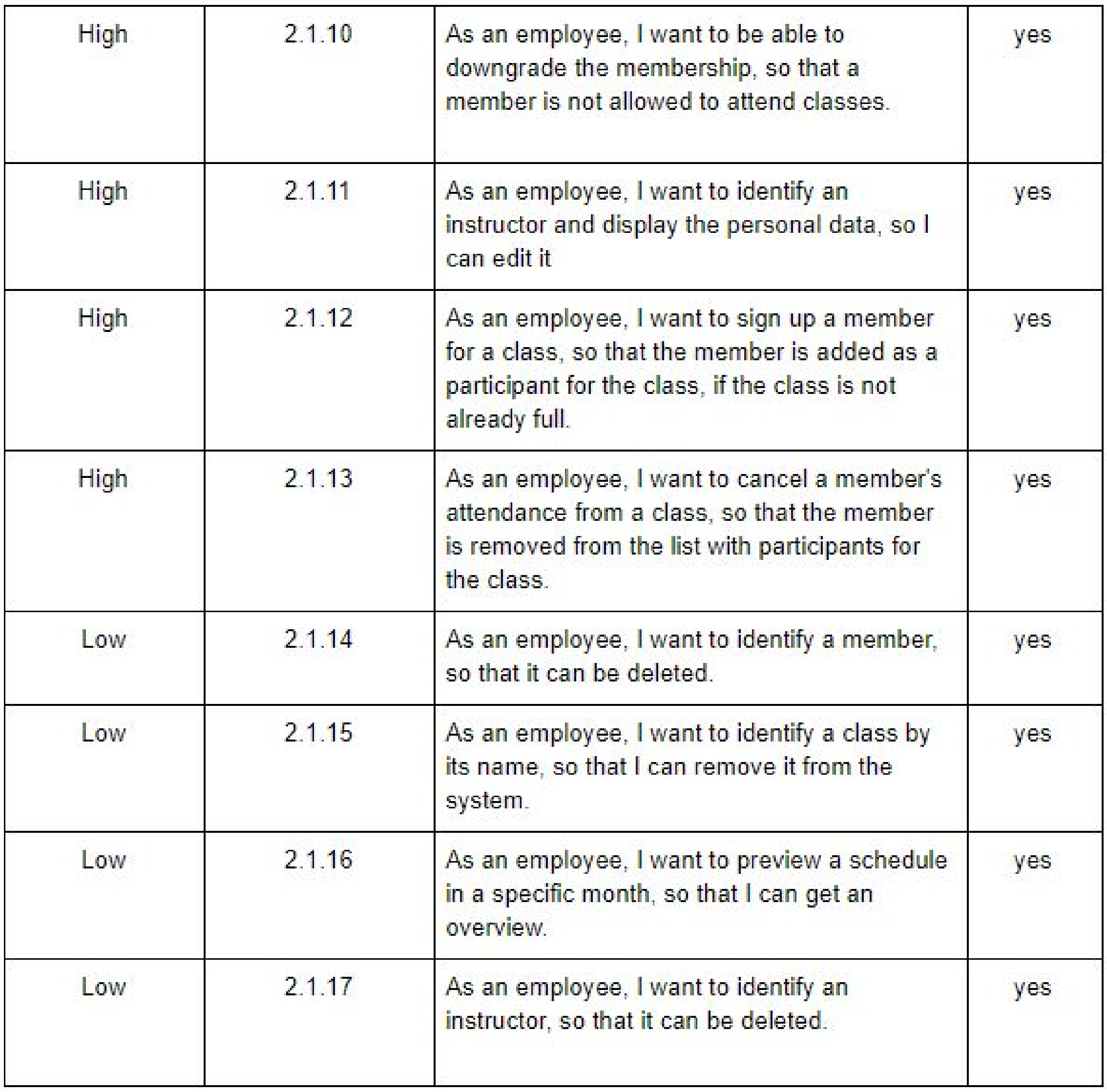
[all group members]

To represent the fulfilment of the requirements the following table has been made.



Priority	Number of Requirement	Requirement	Working
Critical	2.1.1	As an employee, I want to register a new member with a full name, address, e-mail, type of membership and a unique phone number, so the member can be identified in the system by a phone number.	yes
Critical	2.1.2	As an employee, I want to add a new class in the system with its name and maximum participants, so that it is safely stored and accessible.	yes
Critical	2.1.3	As an employee, I want to assign a date, time and a relevant instructor to a class, so that it can be added to the schedule.	yes
Critical	2.1.4	As an employee, I want to identify a class in the schedule within a certain timeframe, so that I can remove it.	yes
Critical	2.1.5	As an employee, I want to be able to upgrade the membership, so that a member is allowed to attend classes.	yes
Critical	2.1.6	As an employee, I want to register an instructor with a full name, address, e-mail, the class taught by the instructor and a unique phone number, so that the instructor can be identified by phone number.	yes
Critical	2.1.7	As an employee, I want to export the schedule for a specific month, so that it can be displayed on the website	yes
High	2.1.8	As an employee, I want to identify a member and display the personal data, so I can edit it.	yes
High	2.1.9	As an employee, I want to identify and display the information about a class, so I can edit it.	yes

Figure 25. Result Table, part 1



High	2.1.10	As an employee, I want to be able to downgrade the membership, so that a member is not allowed to attend classes.	yes
High	2.1.11	As an employee, I want to identify an instructor and display the personal data, so I can edit it	yes
High	2.1.12	As an employee, I want to sign up a member for a class, so that the member is added as a participant for the class, if the class is not already full.	yes
High	2.1.13	As an employee, I want to cancel a member's attendance from a class, so that the member is removed from the list with participants for the class.	yes
Low	2.1.14	As an employee, I want to identify a member, so that it can be deleted.	yes
Low	2.1.15	As an employee, I want to identify a class by its name, so that I can remove it from the system.	yes
Low	2.1.16	As an employee, I want to preview a schedule in a specific month, so that I can get an overview.	yes
Low	2.1.17	As an employee, I want to identify an instructor, so that it can be deleted.	yes

Figure 26. Result Table, part 2

Additionally to the requirements presented in the analysis part, a screen saver and an overview pane has been implemented.

7. Discussion

[all group members]

The implementation process started by implementing all methods and fields from the class diagram. Later in the process when the GUIController class was being implemented a few problems arose.

In the basic class Instructor, there was a missing method for retrieving an ArrayList of classes. The method was added to the class since it was needed for managing the data regarding instructors.

Based on the requirements regarding editing existing data, a few methods needed to be added to the FitnessCenterFileAdapter class. Initially, the file was created with methods for saving, removing and retrieving data from the binary file. During the implementation of the GUIController class, additional methods for editing the existing data needed to be added.

8. Conclusion

[all group members]

To sum up, the goal of the project was to create a reliable software, which will help VIAFit with storing data and exporting it to a newly designed webpage.

The analysis started with outlining the requirements and every following chapter was based upon them. While going through the creation of the software, every aspect of the project was formed with an aim to fulfill all requirements.

Based on testing the software and its capabilities, it has been concluded that every requirement is met and additional features have been added. The end result is a system able to store data and to manage its every variable. The aim of the project has been fulfilled and all VIAFit's requirements are met.

9. Sources of Information

Harshite Arora, 2018. How typography Determines Readability: Serif vs, Sans Serif, and how to combine fonts
Available at https://www.freecodecamp.org/news/how-typography-determines-readability-serif-vs-sans-serif-and-how-to-combine-fonts-629a51ad8cce/
[Accessed May 27, 2020]

Chris Coyier, 2020. A complete guide to flexbox.
Available at https://css-tricks.com/snippets/css/a-guide-to-flexbox/
[Accessed May 28, 2020]

getbootstrap.com. 9 stages of conflict escalation according to Friedrich Glasl
Available at https://getbootstrap.com/docs/4.0/getting-started/introduction/
[Accessed June 1, 2020]

getbootstrap, 2020.Build fast, responsive sites with Bootstrap. [online] Available at: <https://getbootstrap.com//>
[Accessed 1 June 2020].

Tony Gaddis, 2015, *Starting out with Java: Early Objects*, Fifth edition, Harlow, England, Published by Pearson Education 2015.

Jon Duckett, 2014, *Javascript and jQuery: interactive front-end web development*, Indianapolis, Indiana, Published by John Wiley and Sons 2014.

Jon Duckett, 2011, *HTML and CSS: idesign and build websites*, Indianapolis, Indiana, Published by John Wiley and Sons 2011.

Benjamin LaGrone, 2013, *HTML and CSS: idesign and build websites*, Birmingham, Published by Packt Publishing 2013.

List of Appendices

Appendix L: Website Source Code