CS 319 - 3
GROUP 3E: COVOID-21

DESIGN REPORT ITERATION 2
PANDEMIC MANAGER

Kutay Demiray - 21901815

Gökberk Keskinkılıç - 21801666

Berke Uçar - 21902238

Yağız Yaşar - 21902951

Ömer Burak Yıldıran - 21901425

Table of Contents

Table of Contents 1

1. Introduction 2
1.1. Purpose of the System 2
1.2. Design Goals 2
1.2.1. Information Security & Privacy 2
1.2.2. Functionality 3
1.2.3. Maintainability 4

2. High-Level Software Architecture 4
2.1. Overview 4
2.2. Subsystem Decomposition 4
2.2.1. Presentation Layer 6
2.2.2. Application Layer 7
2.2.3. Database Interface Layer 9
2.2.4. Data Layer 10
2.3. Deployment Diagram 11
2.4. Hardware/Software Mapping 11
2.5. Persistent Data Management 12
2.6. Access Control and Security 13
2.7. Boundary Conditions 16
2.7.1. Initialization 16
2.7.2. Termination 17
2.7.3. Failure 17

3. Low-Level Design 17
3.1. Object Design Trade-offs 17
3.2. Final Object Design 18
3.2.1. Strategy Pattern 19
3.2.2. Singleton Pattern 20
3.3. Packages 21
3.3.1. External Packages 21
3.3.2. Internal Packages 22
3.4. Class Interfaces 24
3.5. Entity Classes 24
3.6. Control Classes 29

4. Improvement Summary 34

5. Glossary 35

6. References 36

1. Introduction

This report defines the design of the software system by explaining its purpose, and enacted non-functional requirements of the software system in the first part. In the second part, it expands the design by specifying its technical details of the software architecture, decomposition of different subsystems inside the system, relation between hardware and software, data management, information access control and security, boundary use cases. Lastly, in the third part, the low-level design which covers the object and class designs.

1.1. Purpose of the System

The purpose of the system is to develop a viable and easy to use pandemy tracking software system in Bilkent University. It is expected to be used by various users - students, academic staff, administration personnel and healthcare personnel. The system is planned to include important information and functions for these users to contribute to the pandemy tracking and present a safe environment inside the university.

1.2. Design Goals

The non-functional requirements are set according to the needs of the application’s users and targeted to make the software system more viable. These goals are detected through background research and planned to be implemented.

1.2.1. Information Security & Privacy

Since the application includes sensitive information, such as the user's HES code, location, and uses external applications for this kind of information, the users would not want their data to be leaked and reached from 3rd party groups. So, the information security issue becomes important for each user’s privacy. Any of the data should not be reached to any individual or group that users do not want to share to. Hence, the application should provide a great deal of security and privacy in its usage and implementation as well. Passwords will be required to include all character groups to improve

password complexity, also our design choices are made considering information security, MySQL and AWS are selected in line with this since this database solution has fewer security vulnerabilities in comparison with other solutions. AWS allows encryption of database instances with SSL/TSL, also different security groups can be created to assure the security robustness [10].

1.2.2. Functionality

The application is planned to be functional because the users would want to share information according to the application's capability to affect the current pandemy tracking status in the university and users' benefits. Since the university maintains a different set of facilities and provides several services for its members, interactions of university members in person should be tracked to create a safe environment for the current pandemic. Previous situation of the university requires meaningful functionalities to be implemented inside the application. The system should present only infection management related functions for the pandemy tracking, and these functions should be made sure to have a feasible use in the application.

1.2.3. Usability

As we all have experienced, unfortunately, pandemics at bigger scales tend to have a high rate of mortality. To slow down the spreading process of such a pandemic, a good management strategy and a flow of information should be maintained. Even though functionality requirements should be fulfilled at the end of the implementation, another critical thing to be considered is usability of the system. To support the information flow from users to the system and vice versa, the application should allow users to interact with itself in a simple and effective way. Besides the user interface being simple and easy to learn and navigate through the site for users with the positioning of buttons etc, the program should be basic enough to calculate the critical flows that it must provide to create a safe pandemic environment. Also, it is considered that the

interface is not intimidating for the users with naming of the buttons and assuring color palette choices.

1.2.4. Maintainability

As we experienced for the last couple of years, the pandemics have a dynamic structure. The university or government may alter the criterias, such as the duration of the quarantine period for infected various individuals, which people are regarded as contacted to the virus and as fully vaccinated etc., may change in the different stages of the pandemy according to the course of events. Therefore, the tracking statuses inside the system are planned to be made in a way that they can easily be altered on these criteria changes.

2. High-Level Software Architecture

2.1. Overview

In this section subsystem decomposition is explained by looking deep into layers that make a whole. Also hardware and software mapping is explained to describe minimum requirements to use our program. Persistent data management is explained by describing usage of the database, and also mentions which data is stored where. Access control and security mentions which user types have what kind of controls on some properties. This increases comprehension of the logic of the program. Boundary conditions explain how the program reacts in unexpected conditions.

2.2. Subsystem Decomposition

We applied a 4 layered architecture for our subsystem decomposition which consists of Presentation Layer, Application Layer, Database Interface Layer and Data Layer. It separates the program structure into 4 different sub-structures where they differ with similar packages. We thought that this architecture would complement our system to be more secured, functional, maintainable and usable.

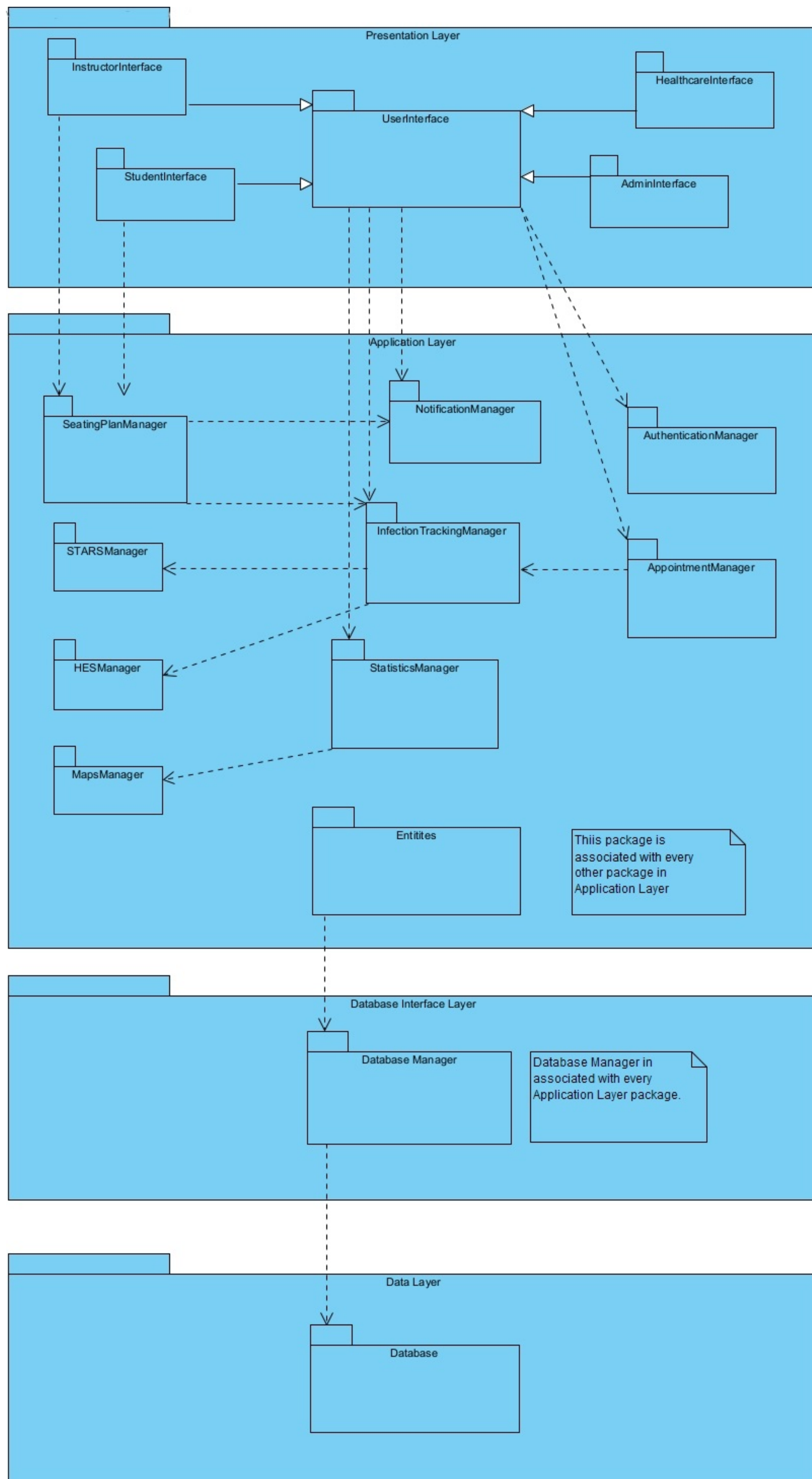

Figure 1: Subsystem Decomposition

2.2.1. Presentation Layer

Presentation Layer refers to the interaction between users and the screens, where they can communicate with the program. They can enter information to the prompted inputs, this information would be passed into the Application Layer.

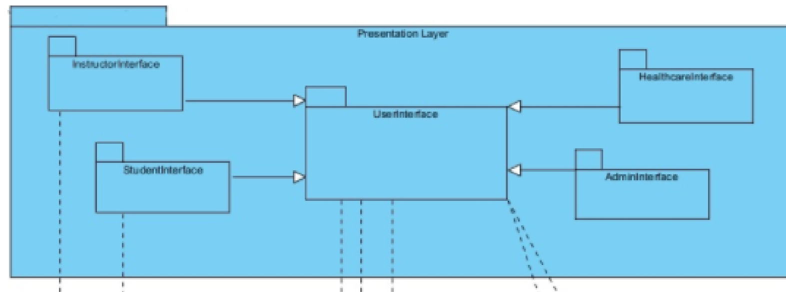

Figure 2: Presentation Layer

<u>InstructorInterface</u>

- This package consists of screens and interactions that only with instructor authorization can access and have.
- This package is generalized to the UserInterface package.
- This package is associated with the SeatingPlanManager package.

<u>StudentInterface</u>

- This package consists of screens and interactions that only with student authorization can access and have.
- This package is generalized to the UserInterface package.
- This package is associated with the SeatingPlanManager package.

<u>HealthcareInterface</u>

- This package consists of screens and interactions that only with healthcare personnel authorization can access and have.
- This package is generalized to the UserInterface package.

<u>AdminInterface</u>

- This package consists of screens and interactions that only with administrative personnel authorization can access and have.
- This package is generalized to the UserInterface package.

UserInterface

- This package consists of screens and interactions that users can access and have. This package is generalized to the UserScreens package.
- This package is associatedwithNotificationManager, InfectionTrackingManager, StatisticsManager, AuthenticationManager and AppointmentManager packages.

2.2.2. Application Layer

Application Layer consists of control object packages that can alter the data according to the retrieved data. This data from the presentation layer is processed in the application layer to the referring information according to the Presentation Layer. These control object packages communicate with Database Manager inside the Database Interface Layer.

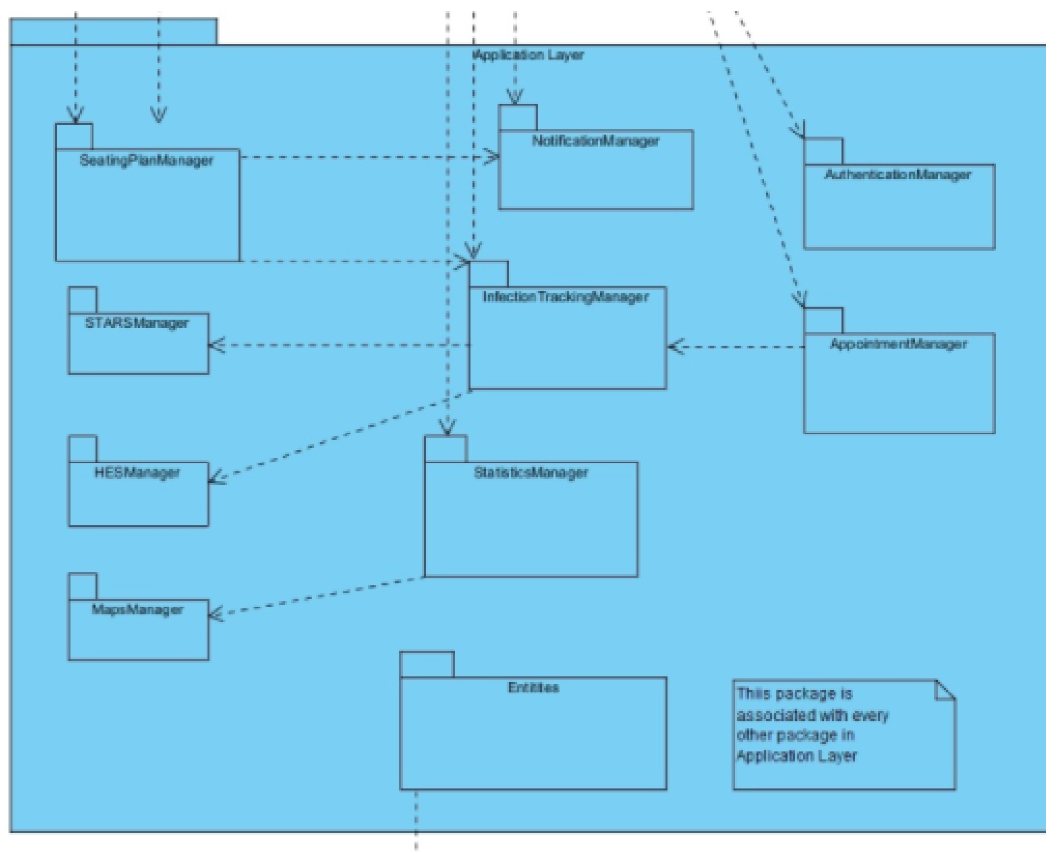

Figure 3: Application Layer

SeatingPlanManager

- This package consists of control classes related to changing the seating plan.
- This package is associated with InstructorInterface, Student Interface, NotificationManager and InfectionTrackingManager packages.

STARSManager

- This package consists of control classes related to STARS API management.
- This package is associated with the InfectionTrackingManager package.

HESManager

- This package consists of control classes related to HES API management.
- This package is associated with the InfectionTrackingManager package.

MapsManager

- This package consists of control classes related to Google Maps API management.
- This package is associated with the StatisticsManager package.

NotificationManager

- This package consists of control classes related to notifications.
- This package is associated with UserInterface and SeatingPlanManager packages.

InfectionTrackingManager

- This package consists of control classes related to change the seating plan.
- This package is associated with UserInterface, SeatingPlanManager, STARSManager, HESManager, AppointmentManager packages.

StatisticsManager

- This package consists of control classes related to statistics.
- This package is associated with UserInterface and MapsManager packages.

AuthenticationManager

- This package consists of control classes related to user authentication management.
- This package is associated with UserInterface package.

AppointmentManager

- This package consists of control classes related to appointment management.
- This package is associated with UserInterface and InfectionTrackingManager packages.

Entities

- This package consists of entities in the program.
- This package is associated with every other package in Application Layer.

2.2.3. Database Interface Layer

Database Interface Layer only contains the Database Manager which has the task for managing database instances. This is a transactional managing layer between Application Layer and Data Layer.

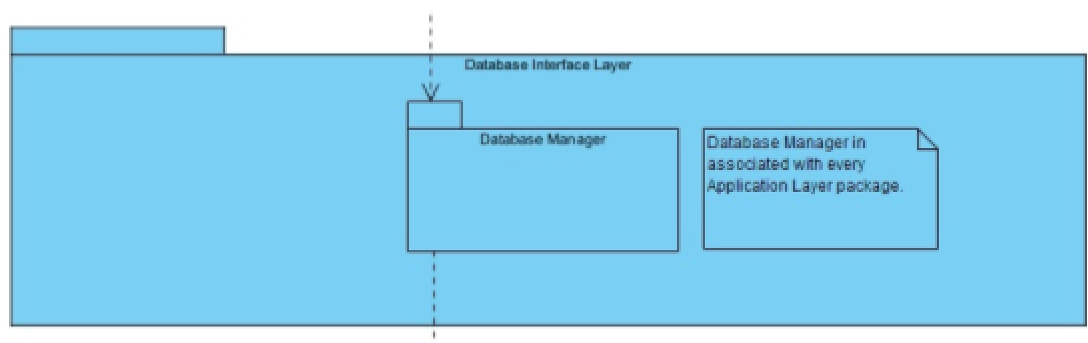

Figure 4: Database Interface Layer

DatabaseManager

- This package consists of classes that are related to database management.

- This package is associated with every other package in Application Layer and Database package.

2.2.4. Data Layer

Data Layer only contains the Database. Database is in charge of storing data, and maintaining the required information for the system. The data is read and written on the Database.

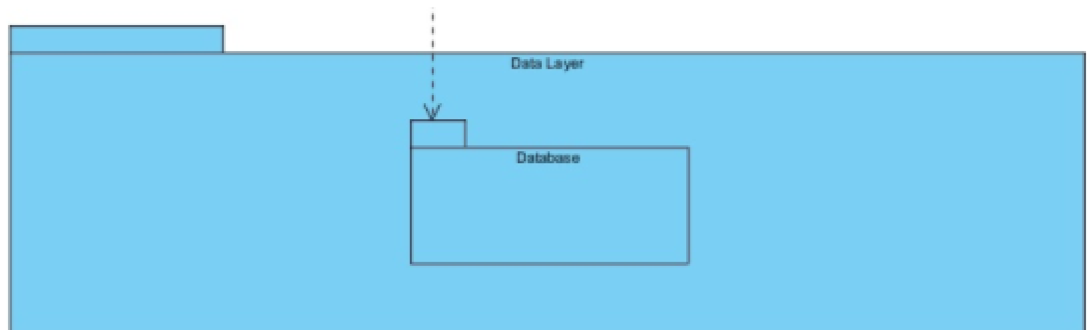

Figure 5: Data Layer

<u>Database</u>

- This package refers to the external database.
- This package is associated with Database Manager package.

2.3. Deployment Diagram

This diagram shows the interaction between client and server. Its design is very similar to subsystem decomposition. Component groups classes that work together closely and they can be classified by their type whereas artifacts represent concrete elements in the physical world.

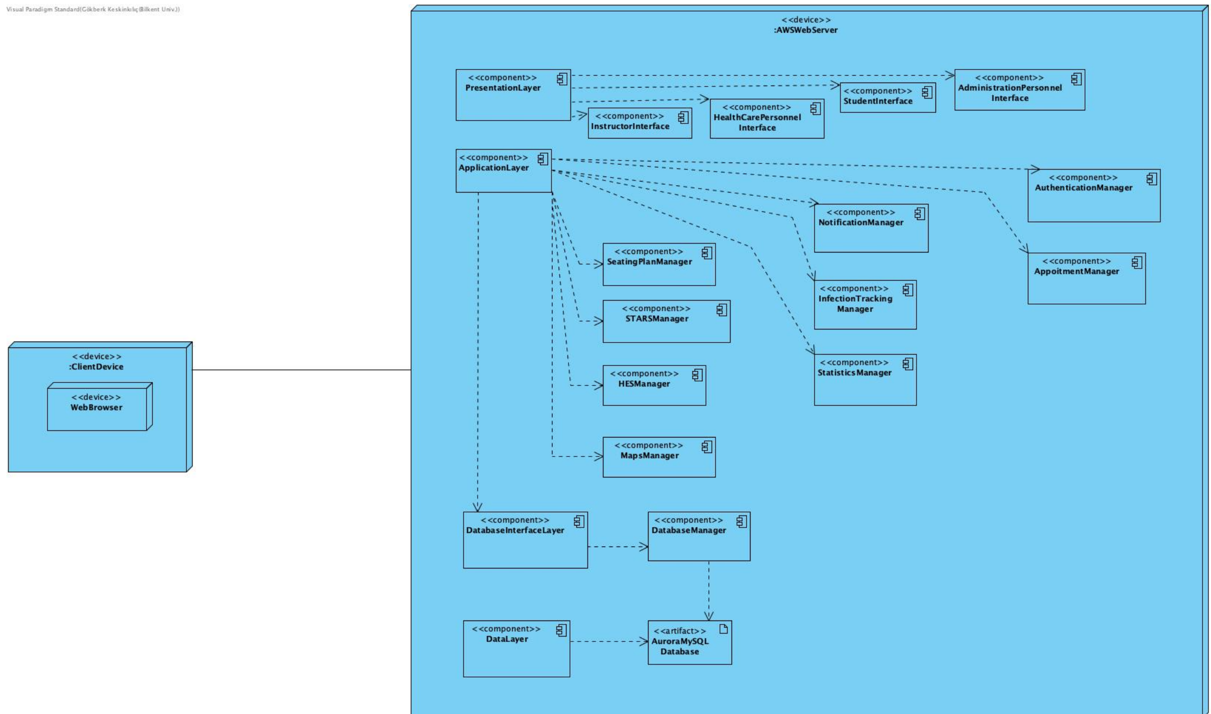

Figure 6: Deployment diagram (See https://i.imgur.com/sWpuIKe.jpg for high resolution image)

2.4. Hardware/Software Mapping

The app uses Spring Framework 2.6.0 to manage its back-end logic because of its robust nature and wide usage across the software industry. The front-end of the app is handled by React Framework 17.0.2 as it allows for a considerable amount of code reusability and yields overall better performance when compared to its alternatives. As for the database, MySQL with Amazon RDS is used due to its ease of use and maintainability. "t4g micro" type database is used from Amazon RDS. The server

properties are as follows: 1 GiB of Memory, 2 vCPUs, EBS only, 64-bit Arm platform (Amazon Graviton2 processor) [1, 2, 3].

The project does not require any specialized hardware components to be used. The app runs on the web; thus, as long as the user is equipped with hardware capable of running any modern browser, there are no other limitations to which device can be used to access it. Therefore, the app is usable on both computers and mobile phones that can run a modern browser*.

To use the app via computers, standard peripherals (such as monitor, keyboard, etc.) are required. For mobile user experience, no other accessory is needed since mobile devices are pre-equipped with the necessary input devices (touchscreen).

*Referred modern browsers include the following, but are not limited to: Google Chrome (v41.0.2272 and higher), Mozilla Firefox (v21 and higher), Microsoft Edge (any version), Safari (v6 and higher), Opera (v11 and higher). Legacy browsers such as IE8 and Safari 3 might cause incompatibility issues since used technologies contain many modern libraries with several API calls.

2.5. Persistent Data Management

MySQL will be used as the database technology since MySQL provides easy access operations during the development phase. Also, Amazon Aurora is compatible with MySQL, which allows for even improved ease of maintainability. Hence, our OOP model of the project will be easier for us to implement and manage from MySQL[3].

Users (Student, Instructor, Healthcare Personnel, Administration Personnel), Seating Plans, Appointments, and Sent Notifications will be stored in the MySQL database. Amazon AWS will be used to deploy the server online [3, 4].

Since the objects stored in the database are changeful, the requests done via the app must be dynamic. By doing so, any possible inconsistency in stored data will be eliminated. Cookies will be managed by the Spring Cookie class and create states between the pages, which is essential for an app like the one we are working with.

2.6. Access Control and Security

Pandemic manager is a web-based application, thus our app requires enhanced security and authorization. This will be provided with Spring Security at the back-end logic. Authentications will be handled by built-in support of Spring Security Authenticate. Furthermore, using Spring Security makes it possible to give the provided permissions to each user type. Lastly, Spring Security also has built-in security tools to defend the application against common types of attacks [1].

Aside from the security features of Spring Security, Amazon AWS offers one of the best protection against possible attacks (such as Denial of Service) [4].

Below the access, you can see methods of control classes that actors in the program can call directly using the website interface buttons. Entity classes are excluded, as they only contain getters and setters which are not directly called by any of the buttons. Note that the methods not shown in the access matrix are still used in the methods called with the buttons.

Security is mainly provided out-of-the-box by Amazon AWS and Spring Boot, login credentials are kept securely within the database after they got hashed. Attacks to hack other users' credentials can be blocked by Amazon GuardDuty automatically (such as Brute-Force, Dictionary Attack).

Note that "…" refers to the parameter(s) of the methods.

	Student	Instructor	Healthcare Personnel	Administration Personnel
UserManager	userSettingsScreen() openProfile() changePassword(...) changeReminder(...) changeEmail(...)	userSettingsScreen() openProfile() changePassword(...) changeReminder(...) changeEmail(...)	userSettingsScreen() openProfile() changePassword(...) changeReminder(...) changeEmail(...)	userSettingsScreen() openProfile() changePassword(...) changeReminder(...) changeEmail(...)
AppointmentManager	healthAppointmentScreen() testAppointmentScreen() facilityAppointmentScreen() checkUpForm(...) testForm(...) makeAppointment(...) showHealthTimeTable(...) showTestTimeTable(...) showFacilityTimeTable(...)	healthAppointmentScreen() testAppointmentScreen() facilityAppointmentScreen() checkUpForm(...) testForm(...) makeAppointment(...) showHealthTimeTable(...) showTestTimeTable(...) showFacilityTimeTable(...)	healthAppointmentScreen() testAppointmentScreen() facilityAppointmentScreen() checkUpForm(...) testForm(...) makeAppointment(...) showHealthTimeTable(...) showTestTimeTable(...) showFacilityTimeTable(...) processTestResult()	healthAppointmentScreen() testAppointmentScreen() facilityAppointmentScreen() checkUpForm(...) testForm(...) makeAppointment(...) showHealthTimeTable(...) showTestTimeTable(...) showFacilityTimeTable(...) setHealthCenterWorkingDays(...) setFacilityWorkingDays(...) setHealthCenterTimeSlotSize(...)
StatsManager	showHeatMap() showStatistics(...) showMyContacts() showMyAppointment() showMyStats()	showHeatMap() showStatistics(...) showMyContacts() showMyAppointment() showMyStats()	showHeatMap() showStatistics(...) showMyContacts() showMyAppointment() showMyStats()	showHeatMap() showStatistics(...) showMyContacts() showMyAppointment() showMyStats()
CourseManager	showMyCourses() showCourse(...)	showMyCourses() showCourse(...) getInfectedStudents(...)		
AuthManager	authenticate(...) setTwoFactor(...) setCurrentUser(...) signUpScreen() signInScreen() signUp(...) signIn(...)	authenticate(...) setTwoFactor(...) setCurrentUser(...) signUpScreen() signInScreen() signUp(...) signIn(...)	authenticate(...) setTwoFactor(...) setCurrentUser(...) signUpScreen() signInScreen() signUp(...) signIn(...)	authenticate(...) setTwoFactor(...) setCurrentUser(...) signUpScreen() signInScreen() signUp(...) signIn(...)
NotificationAndAnnouncementManager	showNotificationScreen(...) showAnnouncements	showNotificationScreen(...) showAnnouncements	showNotificationScreen(...) showAnnouncements	showNotificationScreen(...) showAnnouncements

	Screen() refreshNotifications(...) markAsRead(...)	Screen() refreshNotifications(...) markAsRead(...)	Screen() refreshNotifications(...) markAsRead(...)	Screen() refreshNotifications(...) markAsRead(...) sendAnnouncement(...)
	Student	**Instructor**	**Healthcare Personnel**	**Administration Personnel**
StatsManager	showHeatMap() showStatistics(...) showMyContacts() showMyAppointments() showMyStats()	showHeatMap() showStatistics(...) showMyContacts() showMyAppointments() showMyStats()	showHeatMap() showStatistics(...) showMyContacts() showMyAppointments() showMyStats()	showHeatMap() showStatistics(...) showMyContacts() showMyAppointments() showMyStats()
HESManager	getInfectionStatus(...) getVaccinationStatus(...) getStatePolicies()	getInfectionStatus(...) getVaccinationStatus(...) getStatePolicies()	getInfectionStatus(...) getVaccinationStatus(...) getStatePolicies()	getInfectionStatus(...) getVaccinationStatus(...) getStatePolicies()
GoogleMapsManager	showHeatMap(...)	showHeatMap(...)	showHeatMap(...)	showHeatMap(...)
SeatingPlanManager	markCloseSeats(...) showSeatingPlan(...)	markCloseSeats Instructor(...) showSeatingPlan(...) seeInfectedStudents Seats(...)		
PoliciesManager	viewPolicies()	viewPolicies()	viewPolicies()	updateStatePolicies() setUniversityPolicy(...) viewPolicies()
InfectionManager				
DatabaseManager				

Table 1: Access Control Matrix

2.7. Boundary Conditions

2.7.1. Initialization

The application can be started by running the server from Amazon RDS, which builds and deploys the entire application and hosts it to the desired domain.

There is no need for any sort of installation process to use the application since it runs on a remote server and can be accessed from any modern

browser, as long as there exists an internet connection. After access to the website is established, users can log in to access the features. If they do not have an account, they can also register from the same panel, without requiring another type of initialization process.

The dynamic data that is being shown on the pages will be initialized & fetched from the remote server.

2.7.2. Termination

Any sort of termination will result in a complete shut-down across the entire application. This can be triggered manually by an admin or might arise from an unexpected operation. Regardless of the cause of termination, both Spring Framework and Amazon AWS have built-in features to grant protection against possible data loss.

2.7.3. Failure

Failure of services that our program is connected to will be automatically handled by restarting the environment. If failure does not proceed with success, developers will be notified. Service layer has a response for unsuccessful operations caused by developers. A pop-up will be displayed for debugging purposes.

3. Low-Level Design

3.1. Object Design Trade-offs

- **Maintainability vs. Performance**

  Our pandemic manager has many subsystems and many classes that correspond to these subsystems. We designed our class diagram like that because we wanted to create an easily writable and maintainable website, even though this will harm the performance compared to a more compact design where fewer classes do more things due to the extra response time from interaction between classes.

- **Usability vs. Functionality**

  We decided to not include a lot of the features we thought we could add because these features would either be not very useful (e.g. getting vaccination appointment, but we do not have permission to do them so we would have to just redirect users into related government web pages) or not feasible (e.g. we thought of tracking users' ring bus usage for infection tracking, but many students may not be using them often and rings no longer require scanning IDs for entry). Hence, we decided to focus on fewer features that we thought would

- be necessary in a pandemic manager, which allows us to create a simpler and easier to use interface.

3.2. Final Object Design

In this diagram, in order to reduce the complexity and for the sake of the readability we have decided to not include get and set methods. However, we are going to implement these methods in our coding stage. Please consider them as included.

Figure 7: Object Design

(For the high resolution image: https://i.imgur.com/elbvjpl.jpg)

3.2.1. Strategy Pattern

We have used the Strategy design pattern for the appointment control classes. We will have one AppointmentManager per client, however we need to be able to switch between making different types of appointments at runtime. Strategy design pattern allows us to do just that: When the user chooses to check or make a different type of appointment, we can just swap the AppointmentBehavior attribute. When the user wants to display the screen for a type of appointments or make a new appointment, or when an administration personnel wants to change working time of a certain facility,

their button presses initially call AppointmentManager's methods, which then it delegates that operation to the AppointmentBehavior it contains which then shows the relevant list, form or changes the relevant working time.

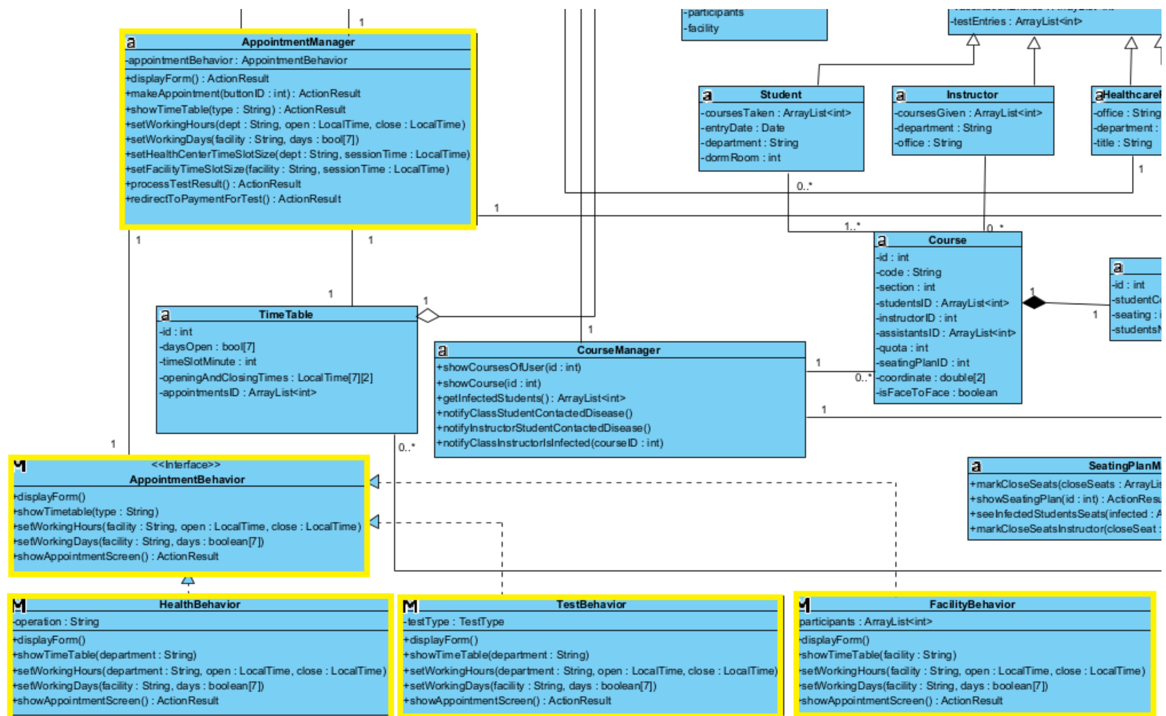

Figure 8: Implementation of Strategy Pattern
(For the high resolution image - at bottom left: https://i.imgur.com/3OrXuCT.jpg)

3.2.2. Singleton Pattern

Some control classes in our diagram, such as NotificationAndAnnouncementManager, AuthManager and DatabaseManager, will only have one instance for the whole website, because otherwise trying to do certain things simultaneously, such as trying to read an object from the database from two different places at the same time, might result in some unexpected behavior or errors. For this reason, we have decided to make these classes according to the Singleton design pattern. This means that each class has a static attribute that stores the only instance

(if it exists) of the class, which is returned by the static getInstance() method (if it does not exist, the instance is created here as well). Finally, the constructors of these classes are made private to ensure that another instance of them cannot be created anywhere else.

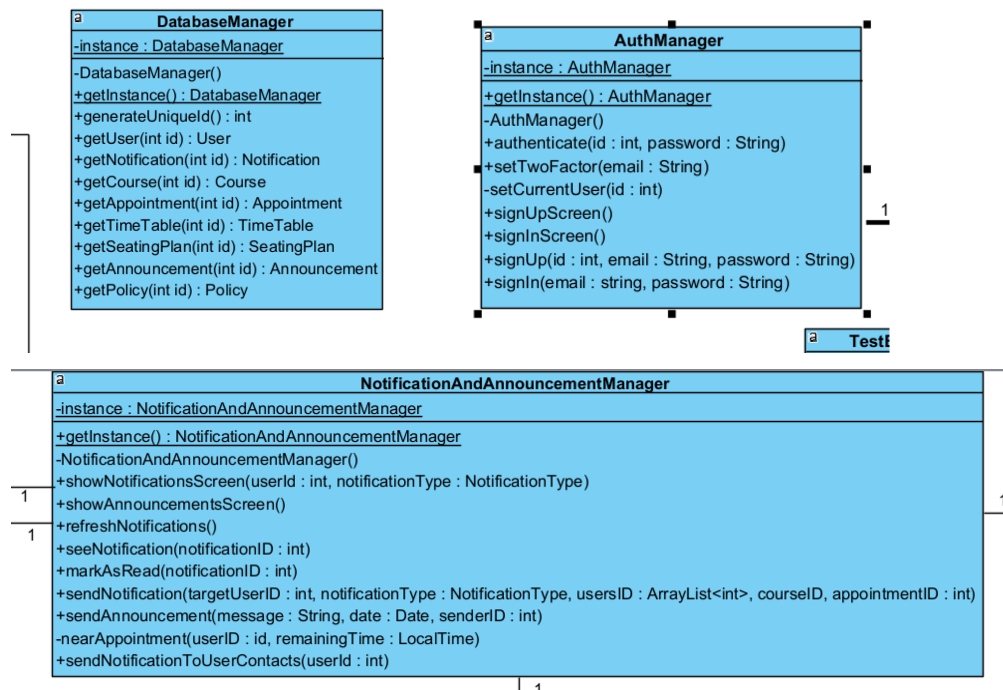

Figure 9: Implementation of Singleton Pattern

3.3. Packages

3.3.1. External Packages

Spring Boot: This package provides the framework that extends the use of Java to manage backend logic. We have decided to operate and develop the backend side of the system with Spring. This package interacts with both React and database [1].

React: This package is used to develop the frontend side of the application. This package interacts with the Spring Boot package [2].

Google Maps API: This package is used to create the heat map [5].

STARS API: This package is used to fetch the dorm mates, close contacts given before for users and the courses' information [6].

HES API: This package is used to obtain the current infection status of users, policies that have been published and other relevant information [7].

MySQL: This package is used to maintain the data relevant to the defined entities of the system. This package also communicates with the Spring Boot. Used as the preferred database choice [3].

Highcharts: This package is used to create charts, plots and other data relevant properties of pandemic with respect to the user's choice of time interval [8].

AWS: This package is used to create the attachment properties between users and remote servers. This package communicates with Spring Boot as well [4].

3.3.2. Internal Packages

User Screens Package: This package is the superpackage of all boundary classes.

Healthcare Screens Package: This package involves all boundary classes of healthcare personnel users.

Students Screens Package: This package involves all boundary classes of student users.

Instructor Screens Package: This package involves all boundary classes of instructor users.

Admin Screens Package: This package involves all boundary classes of administration users.

Seating Plan Package: This package involves all boundary classes related to the seating plan.

SeatingPlan, SeatingPlanManager, Course, CourseManager

Notification Manager Package: This package involves all boundary classes related to the notifications.
NotificationAndAnnouncementManager, Notification, Announcement, NotificationType

Authentication Manager Package: This package involves all boundary classes related to the user authentication.
AuthenticationManager, UserManager

Infection Tracking Package: This package involves all boundary classes to track the infection status.
InfectionManager, UserManager

HES Package: This package involves all boundary classes related to the processes of HES.
HESManager, PoliciesManager

STARS Package: This package involves all boundary classes related to the processes of STARS API.

Google Maps Package: This package involves all boundary classes related to the processes for Google Maps.
GoogleMapsManager

Statistics Manager Package: This package involves all boundary classes related to the statistics.
Heatmap, Statistics Manager

Appointment Manager Package: This package involves all boundary classes related to the appointments.
AppointmentManager, Appointment, HealthAppointment, FacilityAppointment, TestAppointment, Test, TestEntry, VaccinationEntry, TestType, VaccineType, TimeTable

Database Manager Package: This package involves the DatabaseManager class to manage the data transaction.

Database Package: This package involves the internal database for the

data storage.

3.4. Class Interfaces

3.4.1. Entity Classes

User

An abstract entity class that contains attributes common to all types of users.
Attributes:

- private int id: User’s Bilkent ID
- private String userName: User’s (real) name
- private String password: Hash of user’s password
- private String email: User’s email, used in authentication
- private String hesCode: User’s HES code for infection tracking
- private InfectionStatus infectionStatus: User’s infection status (healthy, infected, closely contacted)
- private bool fullyVaccinated: Stores whether user is fully vaccinated right now
- private ArrayList<int> closeContactsID: Stores ID’s of User’s (permanent) close contact Users
- private ArrayList<int> appointmentsID: Stores ID’s of User’s upcoming and past appointments
- private ArrayList<int> notificationsID: Stores ID’s of User’s notifications (read or unread)
- private ArrayList<int> temporaryCloseContactsID: Stores ID’s of User’s temporary close contacts (other Users this User has made a facility appointment together in the past two weeks)
- private ArrayList<VaccinationEntry> vaccinationEntries: Stores previous vaccination information of this User
- private ArrayList<TestEntry> testEntries: Stores previous test results of this User

Student

This entity class represents a student user (undergraduates as well as Ms. and PhD. students) and contains needed attributes of a student.

- Inherits User

Attributes:

- private ArrayList<int> coursesTaken: Stores the ID’s of the Courses this Student is taking
- private Date entryDate: Stores university registration date of the Student
- private String department: Stores the department of this Student

- private String dormRoom: Stores the dormitory room code of the user for infection tracking purposes (empty string if Student is not staying in dorms)

Instructor

This entity class represents an instructor user and contains needed attributes of an instructor.

- Inherits User

Attributes:

- private ArrayList<int> coursesGiven: Stores the ID's of the Courses this Instructor is giving
- private String department: Stores the department of this Instructor
- private String office: Stores the office code of this Instructor ("none" if the Instructor does not have an office)

HealthcarePersonnel

This entity class represents a healthcare personnel user (doctors, nurses etc.) and contains needed attributes of such personnel.

- Inherits User

Attributes:

- private String office: Stores the office code of this Instructor ("none" if the HealthcarePersonnel does not have an office)
- private String department: Stores the department of this HealthcarePersonnel (Dermatology, dentistry etc.)
- private String title: Stores the title of this HealthcarePersonnel (nurse, specialist etc.)

AdministrationPersonnel

This entity class represents various Bilkent administration personnel (registrar's office etc.) and contains needed attributes of such personnel. Not to be confused with website admin (we do not plan to administer the website directly with an admin user type, we will have our admin controls in our hosting service etc. instead).

- Inherits User

Attributes:

- private String office: Stores the office code of this AdministrationPersonnel
- private String title: Stores the title of this AdministrationPersonnel
- private String department: Stores the department of this AdministrationPersonnel (also affects some functionality permissions)

Course:

This entity class represents a course section and contains attributes needed to model it, including the people involved with it and their seating plans as well as the physical location of its classroom (if any).

Attributes:

- private int id: Stores the unique ID of the course to help operations in the database.
- private String code: Stores the code of the course.
- private int section: Stores the section of the course.
- private ArrayList<int> studentsId: Stores the students' IDs of this course.
- private int instructorID: Stores the instructor of the course/section.
- private ArrayList<int> assistantsId: Stores the assistants' IDs of this course.
- private int quota: Stores the quota of the course/section.
- private int seatingPlan: Stores the correspondent seating plan of the course.
- private bool isFaceToFace: Stores the condition of the class being face to face or not.
- private double[2] coordinates: Stores the coordinates of the classroom for the heat map.

SeatingPlan:

This entity class represents the seating plan in a course section and contains attributes required for that. The seating plan is modelled as an adjacency matrix.
Attributes:

- private int id: Stores the unique ID of the seating plan to help operations in the database.
- private int studentCount: Stores the number of students in the plan.
- private ArrayList<int> studentsNearInstructor: Stores the students proximate to the instructor.
- private int[studentCount][studentCount] seating: Stores the matrix of seatings.

Appointment:

This abstract entity class contains attributes required for all types of appointments (that we support) that are done in the university.
Attributes:

- private int id: Stores the unique ID of the appointment to help operations in the database
- private Date date: Stores the appointment's date and time.
- private int hostUserID: Stores the ID of the user that makes the appointment. Provides the binding of appointment and user.
- private String message: Stores the information that may be needed for the appointment.

HealthAppointment:

This entity class represents the appointments of Bilkent Health Center except the test appointment (checkups etc.)

- Inherits Appointment

Attributes:

- private String Department: Stores the appointment’s healthcare department

TestAppointment:

This entity class represents pandemic test appointments of Bilkent Health Center. This class does not extend HealthAppointment for mainly convenience reasons (because test appointments are a main feature on their own and are shown in a different screen than other health appointments).

- Inherits Appointment

Attributes:

- private TestType testType: Stores the preferred test type

FacilityAppointment:

This entity class represents the appointments of facilities except the Bilkent Health Center (private rooms in the library, gym, swimming pool etc.)

- Inherits Appointment

Attributes:

- private ArrayList<int> participants: Stores the participants’ IDs of the appointments that contain group activities and close contact.

TimeTable:

This entity class represents various time tables which are used in scheduling various types of appointments.

Attributes:

- private int id: Stores the unique ID of this TimeTable
- private bool[7] days open: Stores the days that are open to use of the facility. Indexes start from monday, end at sunday. True if open, false otherwise.
- private int timeSlotMinute: Stores the time interval length of a slot of this instance.
- private LocalTime[7][2] openingAndClosingTimes: Stores opening and closing times of facilities daywise. First dimension is for days and the second dimension is for the opening and closing times. First dimension goes Monday to Sunday and the second dimension goes as opening and closing.
- private ArrayList<ArrayList<int>> appointmentsID: Stores the appointment IDs. First dimension is days and the second dimension is time intervals.

Notification:

This entity class represents notifications users get for reasons such as having a close contact infected, upcoming appointment reminders etc.

Attributes:

- private int id: Stores the unique ID of this Notification. Helps with database manipulation.
- private String message: Stores the notification text.

- private NotificationType notificationType: Stores the type of the notification (contact infection alert, appointment reminder, infection alert for instructors, other). Helps with notification filtering.
- private int receiverID: Stores the receiver’s ID of this notification. Helps with binding of users to the notifications.
- private Date date: Stores the sent date and time of the notification.
- private bool isRead: Stores the state of the notification, if it is read or not.

Announcement:

This entity class represents an announcement, which are similar to notifications except that they are sitewide rather than per-user. They are created by the system to announce weekly university-wide infection status reports, state/university policy changes, site updates etc.

- Announcements can be thought of as global (sitewide) notifications

Attributes:

- private int id: Stores the unique ID of this announcement. Helps with database manipulation.
- private String message: Stores the announcement text.
- private int senderID: Stores the sender’s ID of this announcement. Helps with binding of the sender users to the announcements (Announcements are sent to every user).
- private Date date: Stores the sent date and time of the announcement.

VaccinationEntry:

This entity class represents a single dose of vaccine taken by an user.

Attributes:

- private int id: Stores the unique ID of this TestEntry. Helps with database manipulation.
- private VaccineType vaccineType: Stores the type of this vaccination (BioNTech etc.).
- private Date date: Stores the date and time of this test.
- private String location: Stores the location where the vaccine is done.

TestEntry:

This entity class represents a single test result of an user.

Attributes:

- private int id: Stores the unique ID of this TestEntry. Helps with database manipulation.
- private int testedId: Stores the id of the user that this test has been executed on. Helps with the binding of users to the tests.
- private TestType testType: Stores the type of this test (Diagnovir, PCR etc.).
- private bool isPositive: Stores the result of this test.
- private Date date: Stores the date and time of this test.

Policy:

This entity class represents a policy, a set of pandemic related “rules.” Policies may be statewide or universitywide, or they can be only for specific facilities (gym etc.).

Attributes:

- private int id: Stores the unique ID of this Policy. Helps with database manipulation.
- private String description: Stores the change text of this policy
- private ArrayList<VaccineType> acceptedVaccines: Stores the accepted vaccines by the university.
- private String vaccinationInfo: Stores the standards of the university about vaccination (minimum 2 vaccines of BioNTech etc.).
- private ArrayList<TestType> acceptedTest: Stores the accepted tests by the university.
- private String quarantineInfo: Stores the standards of university about the quarantine of the users that are infected or closely contacted.
- private LocalTime[2] workingHours: Stores the working hours of the personnel.
- private String otherInfo: Stores the information about other factors, maybe factors that cannot be anticipated.

HeatMap:

This entity class represents a heatmap, which is used in the statistics page to show the distribution of current cases at the university.

Attributes:

- private GoogleMap map: Stores the Google Map instance which the heat map is built on.
- private ArrayList<double[2]> caseCoordinates: Stores the all coordinates of where the infected people have been at.

Control Classes

UserManager

This class manages the functionality about the User whose ID is bound to it (usually the logged in user).

Attributes:

- private int currentUser: Stores the ID of the User that is currently bound.

Methods:

- public ActionResult userSettingsScreen(): Opens the User Settings panel
- public ActionResult openProfile(): Opens the User Profile panel
- public ActionResult changePassword(String passwordOld, String passwordNew, String passwordNew2): Opens the change password panel
- public ActionResult changeReminderTime(int reminderHours): Opens the Change Notification Reminder Time panel
- public ActionResult changeEmail(String mailNew): Opens the Change Email panel

- public bool isFullyVaccinated(): Calculates whether the User is fully vaccinated at the moment or not

CourseManager

This class manages the functionality related to Courses.
Methods:

- public ActionResult showMyCourses(): Opens the Show All Courses panel
- public ActionResult showCourse(int id): Opens the Show Course panel with given course ID
- public ArrayList<int> getInfectedStudents(int courseID): Finds and returns the IDs of infected students in given course
- public void notifyClassStudentContactedDisease(int courseID, int studentID, InfectionStatus type): Uses the NotificationAndAnnouncementManager to send a Notification to other Students that are enrolled in class, teaching assistant Students and the Instructor that a Student is infected or closely contacted (infected/closely contacted Student's information is anonymous in Notifications sent to enrolled Students but not in the Notification sent to the Instructor and the teaching assistant Students)
- public void notifyClassInstructorContactedDisease(int courseID): Uses the NotificationAndAnnouncementManager to send a Notification to other people in the class that the instructor is infected or closely contacted

SeatingPlanManager

This class manages the functionality related to SeatingPlans.
Methods:

- public ActionResult markCloseSeats(ArrayList<int> closedSeats): Binds the current user with the closed seaters of him/her. Then, redirects the action to the showSeatingPlan method.
- public ActionResult showSeatingPlan(int id): Shows seating plan to user to make user to be able to mark the closed seatings.
- public ActionResult seeInfectedStudentsSeats(ArrayList<int> infected): Shows the infected students' seats to the instructors.
- public markClosedSeatsInstructor(ArrayList<int> closedSeats): Binds the current instructor with the closed seaters of him/her. Then, redirects the action to the showSeatingPlan method.

AppointmentManager

This class manages the functionality related to appointments, including parts of the creation process as well as setting some global parameters.
Methods:

- public ActionResult healthAppointmentScreen(): Opens the Health Appointments panel
- public ActionResult testAppointmentScreen(): Opens the Test Appointments panel

- public ActionResult facilityAppointmentsScreen(): Opens the Facility Appointments panel
- public ActionResult checkUpForm(String department, String operation): Opens the Check Up Form panel for given health center department and operation
- public ActionResult testForm(TestType testType): Opens the Test Form panel for given test type
- public ActionResult facilityForm(String facilityType, String operation): Opens the Check Up Form panel for given health center department and operation
- public ActionResult makeAppointment(int buttonID): Processes necessary operations after the User clicks the “complete appointment form” button
- public ActionResult showHealthTimeTable(String department): Displays the time table for given health center department
- public ActionResult showTestTimeTable(TestType testType): Displays the time table for given test type
- public ActionResult showFacilityTimeTable(String facility): Displays the time table for given facility
- public void setHealthCenterWorkingHours(LocalTime opening, LocalTime closing): Sets health center working hours
- public void setFacilityWorkingHours(String facility, LocalTime opening, LocalTime closing): Sets working hours of given facility
- public void setHealthCenterWorkingDays(bool[7] days): Sets working days of health center
- public void setFacilityWorkingDays(String facility, bool[7] days): Sets working days of given facility
- public void setHealthCenterTimeSlotSize(String dept, LocalTime time): Sets the time allocated per time slot (in the time table) for given department of the health center
- public void setFacilityTimeSlotSize(String facility, LocalTime time): Sets the time allocated per time slot (in the time table) for given facility
- public ActionResult processTestResult(): Opens the Process Test Result panel where a HealthcarePersonnel can process a test result
- public ActionResult redirectToPaymentForTest(): If the User needs to pay for the test (e.g. they are willingly not vaccinated), redirect them into relevant Bilkent page with instructions

AuthManager

This class manages authentication related functionality, such as signing up and signing in.

Methods:

- public bool authenticate(int id, String password): Fetches the user input into database to complete user authentication.
- public void setTwoFactor(String email): Sets two factor authentication according to given email.

- public void setCurrentUser(int id): Sets current user in current session of web page according to student id.
- public ActionResult signUpScreen(): Displays sign up screen for user.
- public ActionResult signInScreen(): Displays sign in screen for user.
- public void signUp(int id, String email, String password): Processes the sign up process with given inputs id, email and password.
- public void signIn(String email, String password): Processes the sign in process with given inputs email and password.

StatsManager

This class manages functionality related to the statistics page, including showing graphs and heatmaps.

Methods:

- public void createGraph(): Creates a graph of contacted people.
- public ActionResult showHeatMap(): Displays heat map.
- public ActionResult showStatistics(interval:intervalEnum): Shows detailed statistics of the current status in a given interval (daily, weekly, monthly).
- public ActionResult showMyContacts(): Displays user's close contacts.
- public ActionResult showMyAppoitments(): Displays user's different appointments in the health center.
- public ActionResult showMyStats (): Displays user related statistics.

DatabaseManager

This class provides an easy interface, or a facade, for the database operations it manages.

Methods:

- public User getUser(int id): Fetches the user according to its ID.
- public Notification getNotification(int id): Fetches the notification according to the given ID.
- public Course getCourse(int id): Fetches the course according to the given ID.
- public Appointment getAppointment(int id): Fetches the appointment according to the given ID.
- public TimeTable getTimeTable(int id): Fetches the time table according to the given ID.
- public SeatingPlan getSeatingPlan(int id): Fetches the seating plan according to the given ID.
- public Announcement getAnnouncement(int id): Fetches the announcement according to the given ID.
- public Policy getPolicy(int id): Fetches the policy according to the given ID.

NotificationAndAnnouncementManager

This class manages functionality regarding notifications and announcements such as sending, showing etc.
Methods:

- public ActionResult showNotificationsScreen(NotificationType notificationType): Opens the Notifications panel
- public ActionResult showAnnouncementsScreen(): Opens the Announcements panel
- public ActionResult refreshNotifications(): Processes what happens when the “refresh notifications” button is clicked
- public ActionResult seeNotification(int id): Expands the panel of given notification (with more detailed information)
- public void markAsRead(int id): Marks given notification as read
- public void sendNotification(int targetUserID, NotificationType type, ArrayList<int> usersID, int courseID, int appointmentID): Sends notification of given type with given parameters (not all of these parameters are used in all notification types, when this method is called unused parameters should be set to null, 0 or empty string depending on their type)
- public void sendAnnouncement(int id, String message, Date date, int senderID): Creates an Announcement with given parameters and sends it
- private ArrayList<int> nearAppointment(int userID, LocalTime remainingTime): Returns IDs of Appointments which will happen within given remaining time.

PoliciesManager:

This class manages functionality about policies.
Methods:

- public void updateStatePolicies(): Updates the policies made by the State.
- public void setUniversityPolicy(): Changes the policies made by the University.
- public ActionResult viewPolicies(): Shows the policies screen to the user.

HESManager:

This class provides a simpler interface for the necessary HES API functions.
Methods:

- public InfectionStatus getInfectionStatus(String hesCode): Gets the infection status of the user with HES code hesCode.
- public ArrayList<VaccinationEntry> getVaccinationStatus(String hesCode): Gets the vaccinations that the user with HES code hesCode has.
- public String getStatePolicies(): Gets the state policies that are defined in the HES API.

GoogleMapsManager:

This class provides a simpler interface for the necessary Google Maps API functions.
Methods:

- public HeatMap generateHeatMap(): Generates the heat map of the campus.

- public ActionResult showHeatMap(HeatMap map): Shows the generated heat map of the campus to the user.

STARSManager:

This class provides a simpler interface for the necessary STARS API functions.
Methods:

- public ArrayList<int> getDormmates(int id): Gets the dorm-mates of the user with ID = id.
- public ArrayList<JSON>fetchCourseOfferings(int id): Fetches the offerings of the course with ID = id as list of JSONs.

InfectionManager:

This class manages infection related functionality, including refreshing infection statuses of all users every 10 minutes.
Methods:

- public InfectionStatus getInfectedStatus(int id): Gets the infection status of the user with ID = id.
- public ArrayList<int> getInfectedStudents(int courseId): Gets the infected students of the course with id = courseId.
- public void notifyCloseContactsUserIsInfected(int userId): Fires notification manager prompt to notify the close contacts of the user with ID = userId, he/she is infected.
- public ArrayList<int> getInfectedContactsSinceDays(int days): Gets the infected contacts within the given number of days.

4. Improvement Summary

In the Design Goals part, we improved our explanations for our requirements.

We have added a deployment diagram which was missing in the first iteration.

Under the Subsystem Decomposition header, we separated the layers a bit so that the diagram can be read more easily. We also added descriptions for each subsystem in order to explain unambiguously what each subsystem is responsible for.

In Hardware/Software Mapping, we added some more concrete system requirements (minimum RAM, CPU, disk space etc.) of our hosting service.

We updated our access matrix so that instead of a more abstract description of access statuses, it now provides information about the actors' access status of actual methods in our classes.

Under the Boundary Conditions header, we have updated the initialization procedure so that it now represents the admin’s point of view while they are booting up the server in production, rather than the user’s point of view when they are connecting to the website.

In our Class Diagram, we now indicate what the associations mean. We have also explicitly stated the places where the multiplicity is 1 (previously, no indication implied that). Finally, we added two design patterns, Singleton and Strategy, to our diagram, and explained why we did so. We furthermore added new managers to extend the modularity and functionality. We have also added a database manager to interact with our database.

In the Class Interfaces part, we added brief descriptions for each class in order to prevent any ambiguities.

5. Glossary

AP : Administration Personnel

API : Application Programming Interface

AWS : Amazon Web Services

HES : Hayat Eve Sığar

HP: Healthcare Personnel

JS : JavaScript

MVC : Model-View-Controller

OOP : Object-Oriented Programming

SSL : Secure Socket Layer

STARS : Student Academic Information Registration System

SQL : Structured Query Language

TA : Teaching Assistant

TLS : Transport Layer Security

6. References

[1] "Spring Framework". https://spring.io/projects/spring-framework. (accessed Nov. 22, 2021).

[2] "Getting Started - React".https://en.reactjs.org/docs/getting-started.html (accessed Nov, 22, 2021).

[3] "MySQL: Documentation". https://dev.mysql.com/doc/ (accessed Nov. 24, 2021)

[4] "AWS Documentation". https://docs.aws.amazon.com/ (accessed Nov. 27, 2021)

[5] "Google Maps Platform | Google Developers." www.developers.google.com https://www.developers.google.com/maps. (accessed Nov. 24, 2021)

[6] "Bilkent University - STARS" www.stars.bilkent.edu.tr. (accessed Nov. 24, 2021)

[7] "hayatevesığar". https://www.hayatevesigar.saglik.gov.tr. (accessed Nov. 24, 2021)

[8] "Highcharts Documentation - Highcharts". https://www.highcharts.com/docs/index. (accessed Nov. 28, 2021)

[9] Bruegge, B., & Dutoit, A. H. (2003). *Object-oriented software engineering: Using UML, patterns and Java*. Upper Saddle River, NJ: Prentice Hall 47

[10] "Security in Amazon RDS - Amazon Relational Database Service". https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/UsingWithRDS.html (accessed 15.12.2021)