**Technical Manual for Advisement System**

By Angela John, Jaden Severance, Samantha Duffy

1. **Overview**

This document describes in detail, how a system must be set up to be able to successfully run the Schedule Advisement System.

1. **System Requirements**

* Java Development Kit (JDK 8 or newer)
* Spring Framework
* A SQL Server: MySQL, PostgreSQL, or Microsoft SQL server
* Microsoft® Excel® 16.0

1. **Assumptions**

* It is assumed that all system requirements are met except the installation of the Java Development Kit and Spring Framework.
* It is also assumed that the Administrator and Faculty have existing Excel files with the correct names.
  + For the program to each its full potential, the Excel file must contain the first name, last name, email, phone number, and password of the corresponding advisors and advisees. For example:

Administrator

**A close-up of a web page

Description automatically generated**

Faculty

A close-up of a web page

Description automatically generated

* When faculty uploads their schedule, the Excel file must contain the desired date, start time, end time, and length of their available

scheduling appointments. For example:

A close up of a clock

Description automatically generated

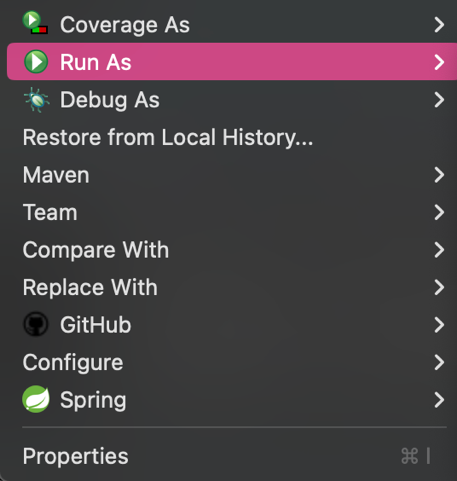
1. **Installing**
   1. **Installing JDK**

This provides the ability to run the Schedule Advisement System from the command line.

* 1. **Installing Spring Framework**

Since the program is developed with the Spring Framework, it is crucial to have the Spring Framework set up in your development environment. Please follow these steps to ensure this:

1. Select the option with your corresponding system and download Spring Tools from <https://spring.io/tools>
   1. If using Eclipse, this will then automatically add the plugin to your system.
2. **Running program**
   1. **Using Eclipse**
3. Open the Schedule Advisement project file in Eclipse.
4. Right-click on the project file and select “Run As”.



1. Select “Spring Boot App”

A screenshot of a computer

Description automatically generated

1. This will then execute the program and indicate what port the program is using.



1. Since the port being used is 8080, we will type “ localhost:8080 “ into a web browser.
2. **Classes**

**AsyncConfiguration.java** – initializes default roles and users into the UserRoleRepository

**MailConfiguration.java** – this class is responsible for email settings that are required for sending emails.

**AdminController.java** – handles administrative tasks and manages user data; uploads, edits, and deletes advisor data, manages system settings, and handles and processes Excel files. This class has access to the UserRepository, AdvisorRepository, UserRoleRepository, and PasswordEncoder.

**AdminSystemSettings.java** – this class displays the current settings and is able to update the settings.

**EmailController.java** – handles email-related operations.

**FacultyController.java** – responsible for rendering the faculty dashboard and sending emails to advisees associated with a faculty member. This class may have access to the EmailController and the AdviseeRepository.

**IndexController**.**java** – this class renders and displays multiple URLs: index, login, missionStatement, FAQ, verify2FAPage.

**LandingPageController.java** – controller that displays the home page, landing pages, as well as user settings page.

**StudentController.java** – renders the advisee dashboard and displays their upcoming meetings.

**UserController**.**java** – has access to the UserRepository and UtilityController. Retrieves, adds, and deletes users to the UserRepository, also generates and manages CAPTCHA for user verification.

**CustomErrorController**.**java** – controller for managing the errorPage. This presents the user with a readable error.

**AdminSettings.java** – allows the administrator to set the academic start and end dates as well as other semester information.

**AdviseeData.java, AdvisorData.java, ScheduleData.java**– classes that use getters and setters to manage advisee, advisor, and schedule data throughout the system.

**Timeslot.java** – entity class that represents the available time slots for scheduling.

**User.java**- represents a user within the system by encapsulating its attributes related to user accounts. Interacts with the UserRepository, UserController and UserRole.

**UserRole.java** – entity for what website features a user has access to. Distinguishes from a regular user to an administrator.

**AdviseeUpload.java** – manages, uploads, creates, updates, and deletes advisee data in the system. This class has access to the UserRepository, AdviseeRepository, and the UserRoleRepository.

**Schedule.java** – controller class that manages the uploaded schedule. Allows the user to also select, cancel, and delete timeslots. Has dependencies TimeService.java, ScheduleService.java, and ScheduleRepository.java.

**ScheduleService.java** – manages the schedule data by interacting with the ScheduleRepository.java.

**TimeService.java** – manages the time slot data and provides methods for retrieving, adding, and deleting time slots.

**TimeslotUpload.java** – interacts with TimeService to upload, edit, and display the schedule.

**UserService.java** – manages user-related operations within the system. Allows correct users to update, retrieve, and delete information.

**Role.java** – defines available roles within the system.

1. **Interactions between Schedule Advisement Classes**

**Admin to System Interactions**

* The administrator logs into the system and sets the semester start date and end date as well as the registration start and end date, this is done with the AdminSettingsController class.
* The administrator uploads excel file named “advisor\_logins” to store the login data within the database.
* Administrator has option to either add, delete, or edit faculty. These methods are all listed in the AdminController class.
* The administrator also has list of all users in system including each name, email, phone number, as well as their assigned role.

**Faculty to System Interactions**

* The faculty must upload their list of advisees, via ‘upload advisees’. This saves the advisee data into the database. Each advisee uploaded will assigned to the faculty member that uploaded their data. This data is saved and assigned using the AdviseeData.java class.
* Faculty also has the option to add, delete, or edit their advisees.
* Faculty must also upload their desired schedule, via ‘upload schedule’. This schedule then gets saved into the database for both the faculty and advisees to see.

**Faculty to Advisee Interactions**

* Faculty are able to send mass email to all advisees. This is done with the *sendEmailToList()* method.

**Advisee to System Interactions**

* The advisee is able to select one meeting with their advisor, this is then saved into the database.
* The advisee is also able to view and manage their already selected meetings via ‘Upcoming Meetings’.

1. **UML Diagrams**

The following UML diagrams will help better explain the functionalities of the Schedule Advisement System as well as the interactions between the various agents. For class diagrams, see PlantUML in documents folder.

**Use Case Conditions**

**Admin Agent**

Condition: The administrator logs into the system and gains access to administrative functions.

Activities: The administrator logs in and out of the system as well as uploads existing faculty data. The administrator

Post Conditions: Update the system with faculty information, modify existing faculty records, or remove faculty data

**Faculty Agent:**

Condition: The faculty interacts with the system to manage student-related and scheduling tasks.

Activities: Faculty logs in and out of the system. The faculty also uploads new student data as well as schedule information. Faculty also edits, edits, or adds new students to the database.

Post Conditions: Updated student information and adjusted scheduling details.

**Student Agent:**

Condition: The student actor interacts with the system primarily for selecting a timeslot.

Activities: The student logs into the system and selects their desired timeslot. The student then logs out of the system.

Post Conditions: The student has scheduled their timeslot and made it unavailable to peers.

**Sequence Diagrams:**

1. **Email Sequence**

The following describes the Email Sequence Diagram as the actions are performed.

**1. User Interaction:**

* The user interacts with the web page through a web browser.

**2. Request for Email Functionalities:**

* User requested email functionalities through the browser. This causes the browser to send a request to the *EmailController* to handle the request

**3. Rendering Email Functionalities:**

* *EmailController* retrieves the necessary data and renders the email functionalities to the browser.

**4. Sending Email to List:**

* User requests to send an email to a list of recipients
* The browser sends a request to the *EmailController.*
* The *EmailController* interacts with the *javaMailSender()* method component to send email to recipients.
* *javaMailSender()* method creates a message using the SimpleMailMessage component and sends it to specified recipients.
* *EmailController* then renders a success message in the browser

**5. Sending Appointment Confirmation Email**

* The *EmailController* interacts with *javaMailSender()* method to send the appointment confirmation email
* The *javaMailSender()* method creates a message using the SimpleMailMessage component and sends email to specified email address.
* With a successful sending, the *EmailController* renders a success message in the browser.

**6. Sending Two-Factor Authentication Email**

* *EmailController* interacts with the *javaMailSender()* method to send the two-factor authentication email.
* The *javaMailSender()* method creates a message using the SimpleMailMessage component and sends to the specified user along with the authentication code.
* Upon a successful sending, the *EmailController* renders a success message in the browser.

**7. Request for Email Verification Page:**

* *EmailController* interacts with the ServletContext to retrieve the context path
* EmailController interacts with the UserEntity to set the email verification code to user
* The IPAddress component is then involved in setting the message text for email.
* *EmailController* interacts with the *javaMailSender()* method to send the welcome email and the verification code.
* *EmailController* then renders email verification page in the browser.

1. **Schedule Sequence**

The following describes the Scheduling Sequence Diagram as the actions are performed.

1. **User Requests Schedule (/schedule):**

* The user initiates a request to view their schedule.
* The request is sent to the ScheduleController.

**2. ScheduleController Handling Request**

* *ScheduleController* receives this request
* Interacts with UserService, TimeService, and ScheduleService as well as UserRepository and TimeslotRepository to fetch the necessary data.
* UserService is invoked to recieve user information based on the email provided.
* TimeService is called to fetch the list of available timeslots.
* ScheduleService is utilized to obtain the schedule list.
* UserRepository is queried to find the advisor’s information by their email.
* TimeslotRepository is accessed to retrieve time slots associated with the advisor.

**3. Data Preparation and Rendering**

* *ScheduleController* prepares the retrieved data for rendering purposes.
* The user’s roles, timeslots, and schedules are added as attributes to the model.
* The schedule view is rendered with provided data.

**4. User Views Schedule**

* The rendered schedule view is displayed to the user.

**5. User Selects Timeslot ( POST / select):**

* The user submits a request to select a timeslot.
* *ScheduleController* handles the request.
* UserService retrieves user information by email.
* UserService fetches a list of all users as well as the advisor information.
* *EmailController* is invoked to send an appointment confirmation email.

**Login Activity Diagram:**

This diagram depicts the steps involved in the login processes within our system. It provides a representation of how users authenticate themselves and are directed based on their roles within the system.

A diagram of a user flow

Description automatically generated