# SOFTWARE PROJECT FINAL REPORT

Mohammed Arshad LNU, Warren Wu

Dec 5, 2024

**Table Of Contents**

| | | |
|---|---|---|
| 6 | Test Management | 6.1. A complete list of system test cases<br>6.2. Traceability of test cases to use cases<br>6.3. Techniques used for test case generation<br>6.4. Test results and assessments<br>6.5. Defects reports |
| 7 | Conclusions | 7.1. Outcomes of the project<br>7.2. Lessons learned<br>7.3. Future development |
| 8 | References | References |

## List of Figures

## List of Tables

## 1. Introduction

---

### 1.1. Purpose and Scope

The schedule planner project aims to create a web-based tool that allows students to design and visualize weekly schedules based on their class timings and personal activities. The tool offers an intuitive interface where users can add, modify, and delete schedule entries. It is intended for college students who want an organized way to manage their time efficiently.

---

### 1.2. Product Overview (including capabilities, scenarios for using the product, etc.)

The Schedule Planner simplifies the process of scheduling by providing an interactive grid layout where users can enter class names, times, and other activities. The system ensures accessibility by allowing cross-platform usage on any browser. Users can:

- Add, edit, or remove schedules in real-time.
- View schedules in a user-friendly grid format.
- Save schedules locally as a file or screenshot for offline use.

**Scenarios for Using the Product**

- **Student Scenario**: A student inputs their class schedule to visualize gaps and overlaps.
- **Personal Time Management**: A user plans study, exercise, or social activities around classes.

---

### 1.3. Structure of the Document

This document is organized into sections covering project management, requirements, architecture, design, testing, and conclusions, each providing in-depth details about the project lifecycle.

---

### 1.4. Terms, Acronyms, and Abbreviations

- UI: User Interface
- API: Application Programming Interface
- CRUD: Create, Read, Update, Delete
- IDE: Integrated Development Kit

## 2. Project Management Plan

---

### 2.1. Project Organization

The project team is structured as follows:

- **Mohammed Arshad**: Responsible for backend development, including schedule generation and algorithm implementation.
- **Warren Wu**: Focused on frontend development, creating the user interface using the Vaadin framework.

Each team member contributed equally, ensuring the project met its deadlines while maintaining quality.

---

### 2.2. Life Cycle Model Used

The project followed the Incremental Development Model:

1. Initial Phase: Development of core schedule generation functionality.
2. Second Phase: Addition of user interface components.
3. Final Phase: Integration of all components, testing, and bug fixes.

This approach allowed for iterative improvements based on feedback from testing.

---

### 2.3. Risk Analysis

| Risk | Likelihood | Impact | Mitigation Strategy |
|---|---|---|---|
| Delays in Integration | Medium | High | Conduct weekly-team meetings |
| Technology compatibility | Low | Medium | Test cross-platform usability frequently |
| Loss of Project files | Medium | High | Use GitHub for version control |

## 2.4. Hardware and Software Resource Requirements

- **Hardware**
  - macOS and Windows Laptops.
- **Software**
  - IntelliJ IDEA for Development.
  - Vaadin framework for UI.
  - Java 17 for backend.

---

## 2.5. Deliverables and schedule

**Project Plan:** 3rd October 2024

**Software requirement specifications:** 10th October 2024

**Software design specification:** 24th October 2024

**Initial version (prototype):** 14th November 2025

**Test plan:** 21st November, 2024

**Project Deliverables (project report, user manual, developer's guide, codes and data), project demonstration:** 5th December 2024

## 3. Requirement Specifications

3.1. Stakeholders for the system

| **Stakeholder** | **Role** |
|---|---|
| Students | Use the system to plan and organize academic schedules. |
| Faculty | Secondary users who may assist students in using the tool. |

---

3.2. Use cases

### Create Schedule

| Attribute | Details |
|---|---|
| Actor(s): | Students, Education Faculty |
| Description: | A user can create a new schedule by entering class details. |
| Data Required: | Class Name, Class Code, Start Time, End Time, Days, Instructor |
| Stimulus: | The user issues a command (e.g., clicking "Create Schedule"). |
| Response: | A blank, editable schedule opens for the user to input their details. |
| Comments: | The system redirects the user to an editable document after clicking the "Create Schedule" button. |

## View Schedule

| Attribute | Details |
|---|---|
| Actor(s): | Students, Education Faculty |
| Description: | The user can view their weekly schedule with each class displayed in its respective time block. |
| Data Required: | Created Schedule |
| Stimulus: | The user clicks the "View" button. |
| Response: | Displays the user’s schedule in a grid format. |
| Comments: | The schedule is displayed as a weekly grid, offering a clear visualization of classes. |

## Modify Schedule

| Attribute | Details |
|---|---|
| Actor(s): | Students, Education Faculty |
| Description: | The user can edit or update an existing schedule to fix mistakes or reflect changes. |
| Data Required: | New Class Name, Class Code, Start Time, End Time, Days, Instructor |
| Stimulus: | The user clicks the "Edit" button to modify the schedule. |
| Response: | The system opens the schedule in edit mode, allowing changes. |
| Comments: | Changes overwrite the existing schedule after being saved. |

## Delete Class in Schedule

| Attribute | Details |
|---|---|
| Actor(s): | Students, Education Faculty |
| Description: | A user can delete a specific class entry from the schedule. |
| Data Required: | None |

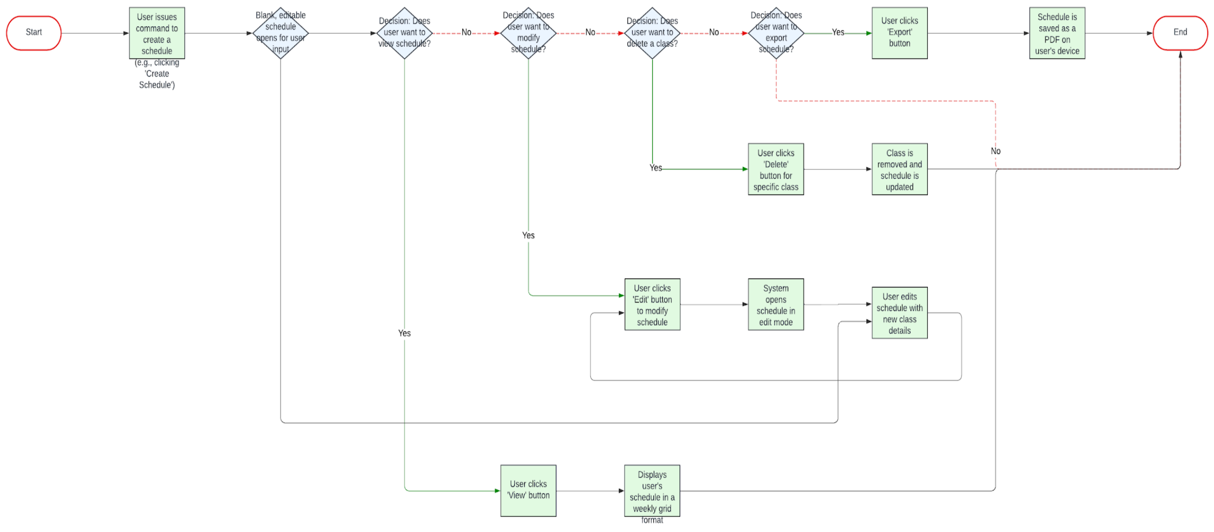
|  |  |
|---|---|
| Stimulus: | The user clicks the "Delete" button for the corresponding class. |
| Response: | Removes the selected class from the schedule. |
| Comments: | The updated schedule saves automatically after deletion. |

## Export Schedule

| Attribute | Details |
|---|---|
| Actor(s): | Students, Education Faculty |
| Description: | Users can export and download their finalized schedule for offline use. |
| Data Required: | PDF file format |
| Stimulus: | The user clicks the "Export" button. |
| Response: | The schedule is saved as a PDF file on the user’s device. |
| Comments: | The exported file can be shared or printed as needed. |

---

### 3.2.1. Graphic use case model


Start
User issues command to create a schedule (e.g., clicking 'Create Schedule')
Blank, editable schedule opens for user input
Decision: Does user want to view schedule?
No
Decision: Does user want to modify schedule?
No
Decision: Does user want to delete a class?
No
Decision: Does user want to export schedule?
Yes
User clicks 'Export' button
Schedule is saved as a PDF on user's device
End
Yes
User clicks 'Delete' button for specific class
Class is removed and schedule is updated
No
Yes
User clicks 'Edit' button to modify schedule
System opens schedule in edit mode
User edits schedule with new class details
Yes
User clicks 'View' button
Displays user's schedule in a weekly grid format

---

### 3.2.2. Textual Description for each use case

1. Create Schedule

- Description: The primary use case where a user (student or faculty) creates a new schedule. This involves entering specific class details, including the class name, code, start and end times, the days the class occurs, and the instructor's name.
- Preconditions:
  - The user is logged into the system.
  - The user has access to a form for creating schedules.
- Postconditions:
  - A new, empty schedule is created, and the user is taken to an editable interface where they can input their schedule details.
- Main Flow:
  - The user clicks the "Create Schedule" button.
  - A new, blank schedule interface opens.
  - The user inputs all necessary details: class name, class code, start time, end time, days, and instructor.
  - The user clicks "Save" to confirm and store the schedule.
- Alternative Flows:
  - Error Handling: If any required fields are left blank or if data is invalid (e.g., end time before start time), the system shows a prompt to correct the entry before proceeding.
- Exceptions: If the system experiences an error while saving the schedule (e.g., due to connectivity issues), a message is displayed prompting the user to try again.

2. View Schedule

- Description: The user views their created schedule in a grid format, displaying classes within a weekly structure. This is a passive use case where the user only wants to view their schedule without making changes.
- Preconditions:
  - The user has previously created or updated a schedule.
  - The user is logged into the system.
- Postconditions:
  - The user sees their weekly schedule with clear visualization of class blocks.
- Main Flow:
  - The user clicks the "View Schedule" button.
  - The system loads the user's schedule.
  - The system displays the schedule in a weekly grid format with each class listed in its respective time slot.
- Alternative Flows:
  - Error Handling: If no schedule exists for the user, a message appears, asking them to create a schedule first.
- Exceptions: If there is a system issue retrieving the schedule (e.g., database issues), an error message is displayed to the user.

3. Modify Schedule

- Description: The user can edit an existing schedule to make changes, such as correcting errors or adjusting class times. This can be done by modifying class details like name, code, instructor, or time.
- Preconditions:
  - The user has already created a schedule.
  - The user is logged into the system.
- Postconditions:
  - The schedule is updated with the new details and saved automatically.
- Main Flow:
  - The user clicks the "Edit" button next to a class in the schedule.
  - The system opens the class details in an editable form.
  - The user makes changes (e.g., updates class times, instructor, etc.).
  - The user clicks "Save" to update the schedule.
- Alternative Flows:
  - Error Handling: If an invalid class detail is entered (e.g., overlapping class times), the system shows an error and prompts the user to fix it before saving.
- Exceptions: If the system fails to save the changes (e.g., a network issue), an error message prompts the user to retry.

4. Delete Class in Schedule

- Description: The user can delete a specific class from their schedule. This is necessary if the class is no longer needed or if there was a mistake when entering the schedule.
- Preconditions:
  - The user has already created a schedule.
  - The user is logged into the system.
- Postconditions:
  - The selected class is removed from the schedule and the updated schedule is saved.
- Main Flow:
  - The user clicks the "Delete" button next to the class they want to remove.
  - The system asks for confirmation to ensure the user wants to delete the class.
  - The user confirms the deletion.
  - The system removes the class and updates the schedule.
- Alternative Flows:
  - Error Handling: If the class cannot be deleted (e.g., due to system issues), an error message is shown.
- Exceptions: If a failure occurs during the deletion process, an alert asks the user to retry the deletion.

5. Export Schedule

- Description: After finalizing the schedule, the user can export it as a PDF for offline use or printing. This allows the user to keep a physical or digital copy of their schedule.
- Preconditions:
  - The user has a finalized schedule ready for export.
  - The user is logged into the system.
- Postconditions:
  - The schedule is saved as a PDF file on the user’s device.
- Main Flow:
  - The user clicks the "Export" button.
  - The system processes the schedule and prepares the PDF.
  - The user is prompted to save the PDF file.
  - The file is saved on the user's device.
- Alternative Flows:
  - Error Handling: If there is a problem generating the PDF (e.g., missing components), an error message will prompt the user to retry.
- Exceptions: If the system cannot generate or save the PDF (e.g., storage issues), an error message will be displayed

### 3.3. Rationale for your use case model

1. User-Centric Design: The model is designed around common actions that users need to perform, ensuring the system is tailored to student and faculty needs.

2. Simplicity: By focusing on the core features (create, view, modify, delete, and export schedules), the system ensures users have intuitive access to important features without overwhelming them with unnecessary complexity.

---

### 3.4. Non-functional requirements

- Performance: The system should load schedules in under 1 second to ensure quick access and minimize waiting time.
- Usability: An easy-to-navigate interface is essential, especially for users who may not be familiar with technology.
- Reliability: The system should work seamlessly across different platforms (e.g., web, mobile) to ensure accessibility for a wide range of users.

## 4. Architecture

---

### 4.1. Architectural style(s) used

The project employs a **Client-Server Architecture** with a **Single Page Application (SPA)** approach. The Vaadin Framework facilitates this by combining server-side Java logic with a client-side rendering process.

Key characteristics include:

- Separation of concerns between the frontend and backend.
- Event-driven communication between the user interface and the server.
- Ensuring scalability and ease of maintenance by organizing the code into logical components.

---

### 4.2. Architectural model (includes components and their interactions)

The architectural model is composed of the following key components and their interactions:

1. Frontend (User Interface Layer):
    - Built using the Vaadin Framework to provide an interactive and visually appealing user interface.
    - Handles user input and renders the schedule in real-time.
2. Backend (Business Logic Layer):
    - Written in Java 17 to process user requests, manage schedule data, and enforce scheduling rules (e.g., preventing overlapping classes).
3. Local Storage (Data Persistence):
    - Uses browser-based localStorage or sessionStorage to save user data temporarily.
    - Ensures that schedule information persists across sessions, even without a dedicated database.
4. Interactions:
    - User Actions: Users interact with the UI (e.g., adding or editing a schedule).
    - Request Handling: The frontend sends the request to the backend for processing.
    - Response Delivery: The backend processes the data and updates the UI dynamically.

---

## 4.3 Technology, Software, and Hardware Used

- Frontend: Vaadin Framework for UI development.
- Backend: Java 17 for processing business logic.
- Hardware Requirements:
    - macOS and Windows laptops for development and testing.
- Software Requirements:
    - IntelliJ IDEA as the Integrated Development Environment (IDE).
    - Vaadin libraries and Java SDK.

---

## 4.4. Rationale for your architectural style and model

The choice of architecture was driven by:

1. Team Constraints:
    - Both developers had a limited overlap in their technical expertise.
    - Java was selected as a common language to ensure effective collaboration.
2. Simplicity and Time Efficiency:
    - Vaadin's framework allowed rapid development of the UI without requiring extensive JavaScript or HTML knowledge.
    - A client-server approach simplified the development process, enabling clear division of tasks between the frontend and backend.
3. Resource Availability:
    - Vaadin and Java were accessible tools that provided sufficient functionality for the project’s scope without introducing unnecessary complexity.
4. Future Scalability:
    - The modular nature of the chosen architecture ensures that additional features (e.g., database integration or user authentication) can be added later with minimal restructuring.

## 5. Design

### 5.1. User Interface design

- The user interface (UI) design emphasizes simplicity, clarity, and accessibility to accommodate users of varying technical expertise. The primary goal is to ensure that users can seamlessly add, edit, and view their schedules in an organized manner.
- 
- The UI includes a grid layout for schedule visualization and intuitive forms for input.

---

### 5.2. Components design (static and dynamic models of each component)

**Static Models**: These are the fixed components that make up the application layout.

1. Timetable Grid: The grid structure, with columns representing the days of the week and rows representing time slots (e.g., 8:00 AM to 6:00 PM).
2. Form Layout: Includes fields for input (e.g., class name, classroom, time pickers, and day selectors).

**Dynamic Models**: These components change based on user interactions.

1. Dynamic Class Scheduling: As users add or modify a class, the grid will dynamically update to show the class details for the selected days and time slots.
2. Notifications: After any user action (adding, removing, or clearing classes), a notification is displayed to give feedback on the action.

---

### 5.3. Database design

No Database was implemented.

---

## 5.4. Rationale for your detailed design models

Grid Layout for Visualization: A grid format is familiar to most users (e.g., similar to calendar or spreadsheet applications). This allows users to easily match class times with specific days of the week.

Form Design: The form is kept simple and only includes necessary fields, such as class name, classroom, and time. Complex options are avoided to ensure that adding or editing a schedule is quick and straightforward.

Notifications: Immediate feedback ensures that users understand the results of their actions (e.g., successful addition, validation errors, etc.). This improves the user experience.

---

## 5.5. Traceability from requirements to detailed design models

1. Requirement: Users should be able to view, add, and edit their schedules easily.

   Design Solution: A grid layout was chosen to display the schedules, with intuitive form fields for input. A simple button interface facilitates adding, editing, and clearing classes.

2. Requirement: The UI must be accessible and user-friendly.

   Design Solution: Simple labels and tooltips are used in the form inputs, and clear instructions guide users through their actions. High contrast and readable fonts are implemented for better accessibility.

3. Requirement: Data should persist across sessions or page reloads.

   Design Solution: Since no server-side database was implemented, localStorage or sessionStorage was chosen to persist data on the client side. This ensures that the schedule remains even after the user refreshes the page.

4. Requirement: Prevent overlap in the schedule.

   Design Solution: The system checks for overlapping times when a user adds a class, ensuring that no two classes occupy the same time slot. Notifications inform users of conflicts.

5. Requirement: Data must be dynamically updated when a class is added or removed.

   Design Solution: The grid and form are designed to update dynamically. When a class is added, the grid is updated instantly. If a class is removed or modified, the grid reflects the changes without reloading the page.

## 6. Test Management

6.1. A complete list of system test cases

| | |
|---|---|
| ID | Valid Check |
| Test Input | Valid class name, time, and date. |
| Expected Output | Class and time is added successfully to the schedule. |
| Description | Verifies the functionality of adding a class to the planner with valid inputs. |

| | |
|---|---|
| ID | Invalid Check |
| Test Input | Enter invalid class time (e.g., overlapping times) |
| Expected Output | Error message indicating overlapping schedules. |
| Description | Ensures system validation prevents overlapping classes. |

| | |
|---|---|
| ID | Blank Generate Schedule |
| Test Input | Enter no classes and click "Generate Schedule." |
| Expected Output | Error message or prompt to enter classes. |
| Description | Ensures the system handles empty inputs gracefully. |

| | |
|---|---|
| ID | Generate Schedule |
| Test Input | Enter classes and click "Generate Schedule." |
| Expected Output | Verifies that the schedule is free of error and outputs a schedule. |
| Description | Ensures the system handles generation gracefully. |

| | |
|---|---|
| ID | Clear All Functionality |
| Test Input | Click on the "Clear All" button. |
| Expected Output | All entries are removed from the schedule. |
| Description | Checks the functionality of clearing all user input in the planner. |

| | |
|---|---|
| ID | Editing |
| Test Input | Enter and delete classes repeatedly. |
| Expected Output | Classes are correctly removed and not reappear. |
| Description | Confirms that entries are correctly deleted and removed from memory. |

| | |
|---|---|
| ID | Overlap Classes |
| Test Input | Add multiple classes with overlapping times. |
| Expected Output | Error messages for overlapping schedules; only valid classes are added. |
| Description | Validates conflict for overlapping class times. |

| | |
|---|---|
| ID | Large amount of classes |
| Test Input | Enter a very large number of classes (e.g., 50+). |
| Expected Output | Schedule is generated and displayed without performance issues. |
| Description | Validates the application's ability to handle a high volume of data. |

| | |
|---|---|
| ID | Overtime |
| Test Input | Attempt to add a class beyond the permitted time range (e.g., 3:00 AM). |
| Expected Output | Error message preventing the addition. |
| Description | Tests time range constraints for valid scheduling. |

| | |
|---|---|
| ID | Specific Time |
| Test Input | Add classes with edge-case times (e.g., 11:59 PM). |
| Expected Output | Classes are correctly displayed in the schedule. |
| Description | Validates handling of boundary time values. |

| | |
|---|---|
| ID | Save Progress |
| Test Input | Reopen the application after saving a schedule. |
| Expected Output | Previously saved schedules are loaded. |
| Description | Confirms that saved data is correctly retrieved on application restart. |

6.2. Traceability of test cases to use cases

| Use Case | Test Case ID(s) | Description |
|---|---|---|
| **Create Schedule** | Valid Check, Invalid Check, Overlap Classes, Large Number of Classes, Overtime, Specific Time | Tests the ability to create a schedule with various inputs, including edge cases and constraints. |
| **View Schedule** | Generate Schedule, Specific Time, Large Number of Classes | Verifies that the user can view the schedule correctly, even with edge cases or a large dataset. |
| **Modify Schedule** | Editing, Overlap Classes | Tests the ability to edit and update schedules, ensuring conflicts are handled appropriately. |
| **Delete Class in Schedule** | Clear All Functionality, Editing | Ensures users can delete individual classes or clear all entries from the schedule. |
| **Export Schedule** | Save Progress | Verifies that the completed schedule can be exported and saved for later use. |

6.3. Techniques used for test case generation

**Equivalence Partitioning**:
Test inputs were divided into equivalence classes to ensure representative testing of valid and invalid cases (e.g., valid class names vs. invalid inputs like overlapping times).

**Boundary Value Analysis**:
Test cases included boundary values for times, such as the earliest and latest possible time slots (e.g., 11:59 PM and 12:00 AM).

**Error Guessing**:
Scenarios likely to cause errors, such as entering overlapping class times or leaving required fields blank, were specifically tested.

**Use Case-Based Testing**:
Test cases were derived directly from the defined use cases, ensuring that all functional requirements were validated.

6.4. Test results and assessments (how good are your test cases? (How good is your software?)

- Test Coverage:
  All functional requirements were covered by the test cases, ensuring thorough validation of core features such as adding, editing, viewing, and exporting schedules.
- Effectiveness of Test Cases:
  - Most test cases successfully validated the intended functionality.
  - Edge cases, such as overlapping schedules and invalid times, were consistently caught during testing.
- Software Quality Assessment:
  - Strengths: The system performed well under typical workloads and handled user input reliably. The user interface remained responsive even when handling large datasets.
  - Areas for Improvement:
    - The lack of server-side data storage limited the ability to recover schedules across different devices.

  Overall, the software met its primary objectives and provided a robust scheduling experience for end-users.

6.5. Defects reports

| Defect ID | Description | Severity | Status | Resolution |
|---|---|---|---|---|
| DEF 1 | Overlapping classes allowed under specific conditions. | High | Resolved | Added validation logic to detect overlapping entries. |
| DEF 2 | Error message not displayed for invalid time inputs. | Medium | Resolved | Updated UI feedback for invalid time entries. |
| DEF 3 | "Clear All" button did not reset the grid completely. | Low | Resolved | Fixed clearing logic in the backend and UI synchronization. |
| DEF 4 | Performance lag with large datasets (50+ entries). | Medium | Unresolved | Requires optimization for large-scale inputs in future versions. |
| DEF 5 | Exported PDF format lacked grid alignment in some browsers. | Low | Resolved | Adjusted PDF formatting logic to ensure cross-browser compatibility. |

## 7. Conclusions

---

7.1. Outcomes of the project (are all goals achieved?)

- The project successfully implemented the following core functionalities:
    - Schedule Creation and Editing: Users can easily add, edit, and delete classes or activities in their weekly schedule.
    - User-Friendly Interface: The interface is intuitive, ensuring users can visualize and manage their schedules effectively.
    - Dynamic Updates: Real-time updates to the schedule grid make the system responsive and efficient.
    - Cross-Platform Compatibility: The application functions seamlessly on major browsers, supporting various devices.
- Goals Achieved: Most project goals were met within the given timeline and scope with an exception in some features.
- All CORE functionalities implemented successfully, meeting project goals.

---

7.2. Lessons learned

Team Collaboration:

- Dividing tasks between backend and frontend development was efficient but required clear communication to ensure smooth integration.
- Weekly meetings were essential for tracking progress and addressing challenges.

Technology and Frameworks:

- The Vaadin framework provided a quick setup for the frontend but limited customization options compared to other frameworks.
- Using Java for both backend and frontend ensured consistency but highlighted the need for more versatile language knowledge in future projects.

Project Planning:

- Following the Incremental Development Model helped refine the product iteratively, though earlier integration testing might have prevented some minor issues.

Testing Importance:

- Comprehensive test planning helped identify edge cases early, saving time in debugging later stages.

---

## 7.3. Future development

To enhance the application and broaden its usability, the following features are proposed for future development:

1. **User Authentication:**
   - Implement a login system to allow users to save and retrieve personalized schedules across devices.
2. **Data Storage:**
   - Integrate a database for long-term data storage and synchronization across sessions.
3. **Mobile Optimization:**
   - Develop a mobile-responsive design or a dedicated app to improve usability on smartphones and tablets.
4. **Additional Export Options:**
   - An export option instead of opening a new tab for printing.

## References

### Vaadin Framework Documentation

- *Vaadin Flow Documentation.* Vaadin Ltd. Available at: https://vaadin.com/docs.

### Java Development

- *Java 17 Documentation.* Oracle. Available at: https://docs.oracle.com/en/java/javase/17/.