

Software Requirements Specification for ”Exam Scheduling System”

Kerem Bozdağ 20200602012
Milena Larissa Ünal 20230602107
Rabia Tülay Gokdag 20210602029
Mehmet Sefa Keskin 20230602047
Feyza Gereme 20230602028
Emin Arslan 20230602007

Section: 1
Team: 3

November 16, 2025

SE302 - Software Engineering

1 Introduction

The Exam Scheduling System is a desktop application designed to automate the process of scheduling student examinations for educational institutions. The system will allow Student Affairs staff to import student enrollment data, course information, and classroom capacities from CSV files, and automatically generate exam schedules that satisfy all required constraints. Users will be able to view schedules from multiple perspectives (by classroom, student, course, and day), make limited manual adjustments with constraint validation, and export schedules to CSV or Excel formats for distribution. The application is planned as a standalone Windows desktop application built entirely in Java, using SQLite for local data storage, and does not depend on any external service or network connectivity.

2 User Requirements

The user requirements can be detailed as follows.

Functional Requirement 1: The user shall be able to import data from CSV files including student lists, course codes, classroom capacities, and course attendance lists.

Rationale: Since the system needs information about students, courses, and classrooms to create schedules, data import is the fundamental first step. CSV format is universally supported and can be easily generated from existing institutional databases or spreadsheet applications. The user needs to import four types of files: a list of all student IDs, a list of all course codes, a list of classrooms with their capacities (semicolon-separated format), and attendance lists showing which students are enrolled in which courses. The system should validate the imported data and detect duplicate entries to prevent scheduling errors.

Functional Requirement 2: The user shall be able to configure exam period parameters including the number of exam days and the number of time slots per day.

Rationale: Different institutions or exam periods may have different durations and daily schedules. For example, one institution might schedule exams over 5 days with 4 time slots per day, while another might use 7 days with 3 slots per day. The system must allow users to configure these parameters before generating a schedule so that the scheduling algorithm works within the correct time constraints.

Functional Requirement 3: The user shall be able to select an optimization strategy before generating the schedule.

Rationale: There may be multiple valid schedules that satisfy all hard constraints, but different institutions may have different preferences. Some may want to complete all exams in the minimum number of days, while others prefer to distribute exams evenly across the entire exam period to balance the workload. The system should provide configurable optimization strategies such as minimizing the number of days used, balancing exam distribution across days, or minimizing the number of classrooms needed.

Functional Requirement 4: The user shall be able to automatically generate an exam schedule.

Rationale: This is the core functionality of the system. The schedule generation must satisfy all hard constraints: no student can have exams in consecutive time slots, no student can have more than 2 exams on any single day, and no classroom can exceed its capacity. The system should use a constraint satisfaction algorithm to find a valid schedule. If multiple valid schedules exist, the system should apply the user-selected optimization strategy to choose the best one.

Functional Requirement 5: The user shall be able to view generated schedules in four different formats: by classroom, by student, by course, and by day.

Rationale: Different stakeholders need different views of the schedule. Facilities management needs to see which exams are in which classrooms, students need their personal exam schedules, faculty need to know when their course exams are scheduled, and administrators need a day-by-day overview of all exams. Providing all four views ensures the schedule can be used effectively for different purposes.

Functional Requirement 6: The user shall be able to export generated schedules to CSV or Excel format.

Rationale: Once a schedule is generated, it needs to be distributed to students, faculty, and administrators. CSV format is universally compatible and can be imported into other applications, while Excel format provides better formatting options and is preferred by many users for printing and distribution. Export functionality is essential for making the schedule accessible beyond the application itself.

Functional Requirement 7: The user shall be able to make limited manual adjustments to generated schedules.

Rationale: In some cases, manual adjustments may be necessary due to special circumstances such as instructor preferences, room maintenance, or last-minute changes. However, manual edits should not violate the hard constraints. The system must validate any manual change to ensure it doesn't create consecutive exams for students, exceed the 2-exams-per-day limit, or violate classroom capacity constraints. If a proposed change violates constraints, the system should reject it and display an error message explaining which constraint would be violated.

Non-functional Requirement 1: The user shall be able to use the system after reading the user manual.

Rationale: Since the system is used infrequently (typically once or twice per semester), users may not remember all features between uses. The system should include comprehensive built-in documentation covering installation, data import procedures, schedule generation, and export functions. The user interface should be intuitive with clear labels, and the user manual should enable a new user to successfully generate their first schedule within 30 minutes.

Non-functional Requirement 2: The system shall provide clear and actionable feedback when errors occur.

Rationale: Users need to understand what went wrong and how to fix it. Error messages should be user-friendly rather than technical, and should provide specific guidance. For example, instead of showing a generic error, the system should indicate "Cannot import: Duplicate student ID 'Std_ID_123' found on lines 45 and 67" or "Student Std_ID_089 would have 3 exams on Day 2, which exceeds the maximum of 2 exams per day."

3 System Requirements

The system requirements can be detailed as follows.

Functional Requirement 8: The system shall store all imported data and generated schedules in a local SQLite database.

Rationale: Data persistence is necessary so users don't have to re-import files every time they open the application. SQLite is a lightweight, file-based database that requires no separate server installation and is ideal for desktop applications. The database should store students, courses, classrooms, enrollments, and complete schedules with all exam assignments. This allows users to close and reopen the application without losing their work, and enables quick access to previously generated schedules.

Functional Requirement 9: The system shall guarantee that all generated schedules satisfy 100% of the hard constraints.

Rationale: Violating hard constraints would invalidate the schedule and cause serious problems during the exam period. The system must enforce that no student has consecutive exams (exams in adjacent time slots), no student has more than 2 exams on any single day, and no exam exceeds classroom capacity. The scheduling algorithm must validate these constraints before finalizing any schedule, and any schedule that violates these constraints must be rejected.

Functional Requirement 10: The system shall detect and report specific conflicts when no valid schedule can be generated.

Rationale: When the scheduling algorithm cannot find a solution that satisfies all constraints, the user needs actionable information about why the schedule failed and what can be done to fix it. The system should analyze which constraints are being violated and identify specific problems such as students with too many course enrollments, insufficient time slots, or inadequate classroom capacity. The conflict report should show which students or courses are causing problems and provide recommendations such as extending the exam period, adding more classrooms, or adjusting course enrollments.

Functional Requirement 11: The system shall validate manual schedule edits before applying them.

Rationale: When users make manual adjustments to a schedule, the system must ensure these changes don't introduce constraint violations. Before applying any manual edit, the system should check that the modification doesn't create consecutive exams for any affected student, doesn't cause any student to exceed 2 exams per day, and doesn't exceed the capacity of the new classroom. If the edit is valid, the change should be applied and saved to the database. If invalid, the system should reject the change and display a detailed explanation of which constraint would be violated.

Functional Requirement 12: The system shall allow users to load and view previously saved schedules.

Rationale: Users may need to reference, modify, or export schedules that were generated in previous sessions. The system should maintain a history of all generated schedules in the database, display them with metadata such as creation date and optimization strategy used, and allow users to load any saved schedule for viewing, editing, or exporting. This provides flexibility and ensures that work is never lost.

Non-functional Requirement 3: The system shall run on Windows 10 and Windows 11 operating systems.

Rationale: The target platform is Windows desktop environments, which are standard in most educational institutions. The application should be developed in Java, which provides portability, but the primary requirement is Windows compatibility. Minimum system requirements should include a dual-core processor (2.0 GHz or faster), 4 GB RAM, and 100 MB of disk space for the application and typical data. The application should run efficiently on standard office computers without requiring specialized hardware.

Non-functional Requirement 4: The system's schedule generation performance shall scale based on dataset size.

Rationale: Exam scheduling is a constraint satisfaction problem that can be computationally intensive. The time required to generate a schedule depends on the number of courses, students, classrooms, and the complexity of the constraints. For small datasets (50 or fewer courses with 500 or fewer students), the system should complete schedule generation within 10 seconds. For larger datasets, generation may take several minutes. The system should display a progress indicator during schedule generation so users know the system is working and can cancel long-running operations if needed.

Non-functional Requirement 5: The system shall maintain data integrity and prevent data loss.

Rationale: The system handles critical institutional data that affects hundreds of students. All database operations must use transactions to ensure data consistency. The system should validate all user inputs before processing, maintain referential integrity in the database (ensuring that attendance lists only reference students and courses that actually exist), and handle errors gracefully without crashing or corrupting data. Users should be able to manually backup the database file, and the system should provide a menu option to export the database to a safe location.

Non-functional Requirement 6: The system shall be maintainable and extensible.

Rationale: Future requirements may necessitate new features such as additional optimization strategies, new export formats, or additional constraints. The system should follow a modular architecture with clear separation between the user interface layer, business logic layer (scheduling algorithms and constraint validation), and data access layer (database operations and file I/O). Code should follow Java naming conventions, include appropriate comments, and use meaningful variable and method names to facilitate future maintenance and feature additions.