CHAPTER 6

SYSTEM TESTING

6.1 Introduction

System testing is one of the most crucial phases of the software development lifecycle. It ensures that the complete and integrated system works as intended, meeting both functional and non-functional requirements. For the web-based automatic timetable scheduler, system testing was particularly important since the system directly impacts academic planning, scheduling efficiency, and faculty management.

The primary objectives of system testing were to validate that the scheduler generates conflict-free timetables, handles faculty availability constraints, adheres to the requirement of sequential slot allocation, and provides a user-friendly interface. A combination of unit testing, integration testing, functional testing, and overall system testing was employed to comprehensively evaluate the scheduler's performance and reliability.

6.2 Unit Testing

Unit testing involves testing individual components or modules of the system in isolation to ensure they function correctly. Here, unit testing focused on two major components: the Python back-end modules and the Streamlit-based front-end interface.

The Python back-end formed the core of the scheduler, handling data retrieval, validation, and timetable generation using the Genetic Algorithm.

- **Fitness Function**: Ensured that penalties for conflicts faculty overlap, and lab schedules were correctly applied.
- **Database Operations**: Verified that data retrieval functions such as fetching faculty details, course mappings, and preferences worked correctly without returning null or incorrect values.
- **Algorithmic Components**: Tested crossover and mutation functions to confirm that new candidate timetables were valid and maintained scheduling rules.

The Streamlit interface provided the user interaction. Unit tests for this component focused on input validation and display correctness. Specific tests included:

- **Form Validation**: Verified that all mandatory fields (e.g., faculty name, course selection, semester details) were required before submission.
- **Error Messages**: Checked that clear, user-friendly messages were displayed for invalid inputs (e.g., missing course mapping or duplicate entries).
- **Output Display**: Confirmed that generated timetables were displayed correctly within the Streamlit dashboard, maintaining clarity and alignment.

6.3 Integration Testing

Integration testing was carried out once individual modules were validated, with the objective of ensuring that the modules interacted seamlessly. The focus areas included:

- **Database and Algorithm Integration**: Ensured that real-time faculty-course data from the database was correctly passed into the scheduling algorithm.
- **Algorithm and Interface Integration**: Verified that generated timetables were correctly displayed on the Streamlit interface without mismatched or missing data.

6.4 Functional Testing

Functional testing validated the system against the defined requirements. The main areas tested were:

- **Conflict-Free Allocation**: The system correctly prevented double-booking of faculty across different batches.
- **Faculty Availability**: The scheduler respected unavailability constraints, ensuring no allocations during blocked slots.
- **User Interaction**: Verified that users could easily input data, generate timetables, and export results with minimal steps.

Each function was tested, including typical academic data and simulated different scenarios. The results indicated that the system consistently fulfilled functional requirements.

6.5 System Testing

System testing encompassed the evaluation of the scheduler as a whole in a real-world environment. Unlike unit and integration testing, which focused on individual modules, system testing validated the complete functionality and performance of the integrated application.

- The scheduler could handle multiple batches and faculty members simultaneously without conflict.
- Timetable generation was completed within a reasonable time frame, even for larger datasets.
- Error handling mechanisms, such as invalid input warnings, worked effectively to prevent system crashes.

The results of system testing demonstrated that the timetable scheduler was stable and reliable.

6.6 Test Cases

To validate the functionality and performance of the Web-Based Automatic Timetable Scheduler, test cases was developed. Each test case outlines the input conditions, expected output, and actual results observed during tests are illustrated in Table 6.1. The successful execution of all test cases confirms the system's correctness, reliability, and readiness for deployment.

Table 6.1: Test Cases

Test Case No.	Description	Input	Expected Output	Actual Output	Result
1	Faculty availability constraint	Faculty A unavailable on Monday Slot 2	Faculty A not scheduled in Slot 2	Faculty A not scheduled in Slot 2	Passed
2	Conflict-free scheduling across semesters	Faculty B assigned to multiple semesters	Faculty B not double-booked in same slot	Faculty B scheduled in separate slots	Passed
3	Invalid input handling	Faculty entry without course mapping	Error message, input rejected	Error message displayed, input rejected	Passed
4	Performance test	9 faculty, 3 Semester, 20 courses	Timetable generates within few minutes/ seconds	Generated in 6.4 seconds	Passed

6.7 Summary of Testing

System testing verified that all components of the Web-Based Automatic Timetable Scheduler for Colleges functioned correctly and cohesively. The system consistently generated conflict-free timetables, handled faculty constraints effectively, and maintained reliable performance. The system has proven to be robust, accurate, and efficient. It meets all its design and functional objectives, making it fully suitable for real-world academic scheduling scenarios. The successful completion of testing provides confidence that the scheduler can be deployed for institutional use, ensuring automation, accuracy, and improved productivity in timetable generation.