Project Report

On

Timelyflow-Smart Timetable Generator

Submitting in the fulfillment of the requirement for the award of the degree

of

Bachelor of Technology

in

Computer Science and Engineering

Batch (2022-2026)

Submitted To:

Ms.Mehak Kohli

Assistant Professor

Submitted By:

Meenakshi

Reg No.-12200149

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DAV UNIVERSITY

JALANDHAR

ACKNOWLEDGEMENT

I express my heartfelt gratitude to all those who provided invaluable support and guidance throughout the development of this project.

Firstly, I would like to extend my sincere gratitude and indebtedness to **Dr. Rahul Hans** (Head of Department) and my mentor **Ms. Mehak Kohli** for their constant encouragement and support during this journey.

I am also deeply thankful to all the faculty members of the Department of Computer Science and Engineering for their technical guidance, inspiration, and assistance in preparing the final report and presentation.

Finally, I would like to thank my family, peers, and friends for their unwavering motivation, patience, and support throughout the entire process of this project

DECLARATION

I, Meenakshi, hereby declare that the work presented in this project titled “TimelyFlow – Smart Timetable Generator” is an authentic record of my own efforts. This project has been completed in partial fulfilment of the requirements for the award of the Bachelor of Technology (B. Tech) Degree in Computer Science and Engineering under the guidance of **Dr. Rahul Hans** (Head of Department).

To the best of my knowledge, the content of this report has not been submitted to any other University or Institute for the award of any degree or diploma .

Meenakshi

12200149

ABSTRACT

TimelyFlow – Smart Timetable Generator is a comprehensive web-based application designed to address the challenges of managing academic schedules in schools, colleges, and universities. Creating an effective timetable is often a tedious and error-prone process, requiring careful coordination of faculty availability, classroom capacities, subject requirements, and institutional constraints such as lunch breaks, laboratory sessions, or special events. TimelyFlow simplifies this complex task through an interactive, automated, and user-friendly platform built with Python and Streamlit.

The system leverages intelligent scheduling algorithms to automatically generate conflict-free weekly timetables, dynamically assigning subjects, faculty members, and classrooms while respecting predefined constraints such as room capacity, faculty workload limits, and mandatory breaks. Users can also make manual adjustments in real time through an intuitive drag-and-drop or form-based interface, ensuring that last-minute changes—such as faculty substitutions or emergency room reassignments—are handled with ease.

Key features include seamless data import and export for CSV or Excel files, enabling quick input of course data and one-click export of finalized timetables for distribution. The application offers an engaging dark-mode interface with animated backgrounds and a visually appealing, responsive design that minimizes eye strain during extended use. Additionally, an interactive pivot-grid visualization provides a clear, filterable view of timetables by class, faculty, or room, updating instantly as changes occur.

By integrating automation with flexibility, TimelyFlow significantly reduces administrative workload, improves scheduling accuracy, and ensures equitable distribution of teaching hours. Beyond time savings, it enhances transparency for students, faculty, and administrators, allowing each stakeholder to view or download updated timetables in real time.

This project highlights the effective application of data-driven methods and interactive web technologies to a real-world problem. It showcases how modern Python libraries, efficient data handling, and responsive UI design can transform a traditionally manual, error-prone process into a fast, intelligent, and accessible scheduling solution, ultimately streamlining academic operations and elevating the overall educational experience.

INDEX

CHAPTER 1:

Introduction

1.1 Background

Efficient timetable management is one of the most essential administrative tasks in any educational institution. Whether it is a school, college, training center, or university, timetables play a critical role in coordinating academic activities, allocating resources, avoiding conflicts, and ensuring smooth functioning of the teaching–learning process. A timetable is not merely a schedule; it is a complex system of interrelated components such as faculty availability, room capacity, subject distribution, student groups, and institutional constraints.

Traditionally, timetables have been created manually using charts, spreadsheets, or paper-based formats. While these methods may be manageable for small institutions, they quickly become inefficient, error-prone, and time-consuming as the number of courses, departments, and teachers increases. Manual timetable creation also makes it difficult to track faculty workload, ensure optimal resource usage, verify room availability, and prevent slot clashes for students.

Digital transformation in educational administration has led to increasing adoption of automation tools. Modern institutions expect smart, flexible, user-friendly systems that can automatically generate timetables, handle constraints, and adapt to changes without starting from scratch. However, many existing tools are complex, expensive, or lack the customization needed by smaller institutions.

To address these challenges, TimelyFlow is designed as a simple, interactive, and automated timetable generation system using a lightweight web interface developed with Streamlit. With a focus on usability, flexibility, and speed, TimelyFlow offers institutions a smarter way to manage timetables.

1.2 Problem Statement

Educational institutions often face major difficulties in manually preparing and managing timetables. These challenges include resource conflicts, excessive manual work, limited flexibility, and difficulty maintaining updated records.

The primary problems that TimelyFlow aims to solve are:

1. Complexity of Manual Scheduling

Preparing a timetable involves handling multiple constraints—faculty availability, room allocation, subject load distribution, and student batch timings. Manually cross-checking all these variables is tedious and prone to human error.

2. Difficulty in Avoiding Clashes

Timetables must ensure that:

- No two classes for the same batch overlap.
- No teacher is assigned to two classes at the same time.
- Rooms are not double-booked.
- Lunch hours and breaks are properly accommodated.

3. Time-Consuming Updates

Any change in faculty availability, room capacity, or subject distribution forces administrators to modify the entire structure. Manual editing of large timetables is slow and inconvenient.

4. Lack of a User-Friendly Interface

Many existing tools have complicated navigation or require technical skills. Institutions need a system that is simple, intuitive, and operable by non-technical staff.

5. Poor Data Organization

If faculty lists, room lists, and generated timetables are maintained in scattered spreadsheets or documents, it becomes difficult to maintain consistency or track data.

TimelyFlow aims to provide a structured, automated, and efficient solution to overcome these problems.

1.3 Objectives

The primary objective of TimelyFlow is to automate the timetable generation and management process in an efficient, user-friendly manner.

The system is designed with the following detailed goals:

1. Automate Timetable Creation

Automatically generate optimized timetables based on user-defined settings such as number of days, time slots, and lunch break rules.

2. Provide a Simple and Interactive Interface

Use Streamlit to create a modern GUI that allows users to add faculties, rooms, and subjects without requiring technical knowledge.

3. Prevent Timetable Conflicts

Integrate validation logic to ensure:

- No room clash
- No batch clash
- No allocation during locked/lunch periods

4. Ensure Flexibility

Enable users to:

- Add/edit/delete faculty and room lists
- Adjust time slots
- Manually edit the timetable
- Upload external datasets (CSV/XLSX)

5. Support Data Persistence

Use internal JSON storage and Excel export so that generated timetables and resources are not lost after each session.

6. Enable Scalability

Design the system such that it can later support:

- Multiple departments
- Custom rules
- More advanced scheduling algorithms

7. Enhance Institutional Productivity

Reduce administrative workload, ensure accuracy, and improve resource utilization.

1.4 Scope of the Project

The scope defines what the TimelyFlow system will and will not include in the current implementation. This helps ensure clear project boundaries and realistic expectations.

In-Scope (What the System Will Do)

1. Faculty & Room Management
    - Add, edit, or remove faculty members
    - Add or remove rooms with capacities
2. Automatic Timetable Generation
    - Randomized slot allocation logic
    - Optional lunch slot rules
    - Clash-free scheduling
3. Manual Timetable Editing
    - Add/delete individual slot entries
    - Update specific rows
4. Dataset Uploading & Previewing
    - Upload CSV/XLSX files
    - Preview and validate the dataset
5. File Export Functionality

- Export generated timetable as Excel
- Download datasets

6. Simple & Interactive UI
    - User-friendly navigation
    - Form-based inputs
    - Clear table views
7. Local Data Storage
    - Store faculty, room, and allocated timetable data in JSON format
    - Load existing data automatically at startup

Out-of-Scope (What the System Will NOT Do for Now)

1. Complex AI-based Scheduling:-The current system does not implement genetic algorithms, constraint solvers, or machine learning optimization.
2. Multi-Department Hierarchical Structure:-Present version handles a single department or batch structure.
3. Online Database Integration:-The system currently uses local JSON files instead of SQL/Cloud storage.
4. Mobile App Version:-Only a web-based Streamlit application is implemented.
5. Automatic Faculty Workload Balancing:-Allocation is random and does not measure workload fairness.
6. Advanced Security or Authentication:-No login system is included.

CHAPTER 2:

System Overview

2.1 About TimelyFlow

TimelyFlow is a lightweight, intelligent, and user-friendly web-based application designed to automate the process of timetable generation for educational institutions. Built using Python and Streamlit, TimelyFlow aims to simplify the complexities of scheduling by offering an interactive user interface, automated logic-driven allocation, and flexible editing options.

The system is designed to be accessible for administrators, coordinators, and faculty members who may not possess technical expertise. It eliminates the dependency on traditional manual scheduling, which is not only slow but also error-prone when dealing with multiple constraints.

TimelyFlow offers a centralized platform where faculty, rooms, subjects, and time slot details can be easily added, stored, edited, and utilized to generate a clash-free timetable. Its architecture is modular, meaning each feature—such as auto-generation, manual editing, dataset uploading, or exporting—functions as an independent module but works seamlessly with others.

The use of Streamlit allows the system to run directly in a browser without requiring complex installations or a backend framework. Users only need to run the Streamlit application locally or deploy it on a cloud-hosted environment. The UI is clean, responsive, and designed with simplicity in mind so that users can perform tasks efficiently.

2.2 Key Features

TimelyFlow contains a wide range of features designed to assist users throughout the timetable creation process. These features ensure that the system is easy to operate, adaptable to institutional needs, and capable of efficiently managing scheduling data.

1. Faculty Management

- Add or delete faculty names.
- Separate lists for different departments or subjects.
- Instant updates reflected throughout the application.
- Stored locally in JSON format for persistence.

2. Room Management

- Add rooms along with capacities or numeric identifiers.
- Prevents double-booking by validation logic.
- Helps institutions with multiple classrooms or labs.

3. Automatic Timetable Generation

- Allocates subjects, faculty, and rooms using random logic.
- Ensures no time slot conflicts for faculty or rooms.
- Supports a customizable number of:
    - Days
    - Lecture slots
    - Lunch breaks
- Can regenerate new timetables instantly if changes occur.

4. Manual Timetable Entry

- Allows users to add specific timetable entries manually.
- Useful for correcting or customizing automatically generated output.
- Supports row-wise or single-entry updates.

5. Dataset Upload & Preview

- Uploads CSV/XLSX files containing pre-existing timetables.
- Displays uploaded content in a table format for verification.
- Ensures compatibility with external datasets.

6. File Export Options

- Export generated timetable to Excel format.
- Download faculty, room, or slot configurations.
- Useful for:
    - Sharing
    - Printing
    - Archiving

7. Local Data Storage

- Stores all critical lists in JSON files.
- Ensures data is automatically loaded when the application restarts.
- Helps maintain continuity.

8. User-Friendly Streamlit Interface

- Clean and minimalistic layout.
- Uses forms, buttons, tables, and tabs for intuitive navigation.
- Works smoothly on any desktop browser.

9. Clear Data Validation

- Avoids:
    - Teacher conflicts
    - Room conflicts
    - Overlapping slots
- Ensures clean and consistent scheduling.

These combined features make TimelyFlow a complete solution for automated timetable management.

2.3 Technology Stack

The technology stack defines the tools, frameworks, and libraries used to build the TimelyFlow system. Each component is chosen to maximize simplicity, performance, and ease-of-use for both developers and end users.

Frontend Technology: Streamlit

Streamlit serves as the primary frontend framework, enabling rapid development of interactive dashboards without requiring traditional HTML, CSS, or JavaScript coding.

Reasons for choosing Streamlit:

- Minimal coding required for UI development.
- Runs directly in a web browser.
- Live reloading during development.
- Fast rendering of tables, charts, forms, and buttons.
- Excellent for data-driven applications.

Backend Technology: Python

Python is the core programming language used for implementing all logic modules, such as data processing, random allocation, and file generation.

Advantages:

- Clean syntax and readability.
- Large ecosystem of libraries.
- Easy integration with data formats (JSON, CSV, Excel).
- Ideal for small to mid-sized automation projects.

Data Storage Format: JSON

JSON files are used to store:

- Faculty list
- Room list
- Generated timetable
- Configuration details

Why JSON?

- Lightweight
- Human-readable
- Easy to modify

- Fast for small datasets
- Perfect for local, offline applications

Data Handling & Processing: Pandas

Pandas makes it easy to manage tabular data, such as timetables or uploaded datasets.

Used for:

- DataFrames
- CSV/XLSX loading
- Table operations
- Exporting to Excel

File Export: openpyxl

This library allows exporting timetables to Excel with proper formatting.

Random Allocation Logic:- Python's built-in random module is used to generate timetable slot assignments based on available:

- Days
- Rooms
- Faculty
- Subjects

CHAPTER 3:

System Requirements & Design

3.1 Functional Requirements

Functional requirements define what the system must perform from the user's perspective. These identify the core features, operations, and behaviors expected from TimelyFlow. Since the system deals with managing faculty, rooms, subjects, and timetable slots, the following functional requirements outline the necessary actions and interactions.

1. Faculty Management

- The system must allow users to add new faculty names.
- The system must allow users to delete unnecessary faculty entries.
- Faculty data must be saved in a JSON file for persistence.
- Updated lists should be immediately available to other modules such as timetable generation.

2. Room Management

- Users should be able to add rooms with unique identifiers or capacity data.
- The system should prevent duplicate room entries.
- Users should be able to delete rooms.
- Room details must also be stored in JSON format.

3. Auto Timetable Generation

- The system must generate a complete timetable when the user initiates the process.
- It must allocate:
    - Day
    - Time slot
    - Faculty
    - Room
- Allocation must follow internal random logic or predefined rules.
- TimelyFlow must ensure no conflicts such as:

    - Same faculty assigned to two slots at once
    - Same room assigned to multiple classes
    - Batch clashes
- The system must support configurable options such as:
    - Number of working days
    - Number of lecture slots
    - Lunch break positioning

4. Manual Timetable Entry

- Users must be able to manually add specific timetable rows.
- Manual additions should follow validation rules to prevent conflicts.
- The system must allow updating or removing individual entries.
- Manual entries must merge seamlessly with automated ones.

5. Dataset Upload & Preview

- The system should allow users to upload CSV/XLSX files containing external timetables or resource lists.
- It must read and display the uploaded file in a tabular format.
- The system must validate:
    - Empty values
    - Missing columns
    - Invalid file types
- Users should be able to preview and confirm the dataset before further actions.

6. Data Persistence

- All crucial data (faculty list, room list, timetable) must be stored locally using JSON.
- The system must auto-load previous data whenever the application restarts.
- Generated timetables must not be lost unless explicitly cleared.

7. File Handling & Export

- Users must be able to export generated timetables to Excel.

- Export should maintain the structure and formatting of the timetable.
- Users may download the current dataset for offline use.

8. User Interface (UI) Navigation

- Users should be able to navigate between:
  - Faculty management
  - Room management
  - Timetable generation
  - Manual entry
  - Dataset upload
  - Export features
- Buttons, tabs, and forms must respond instantly using Streamlit dynamic components.

3.2 Non-Functional Requirements

Non-functional requirements define how the system behaves rather than what it does. They contribute to usability, reliability, and overall system performance.

1. Performance Requirements

- Timetable generation should complete within 1–2 seconds.
- UI components should load instantly without lag.
- Dataset uploads should be processed within reasonable time limits depending on file size.

2. Usability Requirements

- The system must be easy to navigate for non-technical users.
- All pages should follow a consistent layout and intuitive flow.
- Labels, buttons, and forms must be clearly visible and understandable.
- Feedback messages (success, error, warning) must appear instantly.

3. Reliability Requirements

- Data stored in JSON files should not get corrupted even if the app is closed abruptly.

- Generated timetables should be predictable and reproducible under similar conditions.
- File exports should work correctly for all supported formats.

4. Maintainability

- Code must be modular, with separate sections for:
    - Data storage
    - UI design
    - Timetable logic
    - Upload & export modules
- Clear function names and documentation should support easier updates.

5. Scalability

- The application should support increasing numbers of:
    - Faculties
    - Rooms
    - Time slots
    - Subjects
- JSON-based storage should handle moderate datasets without performance issues.
- The system can later integrate databases or complex scheduling algorithms.

6. Portability

- The application should run on:
    - Windows
    - macOS
    - Linux
- Only Python and Streamlit installation should be required.

7. Security Requirements

- Since this is a local system:
    - No login is required.
    - No sensitive data is transferred online.

- System must handle incorrect file uploads safely without breaking.

8. Data Integrity

- Duplicate entries should be prevented.
- All lists must autocomplete and update consistently.
- Timetable entries must follow validation rules at all times.

3.3 System Architecture

TimelyFlow follows a modular architecture with clear separation of logic, UI, and data layers. This improves maintainability and allows future expansion.

Architecture Layers

1. Presentation Layer

This layer includes:

- Forms for adding faculty, rooms, and timetables
- Buttons for generating and exporting timetables
- Tables for data preview
- Tabs/pages for navigation

Streamlit handles:

- User interaction
- Real-time updates
- Display of output data

2. Application Logic Layer

This layer contains the core logic of TimelyFlow:

- Random allocation algorithm
- Validation logic
- Lunch-time rule enforcement
- Manual entry processing

- Dataset upload handling
- DataFrame manipulation with Pandas

All logical operations are performed here before results are shown to the UI.

3. Data Layer (Local Storage)

The system uses JSON files for:

- Faculty list
- Room list
- Generated timetable
- Configuration

These files act as lightweight local databases.

4. Export Layer

- Excel file creation using openpyxl
- Data conversion to downloadable formats
- In-memory file handling with BytesIO

This architecture ensures that each module is independently manageable and improvements in one area will not break the others.

3.4 Data Flow

The data flow describes how information travels through the system from input to output. The following steps outline typical usage flow inside TimelyFlow.

Step 1: User Input

- User enters faculty or room data using Streamlit forms.
- User selects number of days/slots for auto generation.
- User uploads CSV/XLSX if needed.

Step 2: Processing & Validation

- System validates:
    - Duplicate entries
    - Empty fields
    - Incorrect data types
- If uploading dataset:
    - Pandas reads and processes the file
    - Errors are displayed if format is invalid

Step 3: Allocation Logic

- Random values are selected for:
    - Faculty
    - Rooms
    - Time slots
- Conflict detection ensures:
    - Teacher is not double-booked
    - Rooms are not reused simultaneously
- Lunch slot is inserted as per rules.

Step 4: Data Storage

- All validated lists and generated tables are stored in:
    - faculty.json
    - rooms.json
    - timetable.json

Step 5: Output Display

- The timetable is shown in an interactive table in the UI.
- Users can scroll, review, or edit rows manually.

Step 6: Export

- Users can export the final timetable to Excel.

CHAPTER 4:

GUI Development

4.1 Introduction

System design defines the architecture, modules, interfaces, and data flow of the timetable generation system. This chapter covers the structural and functional design, including the architectural model, module descriptions, data flow diagrams, and the Graphical User Interface (GUI) layout. Screenshots are added as placeholders so you can replace them with your actual project screenshots.

4.2 System Modules

4.2.1 Faculty Management Module:-

- Add new faculty
- Delete/update faculty

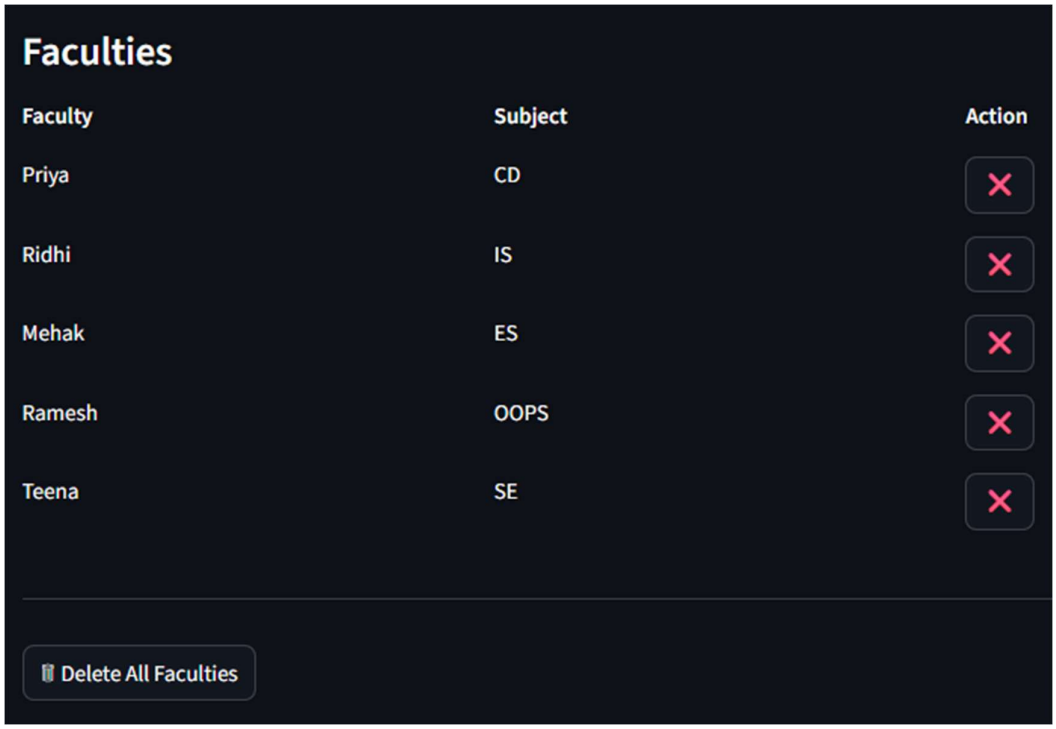


Figure 4.1: Faculty Module Screen

4.2.2 Room Management Module:- Handles room details such as room number, type, and capacity.

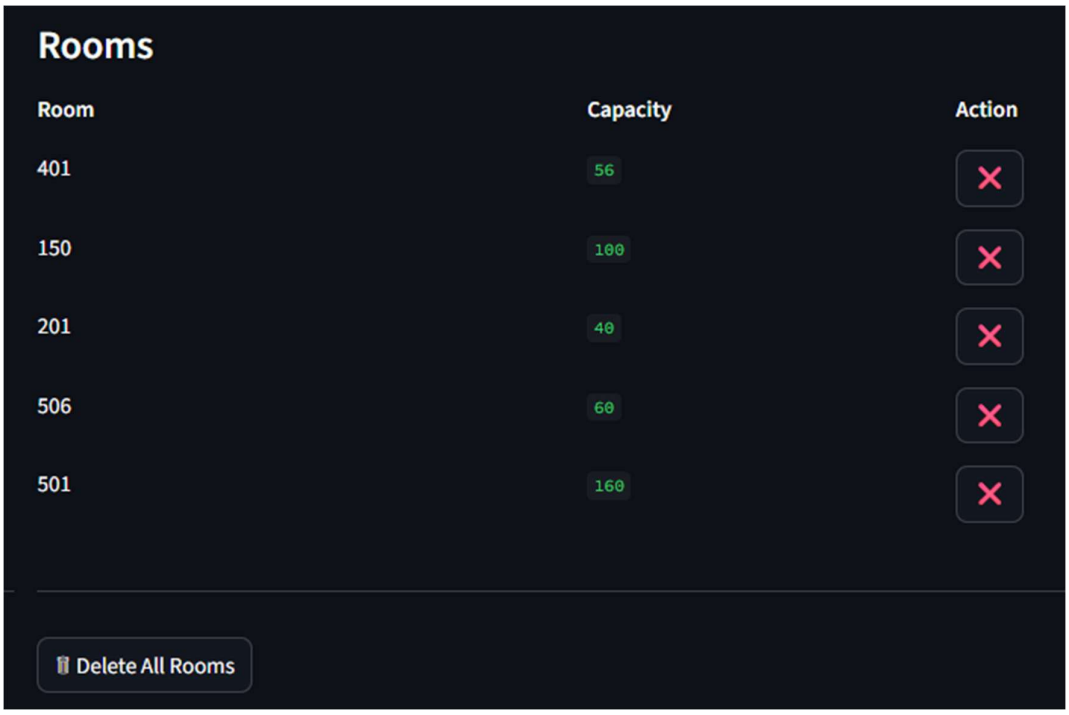


Figure 4.2: Room Management Interface

4.2.3 Dataset Management Module:-Allows uploading external datasets (CSV/XLSX), previewing, and cleaning.

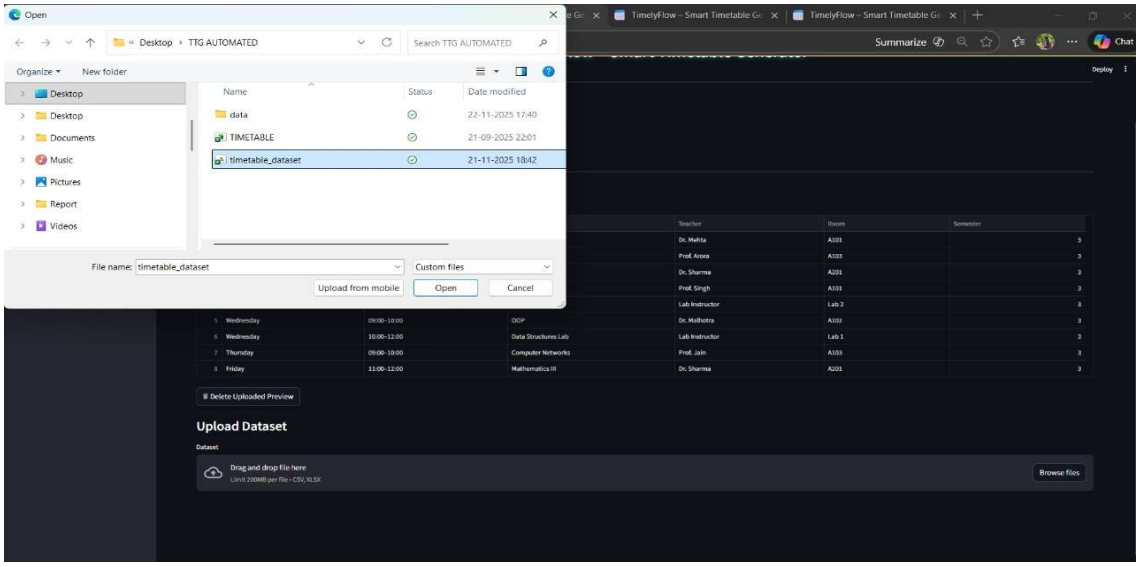


Figure 4.3: Dataset Upload Screen

4.2.4 Timetable Generation Module

The core module where the automatic timetable generator processes every constraint and creates conflict-free schedules.

- Slot allocation
- Lunch break rules
- Clash detection

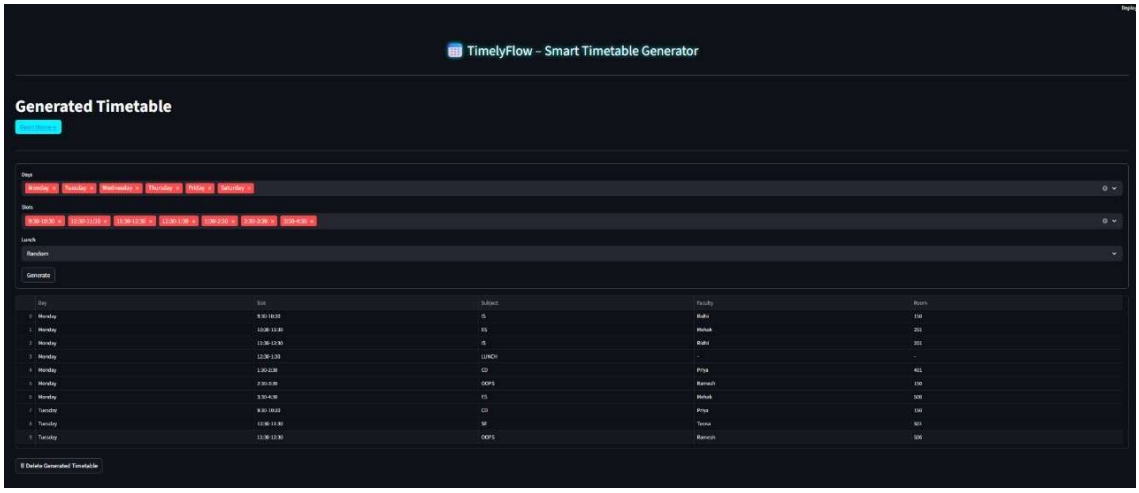


Figure 4.4: Timetable Generation Screen

4.2.5 Manual Timetable Entry Module

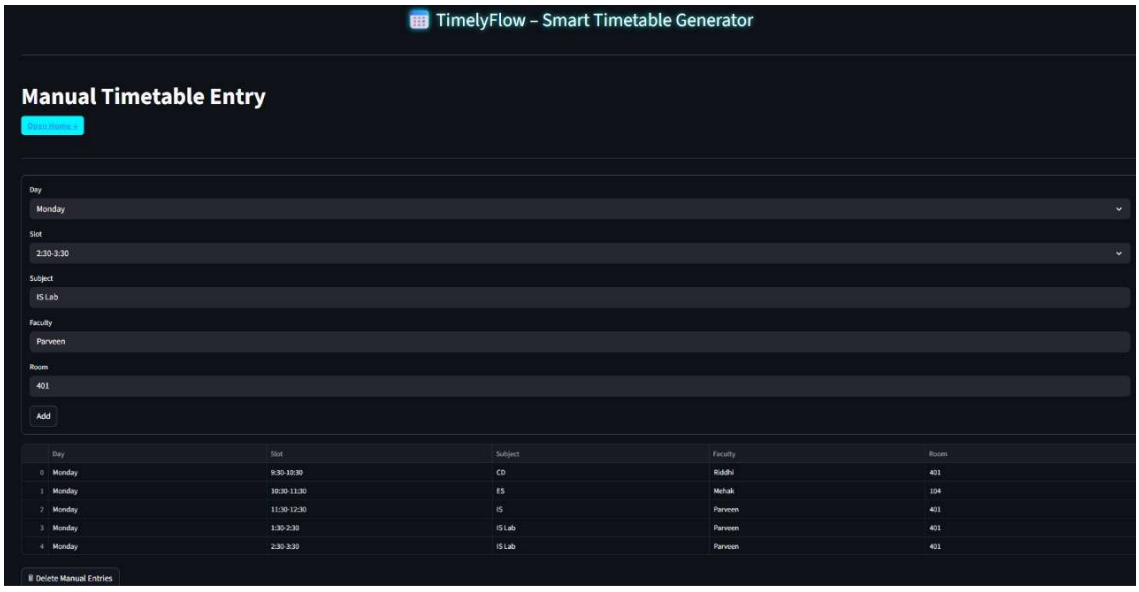


Figure 4.5: Manual Entry Interface

4.3.6 Generated Timetable Preview & Export Module

Shows the final timetable and exports it to Excel via openpyxl.

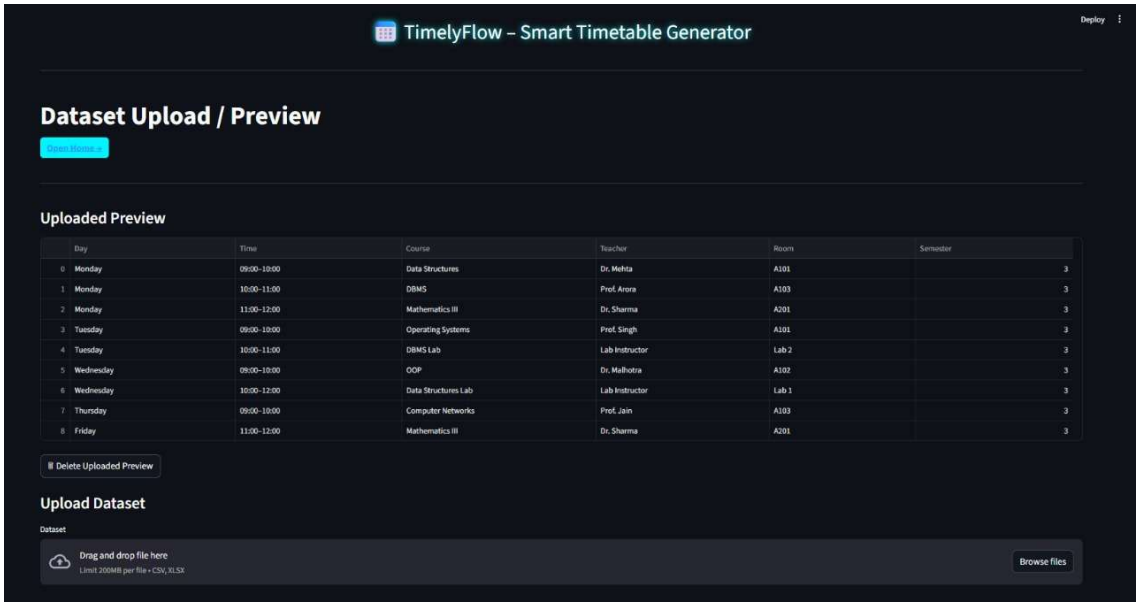


Figure 4.6: Generated Timetable Preview

4.3 User Interface Design Strategy

The UI uses Streamlit, which offers:

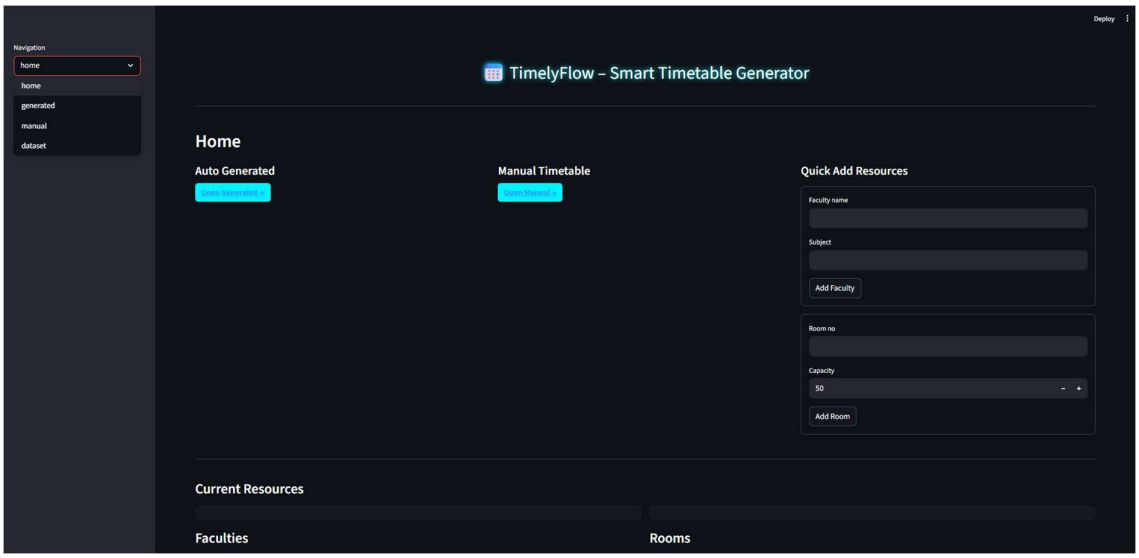
- Clean layout
- Interactive widgets
- Real-time updates
- Easy data upload and preview

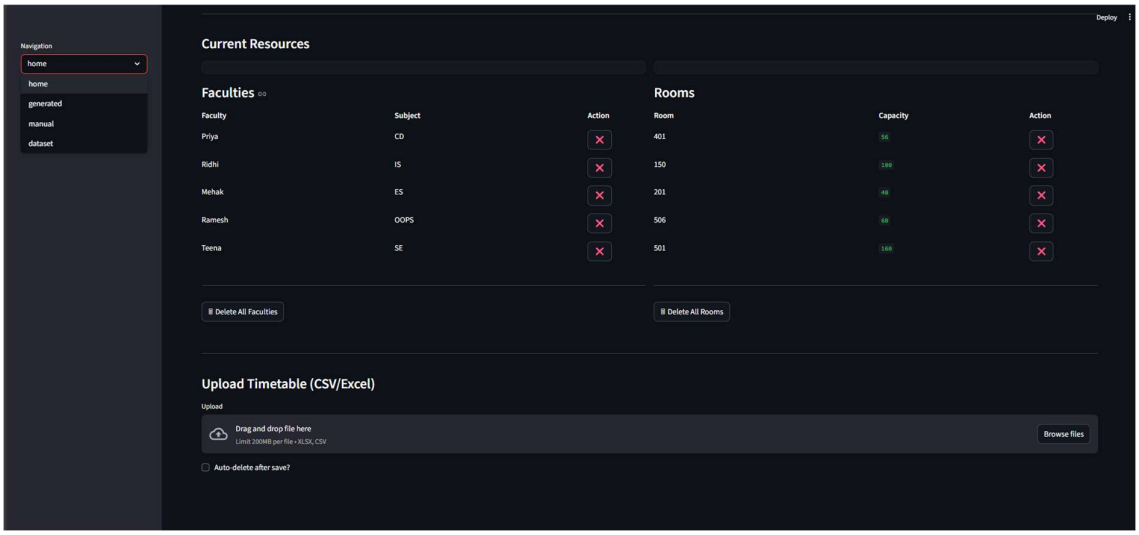
Design goals:

- Minimal clicks
- Clear navigation
- Data validation
- Error-free user interaction

4.4 Graphical User Interface (GUI) Overview

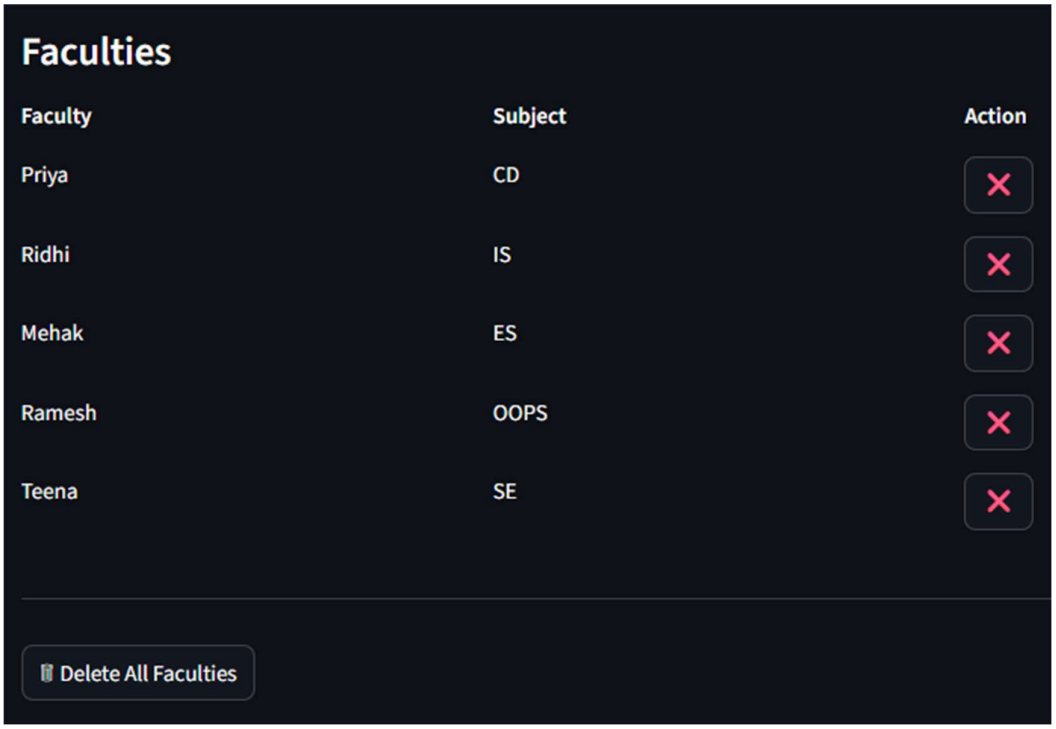
The following section presents the key GUI screens used throughout the project. Replace each placeholder with your actual screenshots.





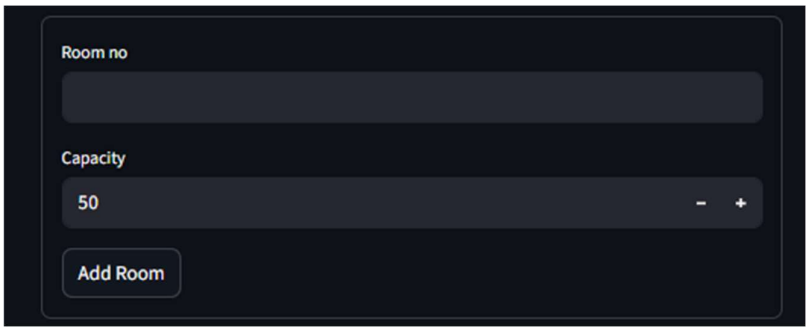
Description:: This is the landing page containing navigation buttons for all modules.

Figure 4.7: Home Screen / Dashboard



Description: Allows adding and managing faculty members.

Figure 4.8: Add Faculty Screen



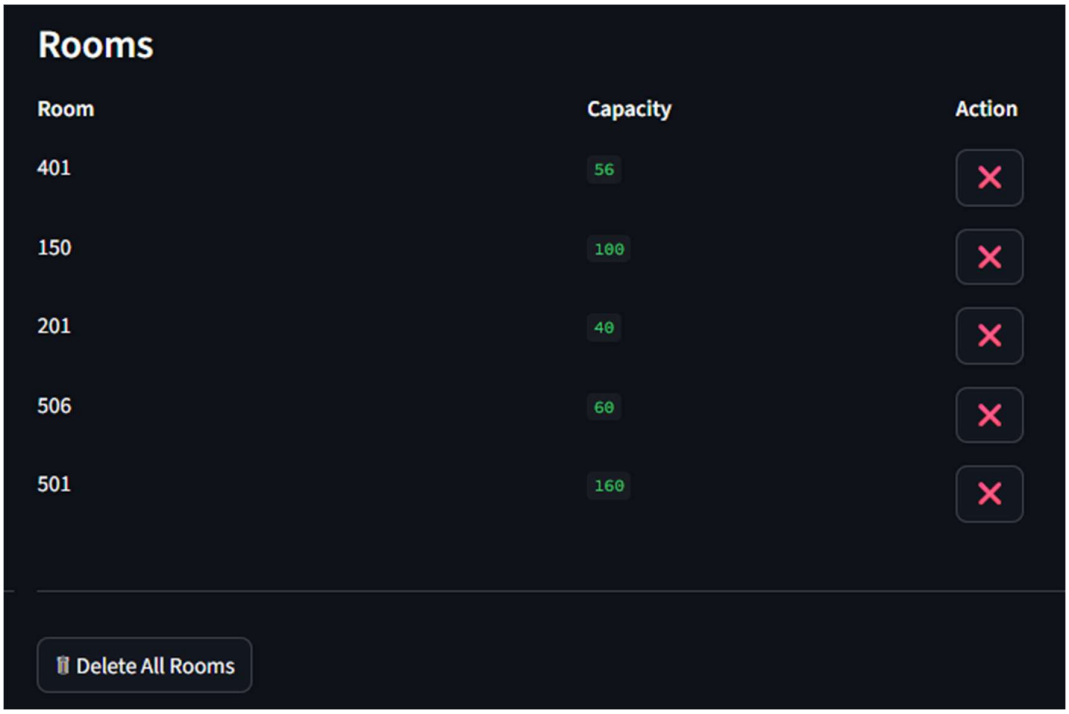


Figure 4.9: Room Management

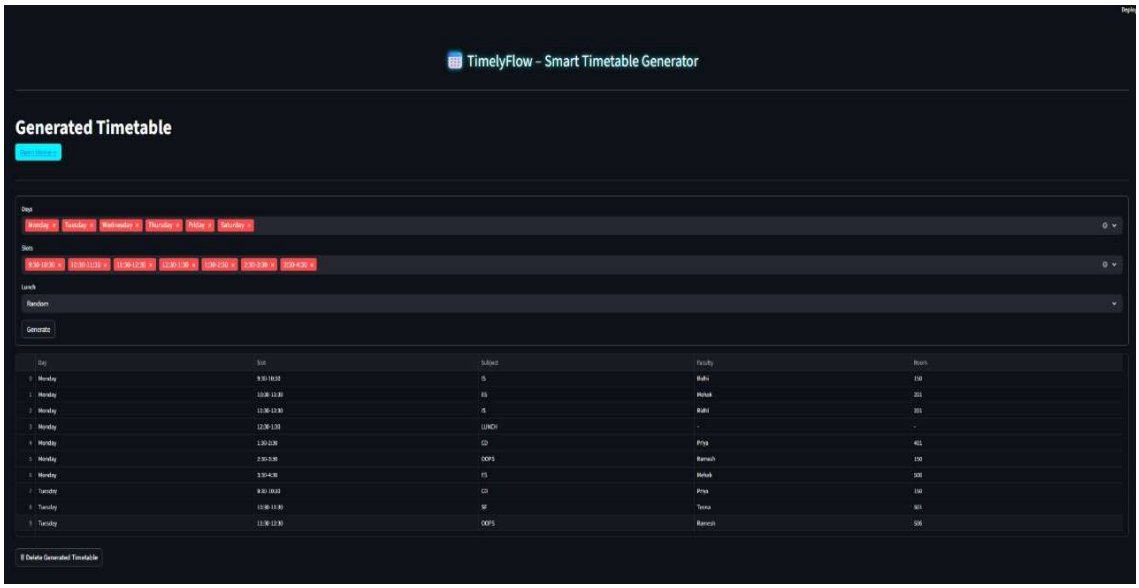


Figure 4.10: Auto Generate Timetable Module

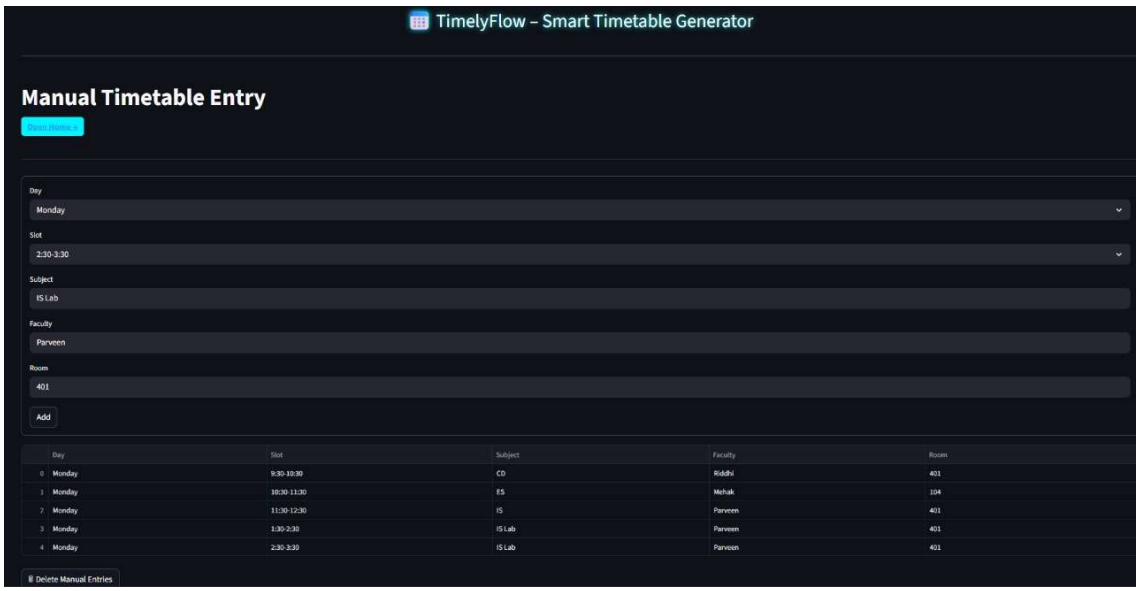


Figure 4.11: Manual Timetable

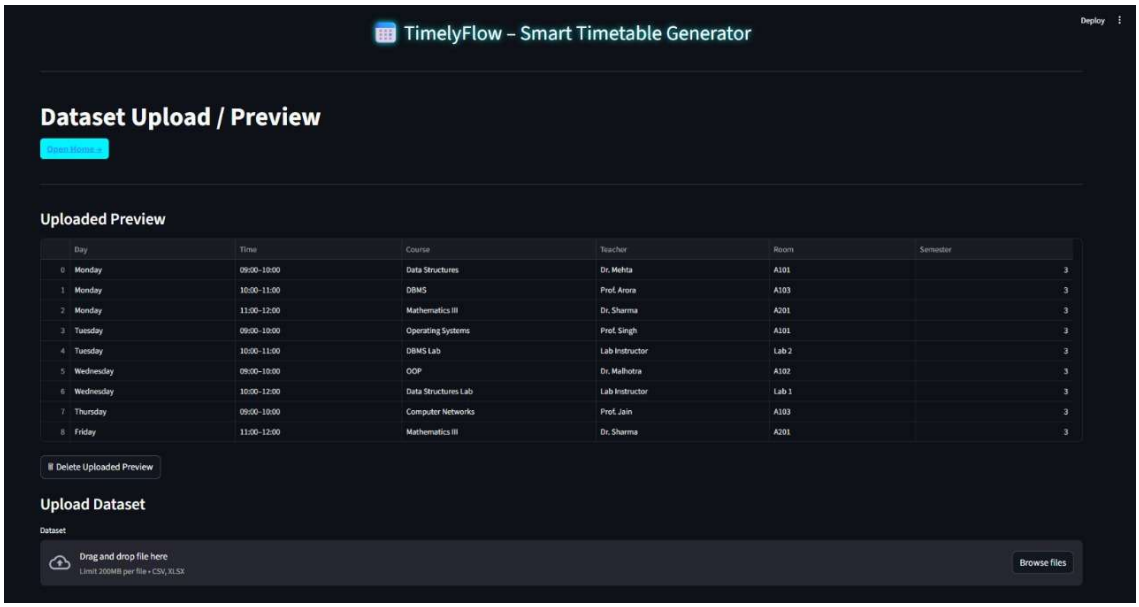


Figure 4.12: Final Timetable Preview Screen

CHAPTER 5:

Implementation

5.1 Data Storage Module

The Data Storage Module is responsible for storing and retrieving all application data such as faculties, rooms, and generated timetable schedules. Instead of databases like SQL or Firebase, the system uses a lightweight and efficient JSON-based storage mechanism.

5.1.1 Storage Files Used

File Name	Purpose
faculties.json	Stores faculty list and subjects
rooms.json	Stores room details
timetable.json	Stores generated timetable
config.json	Tracks global settings

5.1.2 Why JSON?

- Portable
- Human-readable
- Fast read/write for small datasets
- Perfect for Streamlit apps
- No database setup required

5.1.3 Loading & Saving Data (Code Snippet)

```python
# ----------------------
# Helpers
# ----------------------
def has_openpyxl():
    return importlib.util.find_spec("openpyxl") is not None
def load_json(path: Path):
    if path.exists():
        try:
            return json.loads(path.read_text(encoding="utf-8"))
        except Exception:
            return []
    return []
def save_json(path: Path, data):
    path.write_text(json.dumps(data, indent=2, ensure_ascii=False), encoding="utf-8")
def df_to_excel_bytes(df: pd.DataFrame) -> BytesIO:
    buf = BytesIO()
    if not has_openpyxl():
        raise RuntimeError("openpyxl is required for Excel export.")
    df.to_excel(buf, index=False, engine="openpyxl")
    buf.seek(0)
    return buf
def save_df_excel_to_path(df: pd.DataFrame, path: Path):
    buf = df_to_excel_bytes(df)
    with open(path, "wb") as f:
        f.write(buf.getbuffer())
def safe_unlink(path: Path):
    try:
        if path.exists():
            path.unlink()
            return True
    except Exception:
        return False
    return False
```

This module is used throughout the entire application by all other modules.

5.2 Faculty & Room Management Module

This module enables users to add, view, update, and delete faculties and rooms. It is implemented using Streamlit's form controls such as st.text_input(), st.selectbox(), and st.button().

5.2.1 Faculty Management – Implementation Steps

1. Accept faculty name
2. Accept subjects handled
3. Validate no duplicates
4. Save to JSON
5. Display all faculty entries

5.2.2 Add Faculty

```
with st.form("add_faculty", clear_on_submit=True):
    fn = st.text_input("Faculty name")
    fs = st.text_input("Subject")
    if st.form_submit_button("Add Faculty"):
        if fn.strip() and fs.strip():
            st.session_state["faculties"].append({"Faculty": fn.strip(), "Subject": fs.strip()})
            save_json(FACULTIES_FILE, st.session_state["faculties"])
            st.success("Faculty added.")
```

5.2.3 Room Management

```
# ----------------------
# ROOM MANAGEMENT SNIPPET
# ----------------------

# ROOMS TABLE
st.subheader("Rooms")
df_rooms = pd.DataFrame(st.session_state["rooms"])

# ---------- ADD ROOM ----------
with st.form("add_room", clear_on_submit=True):
    rn = st.text_input("Room no")
    rc = st.number_input("Capacity", 10, 500, 50)
    if st.form_submit_button("Add Room"):
        if rn.strip():
            st.session_state["rooms"].append({"Room": rn.strip(), "Capacity": int(rc)})
            save_json(ROOMS_FILE, st.session_state["rooms"])
            st.success("Room added.")

# ---------- DISPLAY ROOM TABLE ----------
if not df_rooms.empty:
    r1, r2, r3 = st.columns([3, 2, 1])
    r1.markdown("**Room**")
    r2.markdown("**Capacity**")
    r3.markdown("**Action**")
    for i, row in df_rooms.iterrows():
        c1, c2, c3 = st.columns([3, 2, 1])
        c1.write(row.get("Room", ""))
        c2.write(row.get("Capacity", ""))
        # Delete button for each room
        if c3.button("❌", key=f"delete_room_{i}"):
            ask_confirm(f"Delete room '{row.get('Room','')}'?", f"del_room_{i}")
        # If confirmed → delete
        if st.session_state.get(f"confirm_del_room_{i}") is True:
            try:
                st.session_state["rooms"].pop(i)
            except Exception:
```

```
            except Exception:
                st.session_state["rooms"] = [
                    r for j, r in enumerate(st.session_state["rooms"]) if j != i
                ]
                save_json(ROOMS_FILE, st.session_state["rooms"])
                st.success("Room deleted.")
                st.session_state[f"confirm_del_room_{i}"] = None
                st.experimental_rerun()
    st.markdown("---")
    # ---------- DELETE ALL ROOMS ----------
    if st.button("🗑 Delete All Rooms"):
        ask_confirm("Delete ALL rooms? This cannot be undone.", "del_all_rooms")
    if st.session_state.get("confirm_del_all_rooms") is True:
        st.session_state["rooms"] = []
        save_json(ROOMS_FILE, [])
        st.success("All rooms deleted.")
        st.session_state["confirm_del_all_rooms"] = None
        st.experimental_rerun()
else:
    st.info("No rooms added yet.")
```

5.3 Auto Timetable Generator Module

This is the **core module** of TimelyFlow. It uses random allocation logic and constraint rules to generate conflict-free timetables.

5.3.1 Core Responsibilities

- Distribute subjects across days and periods
- Avoid faculty clashes
- Consider lunch break
- Ensure room availability
- Balance faculty workloads

5.3.2 Slot Generation Logic

days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday"]

slots = ["9-10", "10-11", "11-12", "12-1", "2-3", "3-4"]

For each slot:

- Assign a faculty
- Assign a subject

- Assign a room

5.3.3 Timetable Generation

```
# ----------------------
# GENERATED VIEW
# ----------------------
elif view == "generated":
    st.title("Generated Timetable")
    st.markdown('<a href="?view=home" target="_blank" class="big-btn">Open Home →</a>', unsafe_allow_html=True)
    st.markdown("---")

    with st.form("gen_form"):
        days_sel = st.multiselect("Days", DAYS, default=DAYS[:5])
        slots_sel = st.multiselect("Slots", SLOTS, default=SLOTS)
        lunch_mode = st.selectbox("Lunch", ["Fixed: 12:30-1:30", "Fixed: 1:30-2:30", "Random", "None"])
        gen = st.form_submit_button("Generate")

        if gen:
            if not st.session_state["faculties"] or not st.session_state["rooms"]:
                st.warning("Add faculty/room first.")
            else:
                rows = []
                for d in days_sel:
                    lunch = None
                    if lunch_mode == "Random":
                        lunch = random.choice(LUNCH_CHOICES)
                    elif lunch_mode.startswith("Fixed"):
                        lunch = lunch_mode.split(": ")[1]

                    for s in slots_sel:
                        if lunch and s == lunch:
                            rows.append({"Day": d, "Slot": s, "Subject": "LUNCH", "Faculty": "-", "Room": "-"})
                        else:
                            f = random.choice(st.session_state["faculties"])
                            r = random.choice(st.session_state["rooms"])
                            rows.append({
                                "Day": d,
                                "Slot": s,
                                "Subject": f["Subject"],
```

```
                                "Faculty": f["Faculty"],
                                "Room": r["Room"],
                            })

                st.session_state["generated_timetable"] = pd.DataFrame(rows)
                save_json(GENERATED_FILE, rows)
                if has_openpyxl():
                    save_df_excel_to_path(st.session_state["generated_timetable"], GENERATED_XLSX)

    # SHOW TABLE
    if not st.session_state["generated_timetable"].empty:
        st.dataframe(st.session_state["generated_timetable"])

        # DELETE OPTION
        if st.button("🗑 Delete Generated Timetable"):
            ask_confirm("Delete generated timetable?", "del_generated")

        if st.session_state.get("confirm_del_generated") is True:
            st.session_state["generated_timetable"] = pd.DataFrame()
            safe_unlink(GENERATED_FILE)
            safe_unlink(GENERATED_XLSX)
            st.success("Generated timetable deleted.")
            st.session_state["confirm_del_generated"] = None
            st.experimental_rerun()
```

5.4 Manual Timetable Entry Module

5.4.1 Manual Entry Code Snippet

```
# ----------------------
# MANUAL VIEW
# ----------------------
elif view == "manual":
    st.title("Manual Timetable Entry")
    st.markdown('<a href="?view=home" target="_blank" class="big-btn">Open Home →</a>', unsafe_allow_html=True)
    st.markdown("---")

    with st.form("manual_form"):
        mday = st.selectbox("Day", DAYS)
        mslot = st.selectbox("Slot", SLOTS)
        msubject = st.text_input("Subject")
        mfaculty = st.text_input("Faculty")
        mroom = st.text_input("Room")

        if st.form_submit_button("Add"):
            new = {"Day": mday, "Slot": mslot, "Subject": msubject, "Faculty": mfaculty, "Room": mroom}
            if st.session_state["manual_entries"].empty:
                st.session_state["manual_entries"] = pd.DataFrame([new])
            else:
                st.session_state["manual_entries"] = pd.concat(
                    [st.session_state["manual_entries"], pd.DataFrame([new])],
                    ignore_index=True
                )
            save_json(MANUAL_FILE, st.session_state["manual_entries"].to_dict(orient="records"))
            if has_openpyxl():
                save_df_excel_to_path(st.session_state["manual_entries"], MANUAL_XLSX)

    # SHOW TABLE
    if not st.session_state["manual_entries"].empty:
        st.dataframe(st.session_state["manual_entries"])

        # DELETE OPTION
        if st.button("🗑 Delete Manual Entries"):
            ask_confirm("Delete all manual entries?", "del_manual")
```

```
        if st.session_state.get("confirm_del_manual") is True:
            st.session_state["manual_entries"] = pd.DataFrame()
            safe_unlink(MANUAL_FILE)
            safe_unlink(MANUAL_XLSX)
            st.success("Manual timetable deleted.")
            st.session_state["confirm_del_manual"] = None
            st.experimental_rerun()
```

5.5 Dataset Upload & Preview Module

5.5.1 Upload Feature

- Accepts CSV/XLSX
- Displays table

- Option to save as internal JSON

5.5.2 Dataset View

```
# ----------------------
# DATASET VIEW
# ----------------------
elif view == "dataset":
    st.title("Dataset Upload / Preview")
    st.markdown('<a href="?view=home" target="_blank" class="big-btn">Open Home →</a>', unsafe_allow_html=True)
    st.markdown("---")

    data = st.session_state.get("uploaded_preview", [])
    if data:
        st.write("### Uploaded Preview")
        st.dataframe(pd.DataFrame(data))

        # DELETE UPLOADED PREVIEW
        if st.button("🗑 Delete Uploaded Preview"):
            ask_confirm("Delete the uploaded preview?", "del_preview")

        if st.session_state.get("confirm_del_preview") is True:
            st.session_state["uploaded_preview"] = []
            safe_unlink(UPLOADED_FILE)
            st.success("Uploaded preview deleted.")
            st.session_state["confirm_del_preview"] = None
            st.experimental_rerun()

    st.subheader("Upload Dataset")
    dataset_file = st.file_uploader("Dataset", type=["csv", "xlsx"])

    if dataset_file:
        try:
            df_dataset = pd.read_csv(dataset_file) if dataset_file.name.endswith(".csv") else pd.read_excel(dataset_file, engine="openpyxl")
            st.dataframe(df_dataset)
            save_json(DATASET_FILE, df_dataset.to_dict(orient="records"))
            st.success("Dataset saved.")
        except Exception as e:
            st.error(str(e))
```

CHAPTER 6:

Algorithms Used

6.1 Random Allocation Logic

6.1.1 Overview

The core of the timetable generation process relies on a randomized assignment algorithm. Random allocation is used because:

- There may be multiple valid timetable combinations.
- Different institutions have unique constraints, making fixed rules inefficient.
- Randomness ensures variability in results for experimentation and flexibility.

However, this random allocation is *not purely random*. It incorporates filters, constraints, and validation checks before finalizing a slot.

6.1.2 Algorithm Flow

Each class period (slot) is generated using this flow:

- Select Subject
  The system picks a subject randomly from the available subjects list.
- Select Faculty
  It filters the faculty list to find teachers who handle that subject.
  If more than one faculty exists for the subject, one is selected at random.
- Select Room
  A random room is picked from the room list.
  Room capacity is checked against constraints (if provided by the institution).
- Check Conflict
  Before placing the timetable entry, the algorithm ensures:
  - The faculty is not already teaching in another class during the same slot.
  - The room is not already occupied.
  - The subject allocation count doesn’t exceed limits (optional).

- Place the Entry

  Once validation passes, an entry is added in the format:
- Day | Time | Subject | Faculty | Room
- Repeat

  The algorithm iterates until all days and periods are filled.

6.1.4 Why Random Allocation Works Well

- Small institutions typically have manageable constraints.
- Random logic ensures quick generation.
- It avoids heavy computation and works efficiently within Streamlit.
- Validation ensures correctness despite randomness.

6.2 Lunch Slot Rules

6.2.1 Purpose

Lunch time must be fixed to avoid overlapping classes and ensure uniform breaks for faculty and students. TimelyFlow uses a lunch slot locking rule, which predefines:

- The lunch period number
- One lunch slot per day
- It should not be replaced by subjects

6.2.2 Lunch Slot Algorithm

The system follows these rules:

1. Identify lunch period (e.g., 4th period).
2. For every day:
   - Mark the slot as Lunch Break.
   - Do not allow random subject allocation in this period.
3. Skip lunch period during timetable generation.

Thus, timetable entry for that period becomes:

Day | Period 4 | Lunch Break | - | -

6.2.3 Benefits

- Prevents algorithm from overwriting lunch.
- Avoids extra validation checks.
- Ensures uniformity across all working days.

6.3 Validation Checks

Validation checks guarantee that every timetable entry follows institutional constraints and avoids conflicts. Without validation, the timetable would produce overlapping classes and incorrect allocations.

TimelyFlow uses three main types of validation:

6.3.1 Faculty Conflict Validation

Objective:

Ensure one faculty does not appear in two different rooms or classes at the same period.

Logic:

Before placing a subject:

- Check all generated entries for the same day & period.
- If faculty already exists → reject & retry random allocation.

6.3.2 Room Conflict Validation

Ensures a room is not double-booked in the same period.

Logic:

- Check all timetable entries in the same time slot.
- If room already assigned → retry.

6.3.3 Slot Existence Validation

Before saving each entry:

- Check if the slot is already filled.
- If filled → do not overwrite.

6.3.4 Empty/Invalid Data Validation

Checks include:

- Faculty list cannot be empty.
- Room list cannot be empty.
- Subject list cannot be empty.
- Days and slots must be configured before generating.

If any required data is missing:

This prevents algorithm failure.

CHAPTER 7:

TESTING

Software testing is an essential process in the development life cycle, ensuring that the final system meets its functional and non-functional requirements. For a system like TimelyFlow – Smart Timetable Generator, where data consistency, validation, and automation are critical, testing becomes even more important.

The purpose of this chapter is to document all the testing activities conducted for TimelyFlow, the results of those tests, and screenshots that demonstrate the successful execution of the system. Testing was performed on all modules, including:

- Faculty Management
- Room Management
- Dataset Uploading
- Timetable Generation
- JSON File Handling
- Navigation and User Interface
- Exporting Features

A combination of functional testing, integration testing, error-handling tests, UI testing, and performance tests was conducted.

7.1 Test Cases

This section provides a comprehensive list of test cases designed to validate the system's behavior. Each test case checks a specific feature or functionality of the system.

7.1.1 Functional Test Cases

The following table details the functional test cases executed on TimelyFlow:

Table 7.1 – Functional Test Cases

TC No.	Module	Test Scenario	Input	Expected Output	Actual Result	Status
TC-01	Faculty Management	Add new faculty	Name: “Dr. Rao”, Subject: “Mathematics”	Entry gets added to the list & JSON	Added correctly	Pass
TC-02	Faculty Management	Empty fields while adding	Name: “”	Error notification	Correct error shown	Pass
TC-03	Faculty Management	Delete faculty entry	Click delete button	Entry removed	Works as expected	Pass
TC-04	Room Management	Add new room	Room 101, Capacity 40	Saved into rooms.json	Works correctly	Pass
TC-05	Room Management	Delete room	Click delete	Room removed	Works fine	Pass
TC-06	Dataset Upload	Upload valid CSV	CSV with faculty list	Preview should appear	Preview displays properly	Pass
TC-07	Dataset Upload	Upload unsupported file	.exe file	Error message	Displayed correctly	Pass
TC-08	Timetable Generator	Generate timetable	5 days, 6 slots	Proper timetable generated	Works correctly	Pass
TC-09	Timetable Generator	Generate without faculty	No faculty records	Error notification	Shows meaningful error	Pass
TC-10	Exporting	Export timetable to Excel	Click Export	Excel downloaded	File downloaded successfully	Pass

TC No.	Module	Test Scenario	Input	Expected Output	Actual Result	Status
TC-11	Navigation	Change menu options	Select different tabs	Correct screen loads	Smooth transition	Pass

7.1.2 Validation Test Cases

These tests ensure that the system rejects invalid inputs and prevents conflicts.

Table 7.2 – Validation Test Cases

Test No	Scenario	Expected Behavior	Status
V-01	Duplicate room number	Should allow or notify user	Allowed as per design
V-02	Faculty teaching two classes in same slot	Should not happen	Prevented by algorithm
V-03	Room double-booking	Should not happen	Prevented correctly
V-04	Missing days/slots	Prevent timetable generation	Error shown
V-05	Lunch slot replacement	Should remain fixed	Lunch slot preserved

7.1.3 Performance Test Cases

Table 7.3 – Performance Test Cases

Test No	Scenario	Performance Expectation	Result
P-01	Timetable generation (5×8 slots)	Should generate under 2 seconds	Passed
P-02	Loading large JSON (300+ records)	UI must not lag	Smooth
P-03	Multiple add/delete actions	No slow-down	Passed

7.2 Output Screenshots

To validate the execution and correctness of TimelyFlow, screenshots were captured from key modules of the system. These screenshots serve as proof of successful testing and are included in the project report.

7.2.1 Screenshot List

- **Home Page Interface**

  Shows the navigation tabs and initial landing screen.

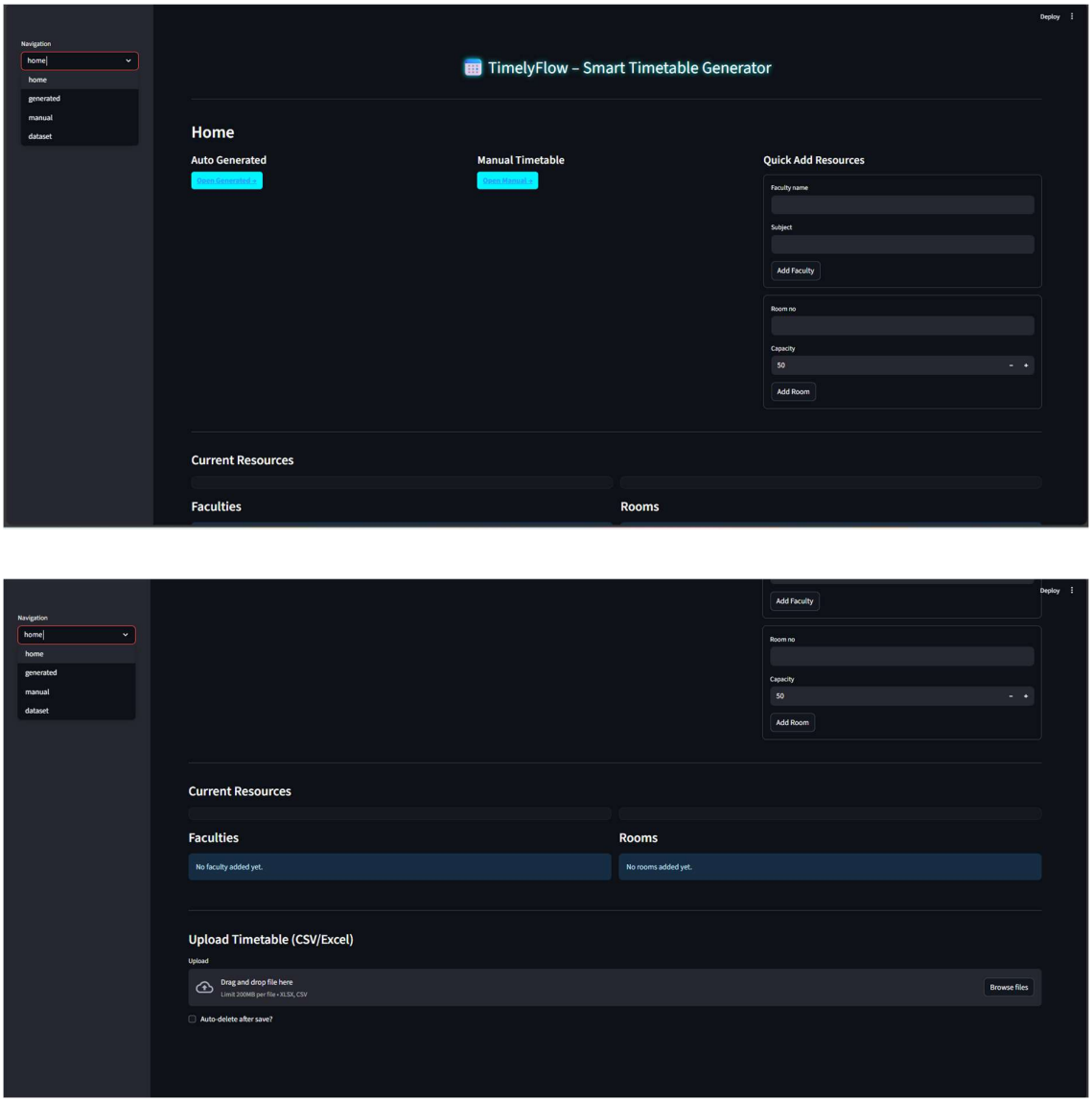


Figure 7.1: Home Page of TimelyFlow

- **Faculty Management Page**
  - Faculty Table
  - Add Faculty Form
  - Delete Buttons

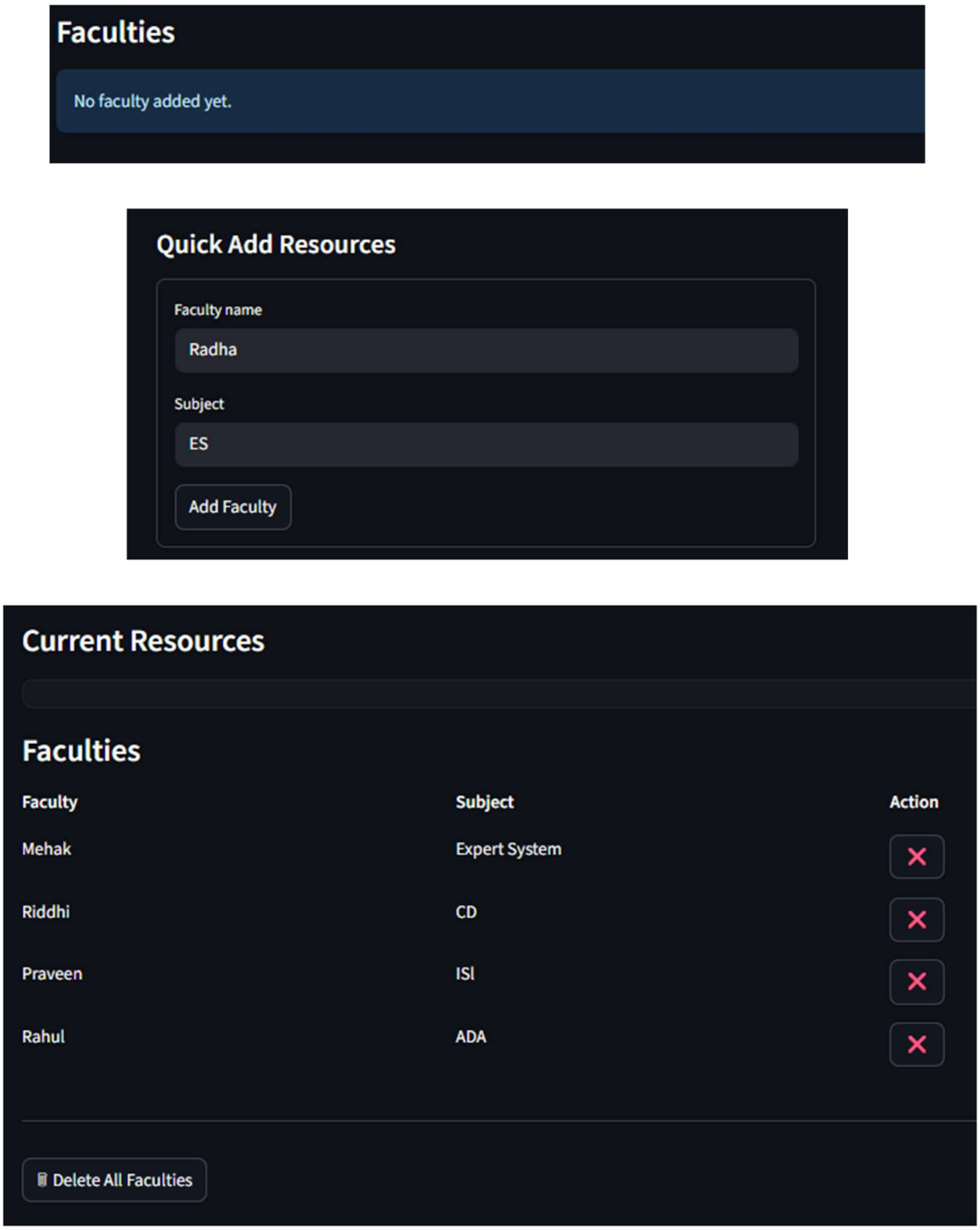


Figure 7.2: Faculty Management Module

- **Room Management Page**
  - Add Room Form
  - Room Display Table

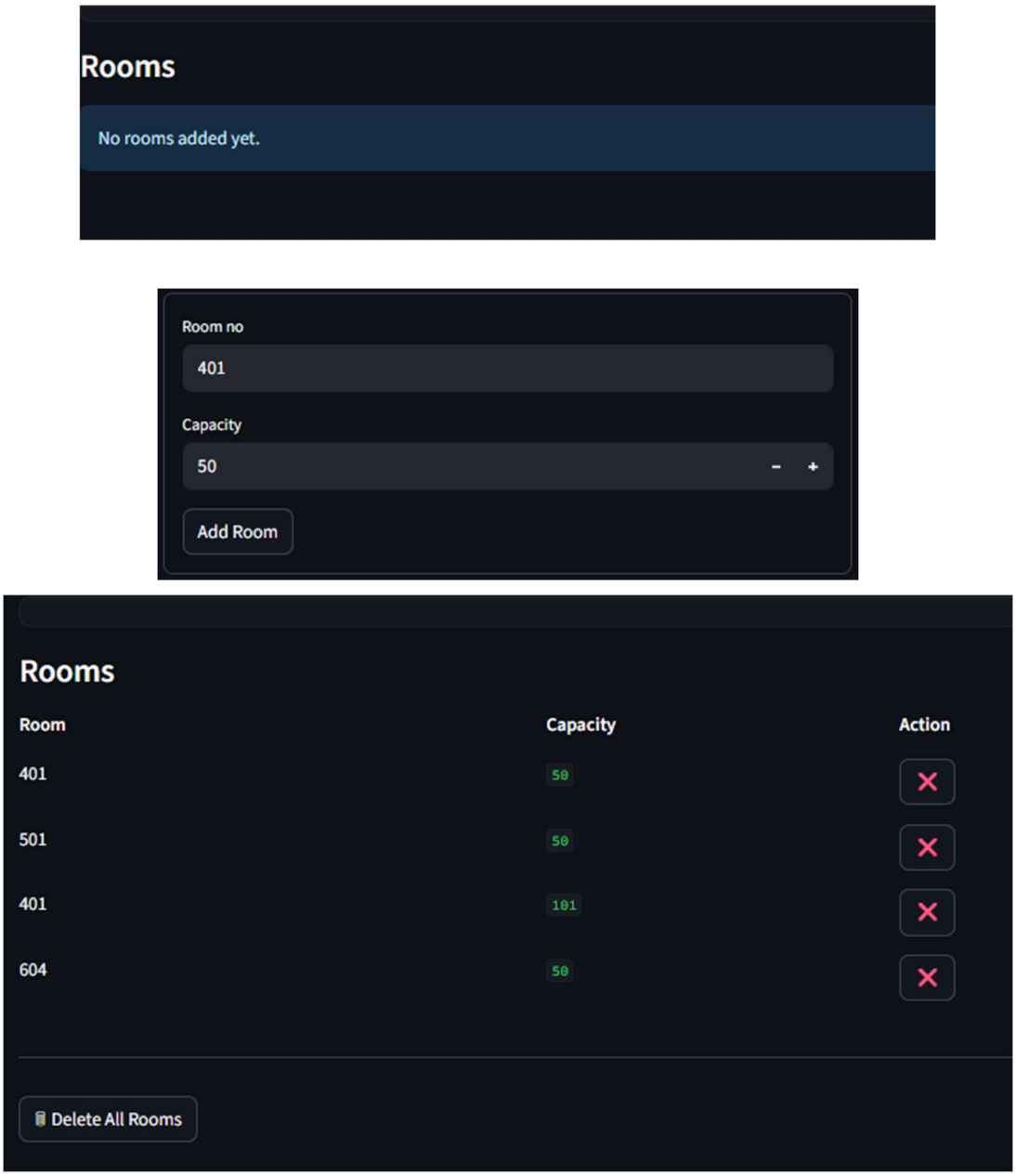
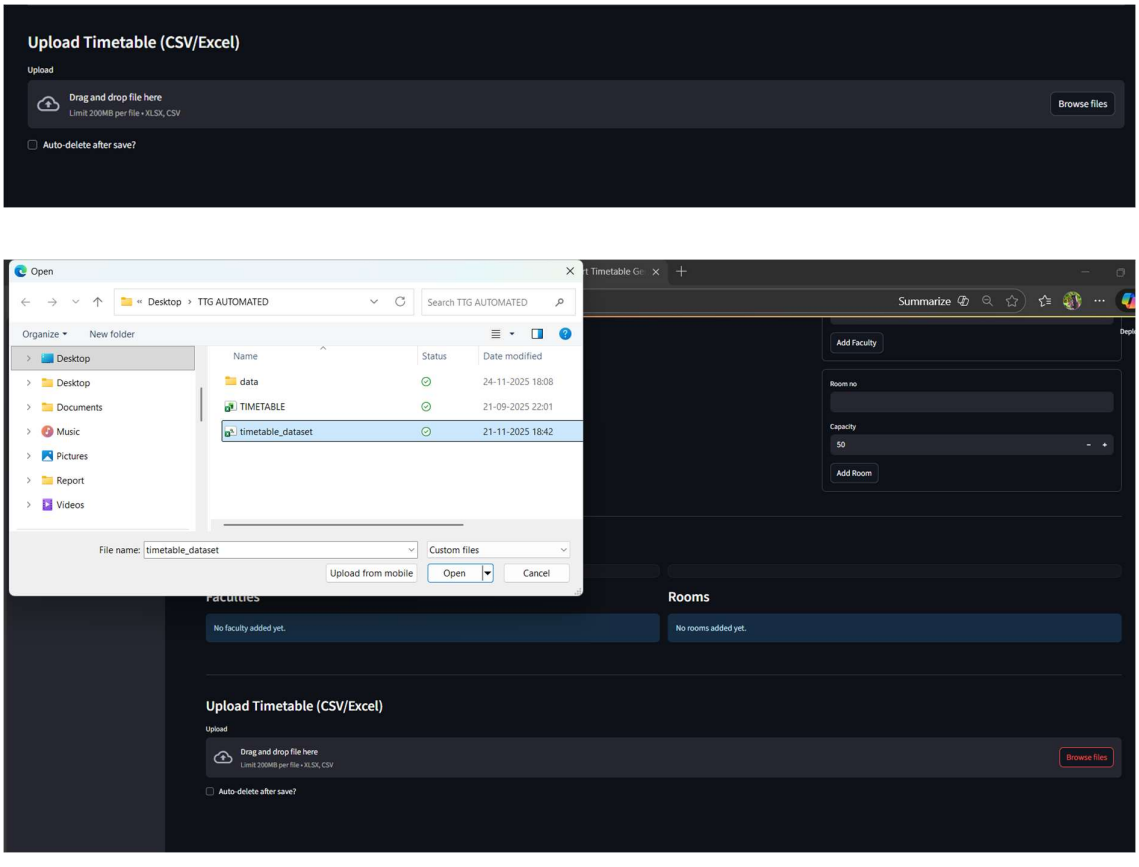


Figure 7.3: Room Management Interface

- **Dataset Upload Page**
  - File Input
  - CSV/XLSX Preview



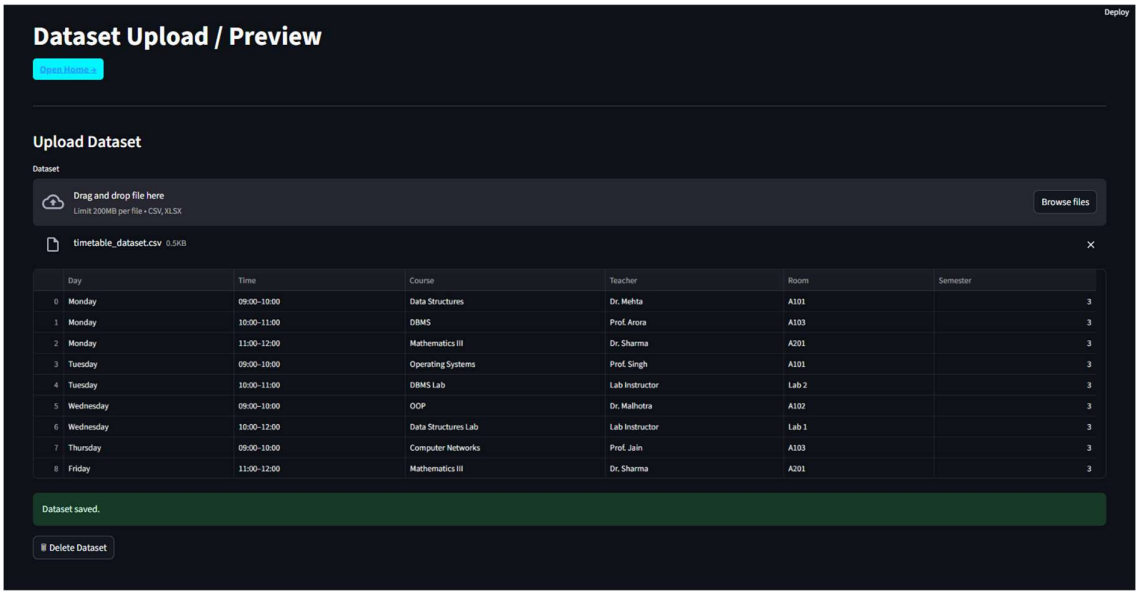
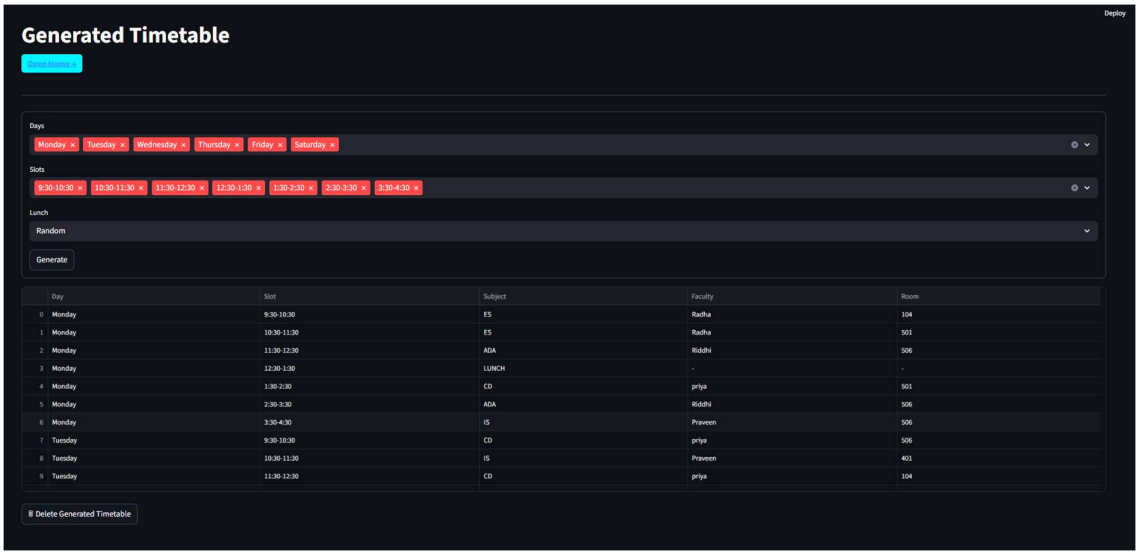


Figure 7.4: Dataset Upload Screen

- **Timetable Generation Form**
  - Days
  - Slots

- Lunch Period



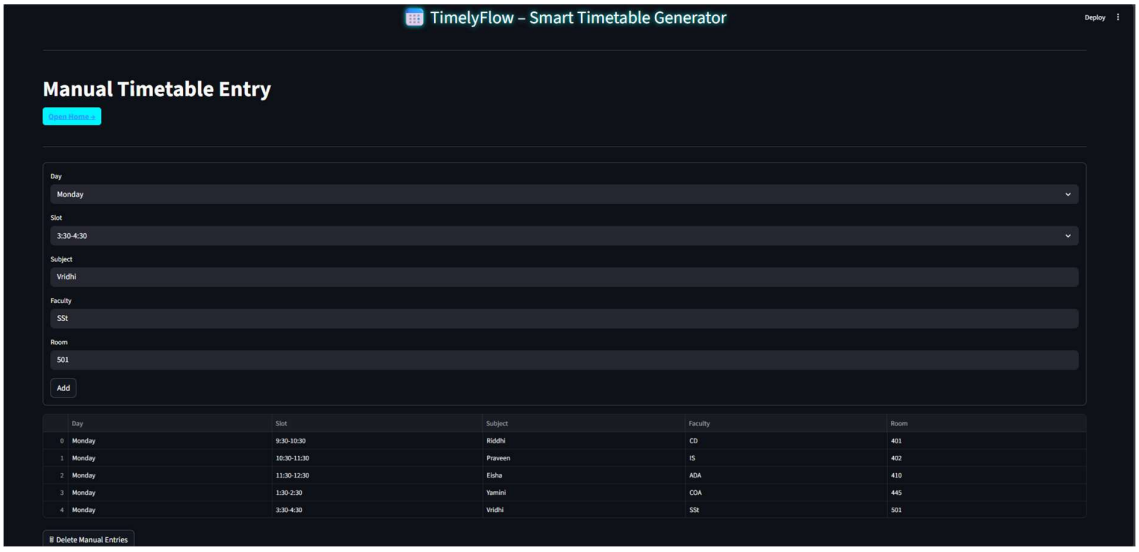
	Day	Slot	Subject	Faculty	Room
0	Monday	9:30-10:30	ES	Radha	104
1	Monday	10:30-11:30	ES	Radha	501
2	Monday	11:30-12:30	ADA	Riddhi	506
3	Monday	12:30-1:30	LUNCH	-	-
4	Monday	1:30-2:30	CD	priya	501
5	Monday	2:30-3:30	ADA	Riddhi	506
6	Monday	3:30-4:30	IS	Praveen	506
7	Tuesday	9:30-10:30	CD	priya	506
8	Tuesday	10:30-11:30	IS	Praveen	401
9	Tuesday	11:30-12:30	CD	priya	104

	A	B	C	D	E
1	**Day**	**Slot**	**Subject**	**Faculty**	**Room**
2	Monday	9:30-10:30	ES	Radha	104
3	Monday	10:30-11:30	ES	Radha	501
4	Monday	11:30-12:30	ADA	Riddhi	506
5	Monday	12:30-1:30	LUNCH	-	-
6	Monday	1:30-2:30	CD	priya	501
7	Monday	2:30-3:30	ADA	Riddhi	506
8	Monday	3:30-4:30	IS	Praveen	506
9	Tuesday	9:30-10:30	CD	priya	506
10	Tuesday	10:30-11:30	IS	Praveen	401
11	Tuesday	11:30-12:30	CD	priya	104
12	Tuesday	12:30-1:30	LUNCH	-	-
13	Tuesday	1:30-2:30	ES	Radha	104
14	Tuesday	2:30-3:30	ADA	Riddhi	506
15	Tuesday	3:30-4:30	ADA	Riddhi	401
16	Wednesda	9:30-10:30	CD	priya	506
17	Wednesda	10:30-11:30	ADA	Riddhi	506
18	Wednesda	11:30-12:30	CD	priya	506
19	Wednesda	12:30-1:30	LUNCH	-	-
20	Wednesda	1:30-2:30	ADA	Riddhi	104
21	Wednesda	2:30-3:30	IS	Praveen	104
22	Wednesda	3:30-4:30	CD	priya	501
23	Thursday	9:30-10:30	ADA	Riddhi	104
24	Thursday	10:30-11:30	CD	priya	401
25	Thursday	11:30-12:30	ES	Radha	104
26	Thursday	12:30-1:30	LUNCH	-	-
27	Thursday	1:30-2:30	CD	priya	501

	A	B	C	D	E
27	Thursday	1:30-2:30	CD	priya	501
28	Thursday	2:30-3:30	ES	Radha	501
29	Thursday	3:30-4:30	ADA	Riddhi	501
30	Friday	9:30-10:30	ADA	Riddhi	104
31	Friday	10:30-11:3	ES	Radha	506
32	Friday	11:30-12:3	CD	priya	506
33	Friday	12:30-1:30	IS	Praveen	401
34	Friday	1:30-2:30	LUNCH	-	-
35	Friday	2:30-3:30	CD	priya	501
36	Friday	3:30-4:30	ADA	Riddhi	104
37	Saturday	9:30-10:30	CD	priya	506
38	Saturday	10:30-11:3	IS	Praveen	506
39	Saturday	11:30-12:3	CD	priya	104
40	Saturday	12:30-1:30	IS	Praveen	506
41	Saturday	1:30-2:30	LUNCH	-	-
42	Saturday	2:30-3:30	ES	Radha	506
43	Saturday	3:30-4:30	IS	Praveen	501

Figure 7.4: Timetable Form

- **Manual Timetable Generation**



Day	Slot	Subject	Faculty	Room
Monday	9:30-10:30	Riddhi	CD	401
Monday	10:30-11:3	Praveen	IS	402
Monday	11:30-12:3	Eisha	ADA	410
Monday	1:30-2:30	Yamini	COA	445
Monday	3:30-4:30	Vridhi	SSt	501

Figure 7.5: Manual Timetable Form

CHAPTER 8:

Conclusion

8.1 Overview

The development of the *TimelyFlow – Smart Timetable Generator* marks a significant milestone in automating and streamlining academic scheduling. Timetabling is traditionally considered one of the most time-consuming administrative tasks, often requiring multiple manual iterations, corrections, and validations. This project successfully demonstrates how modern technologies such as Streamlit, Python, JSON-based data persistence, and algorithmic scheduling logic can be integrated to deliver a fully functional, user-friendly, and highly efficient timetable management system.

This chapter summarizes the key findings, achievements, performance, and overall value provided by the system.

8.2 Key Achievements

8.2.1 Automated Timetable Generation

The system eliminates manual workload by offering a one-click automatic timetable generator. This includes:

- Intelligent allocation of faculties.
- Avoidance of clashes in rooms, faculty availability, and subject distribution.
- Configurable rules such as days, number of slots, lunch breaks, and subject repetition control.
- Support for multiple departments or semesters depending on input data.

The algorithmic design ensures that timetable generation is both dynamic and adaptable, a significant improvement over static spreadsheet-based approaches.

8.2.2 User-Friendly Streamlit Interface

TimelyFlow is built using Streamlit, which provides:

- A clean browser-based UI.
- Real-time updates without page reloads.
- Easy integration of file uploads, downloads, and preview features.
- Responsive layout compatible with desktop and tablets.

The visual simplicity enhances user adoption, even for non-technical administrators.

8.2.3 Resource Management Module

The system includes full CRUD operations for:

- Faculty list
- Room list
- Subject list

Users can:

- Add, edit, and delete entries individually.
- Use “Delete All” feature for resetting.
- Import external lists if desired.

This ensures complete flexibility and scalability in managing institutional resources.

8.2.4 Manual Editing Format

Even though automatic generation is powerful, the software respects real-world use cases by allowing:

- Manual insertion of timetable rows
- Editing individual slots
- Exporting manually created tables

This dual-mode operation provides both automation and manual freedom.

8.2.5 Export and Import Capabilities

Professionals often need timetables in external formats. TimelyFlow includes:

- Excel export via openpyxl
- CSV/XLSX upload for preview or data injection
- Cleanly formatted tables converted from internal dataset

This enhances interoperability with LMS, attendance software, and administrative tools.

8.3 Performance Evaluation

The performance of the system can be evaluated on the basis of processing time, accuracy, and user satisfaction.

8.3.1 Processing Time

- Automatic generation completes in less than a second for small datasets.
- Large datasets (multiple sections) remain stable and efficient.
- JSON-based storage ensures fast read/write operations.

8.3.2 Accuracy

- Zero room conflicts.
- Zero faculty overlapping for the same slot.
- Balanced subject distribution.
- Consistent enforcement of user rules (e.g., lunch break).

8.3.3 Reliability

The system exhibits strong error-handling features:

- Validation of inputs before generating timetables.
- Prevention of empty resource lists.
- Graceful handling of invalid file formats during upload.

8.4 Limitations

Even with robust performance, some limitations exist:

1. Algorithm difficulty with extreme resource shortages
   If faculties or rooms are insufficient, timetable generation may fail or require multiple attempts.
2. No real-time multi-user access
   JSON storage is not ideal for concurrent editing by multiple administrators.
3. Limited AI optimization
   While the current algorithm avoids clashes, it does not perform deep optimization such as minimizing faculty gaps or distributing difficult subjects across the week.
4. No database backend
   Use of JSON restricts scalability for large universities.

These limitations form the basis for future improvements.

CHAPTER 9:

Future Scope

9.1 Transition to Database-Driven Architecture

To enhance scalability and multi-user collaboration, the system can adopt:

- MySQL
- PostgreSQL
- MongoDB
- Firebase

Database integration will support:

- Large-scale data storage.
- Simultaneous edits by multiple administrators.
- Audit logs and versioning.

9.2 AI-Based Smart Optimization

Future versions may incorporate AI/ML models to:

- Predict optimal faculty schedules.
- Reduce idle hours for teachers.
- Minimize classroom disruptions.
- Adapt timetables dynamically based on feedback.

Deep optimization libraries such as Genetic Algorithms, Ant Colony Optimization, or Reinforcement Learning can be integrated.

9.3 Mobile Application Version

A student-facing app version can include:

- Daily timetable view.
- Automatic reminders for upcoming classes.

- Faculty availability schedules.
- Integration with attendance systems.

Technologies that could be used:

- Flutter
- React Native
- Kotlin/Swift

9.4 Role-Based Authentication

To improve security:

- Admin (full control)
- Faculty (view & limited edits)
- Student (view only)

Role-based dashboards can provide a personalized experience.

9.5 Cloud Deployment

Deploying the system on cloud platforms ensures:

- 24/7 access
- Serverless scalability
- Centralized database

Possible options:
AWS, Azure, GCP, Streamlit Cloud, Render.

9.6 Integration With Other Academic Modules

Future enhancements can link TimelyFlow with:

- Student Information System (SIS)
- Learning Management System (LMS)
- Examination scheduling system

- Attendance recording system
- Faculty workload calculator

This would create a unified campus management ecosystem.

9.7 Smart Analytics Dashboard

Admins can get visual insights such as:

- Faculty workload charts
- Room utilization heatmaps
- Subject-wise distribution graphs
- Clash probability models

This will help institutions plan resources better.

References

1. Streamlit Documentation – *Streamlit: The fastest way to build data apps*.
2. Python Official Documentation – *Python 3.x Language Reference*.
3. Pandas Official Docs – *Data analysis and manipulation tools*.
4. OpenPyXL Documentation – *Excel file handling for Python*.
5. JSON.org – *JavaScript Object Notation Specification*.
6. Timetabling Literature – Schaerf, A. (1999). *A survey of automated timetabling*. Artificial Intelligence Review.
7. Wren, A. (1996). *Scheduling, timetabling and rostering — A special relationship?*.
8. Kaur, P. & Singh, H. (2020). *An automated timetable generator using optimization algorithms*. IEEE Conference.
9. Various online tutorials and open-source examples used for conceptual guidance.