RESEARCH DOCUMENT

Smart Scheduler and Reminder System

Author: Changyu Jia

Student ID: C00292876

Programme: SETU — BSc Software Development (Year 4)

Supervisor: Dr Jamal Tauseef

Date: 1/10/2025

Abstract

This Research document provides a detailed description of the development plan for the "Smart Scheduler and Reminders System". This system is a professional web-based application designed to address the common scheduling and communication challenges faced by educational institutions and community groups. Its goal is to replace the current reliance on general tools such as Google Calendar and Notion, which lack specialized support for academic workflows. The system aims to become an intelligent interactive center, significantly reducing the occurrence of missed appointments, simplifying the meeting coordination process, and providing valuable, data-based insights into user engagement and punctuality.

This platform is specifically tailored for the academic environment, and its core functions include providing secure, role-based access control for three main user types (administrators, teachers, and students). A comprehensive schedule and event management module supports the automatic loading of semester-based timetables from file data sets such as JSON, while allowing authorized changes. A key innovation is the two-way calendar visibility system, which enhances transparency by allowing users to view each other's schedule commitments, thus overcoming the limitations of traditional one-way calendar sharing. The system provides multi-channel automatic reminders and notifications for any updates through email and in-app alerts. Additionally, it integrates personal to-do list management for students and includes activity participation tracking and reporting features, capable of generating indicators such as "Completed Activities" and "Missed Activities". By consolidating these functions, this platform aims to minimize communication gaps and management burdens, creating a more efficient and closely-connected academic ecosystem.

CONTENTS

1. Introduction

In the digital age, efficient time management and information synchronization are crucial for the success of organizations and individuals. The research shows that 94% of people believe that better time management can enhance productivity, 91% think it can reduce work pressure[1]. However, information delays and misunderstandings are the primary reasons for low team efficiency. Universities usually rely on various tools (such as WhatsApp, email lists, general calendar applications) to manage activities and courses, which easily lead to outdated and missed information. This research project aims to solve these problems through a unified web application, thereby improving collaboration efficiency and activity participation.

2. Comparison and Reflection

2.1 Existing Solutions and General Comparison

In this field, there are already many mature products, each with their own strengths and target audience:

- **Google Calendar** [2] : It is the industry standard in the field of personal and shared calendars, with superior reliability and cross-platform availability. However, its utility for specific academic or organizational needs is limited. It lacks granular role-based permissions.
- **Notion** [3] : It represents a modern way of managing schedules, featuring functions such as a to-do list and AI. However, its functions are relatively basic compared to Google Calendar.

Comparison	**Google Calendar**	**Notion Calendar**	**Smart Scheduler (Proposed)**
Target User	Wide range of individual users and enterprises	Project team, individuals	University
Authority	Lacking role	Strong flexibility	Custom role-based

	permission control		access (Admin, Teacher, Student)
Reminding	Basic reminding	Manual configuration	Multi-channel reminder (Email + In-site)
Application Scenarios	General Scenario	Project Management	Academic and community scheduling
Usability	Extremely high, with an intuitive interface	Lower, requiring lower learning costs	High, optimized for specific context

The design of this project incorporates insights into these existing solutions: the front-end interface emphasizes simplicity and operational efficiency to ensure seamless interaction. The notification and reminder mechanism serves as a core functionality, adopting a multi-channel strategy for enhanced effectiveness.

The design differentiation of this project lies in its targeted focus on specific scenarios. This contextual positioning enables an in-depth exploration of user roles and permission models. Furthermore, the integration of calendar and to-do list functionalities is proposed to generate personalized schedules for users[5], thereby addressing common limitations in general-purpose tools.

2.2 Functional Comparison and Design Reflection

To highlight the commercial competitiveness of this system, on the basis of the original comparison, a functional comparison is conducted between the new functions and the mainstream products:

Functional Characteristics	**Google Calendar**	**Notion Calendar**	**Smart Scheduler (Proposed)**
Cross-role synchronization	Only basic sharing is supported, with	The board needs to be configured	**Bidirectional searchable calendar**:

	no permission subdivision.	manually, and its real-time performance is poor.	Students can check the availability of teachers, and teachers can batch-block time slots for specific courses, eliminating the need for repeated email confirmations.
Multi-mode Notification	Support single-time reminder for basic emails/pop-up notifications	Only supports pop-up notifications	**User-defined channels**: Support combination of email and system notifications; Support repeated reminders with intervals of "1 day/1 hour/15 minutes"
Participation tracking and reporting	No absence record function	Manual marking is required to complete the status.	**Attendance marking**: There are "attendance participation/absence" records; and it can generate personal attendance rate and task completion reports in one click.
Suggestion	Not have	Not have	Recommended tasks

Engine			and events based on popular events or functions

Simply replicating the general calendar function cannot create competitiveness. It is necessary to address the pain points of general tools in cross-role coordination. Although Google Calendar is widely used, it cannot enable mutual viewing between teachers and students; Notion is flexible but requires a lot of manual configurations. This system reduces the burden on user operations while providing decision support.

3. Technology and System Architecture

3.1 Technology Selection and Justification

- **Backend: Python and Django**

  The server-side logic is based on Python and is built using the Django web framework. The choice of Python was made due to its excellent readability, extensive library ecosystem, and rapid development capabilities. Django allows to configure and run different preprocessing tasks efficiently. Its built-in object-relational mapper handles database operations, and it authentication system provides a secure foundation for managing user roles and access [5].

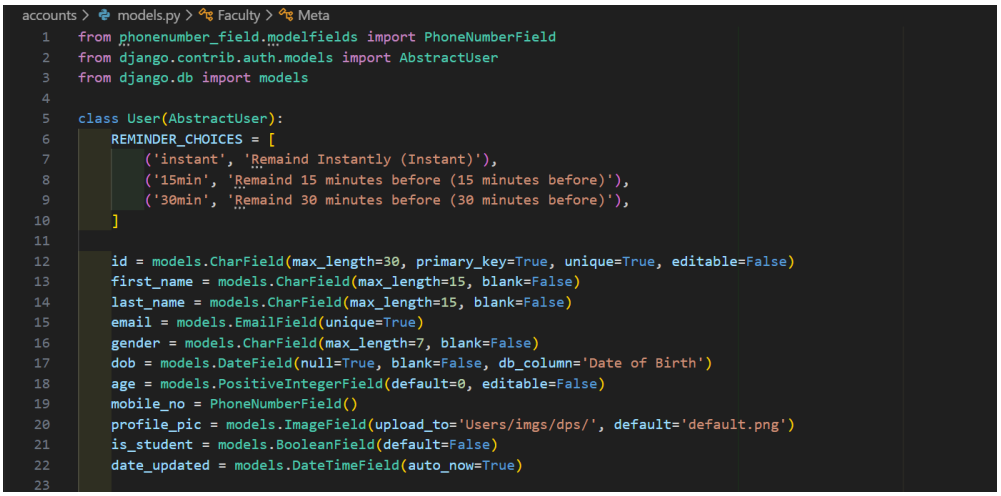


Figure 3.1.1 models.py file.

It epitomizes Django's Model layer by integrating with the authentication system and leveraging the ORM for database operations. (Figure 3.1.1)

```python
campus > models.py > ...
from accounts.models import Student, Faculty
from django.db import models

class BookedUnit(models.Model):
    """ These are records of units assigned to a lecturer each semester. """
    id = models.CharField(max_length=30, primary_key=True, unique=True, editable=False)
    lecturer = models.ForeignKey(Faculty, on_delete=models.CASCADE)
    students_course = models.CharField(max_length=50, blank=False)
    course_name = models.CharField(max_length=80, blank=False)
    year_of_study = models.CharField(max_length=10, blank=False)
    semester = models.CharField(max_length=1, blank=False)
    booking_date = models.DateTimeField(auto_now_add=True)
    date_updated = models.DateTimeField(auto_now=True)
```

Figure 3.1.2 The campus\models.py file

It serves as the Model layer in Django's MTV architecture, utilizing the ORM to define and manage all campus-related data structures and their persistence. (Figure 3.1.2)

- **Frontend: HTML, CSS and JavaScript**

The front end directly affects the usage efficiency and participation rate. The objective of research is to verify which styles can reduce errors and time costs in actual applications. The user interface is constructed using the standard web triad: Using semantic HTML ensures that key functions can still be used even when JavaScript is absent or restricted. CSS for styling and layout, and JavaScript for dynamic interactivity. JavaScript is crucial for implementing dynamic Schedule Management features. This combination allows for the creation of a responsive and intuitive user experience. (Figure 3.1.3)

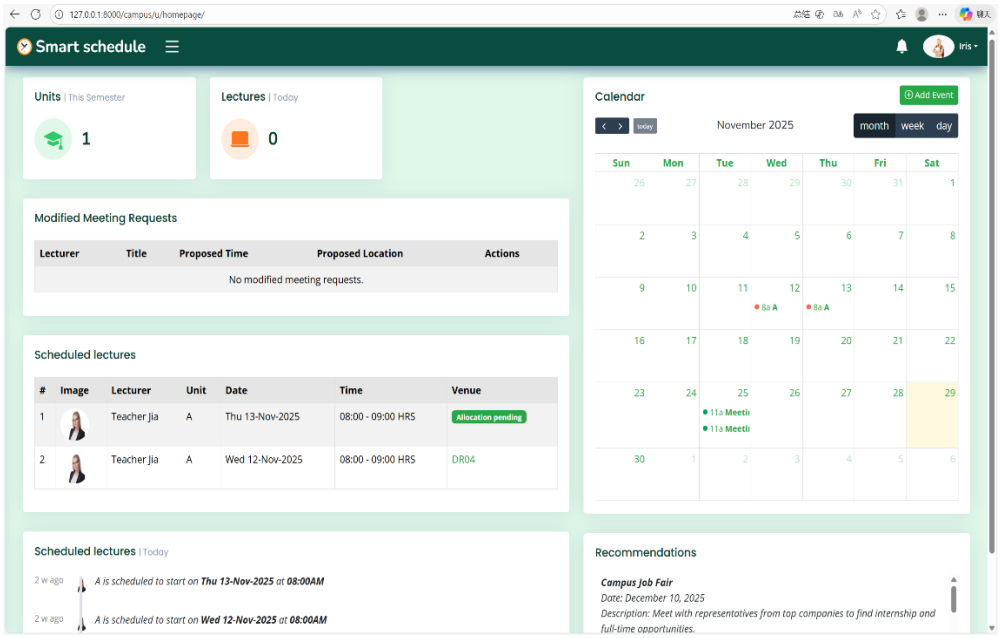


Figure 3.1.3 Output effect of the home page

- **Data Interchange Format: JSON**

JSON serves as the key for asynchronous data exchange between the frontend and backend. All API responses, including lists of events, to-do items, and user data, are formatted in JSON. This lightweight and human-readable format is perfectly suited for transmitting structured data over network requests and is natively supported by both Django and JavaScript.

Figure 3.1.4 popup_response.js.

JSON serves as the lightweight data interchange format that seamlessly bridges Django's backend with the frontend by serializing structured data into HTML and deserializing it for dynamic popup response handling. (Figure 3.1.4)

- **Version Control and Collaboration: GitHub**

GitHub is employed as the distributed version control platform for the project. It facilitates collaborative development, code review, issue tracking, and feature branching, ensuring code integrity and simplifying project management.

Figure 3.1.5 .gitignore file

It defines rules to exclude sensitive configuration files, virtual environments, and automatically generated directories from version control, safeguarding the

repository from unnecessary or exposed data. (Figure 3.1.5)

The following block diagram illustrates the architecture, depicting the data flow and technology usage from the user interface through the backend server to the notification channel (Figure 3.1.6 Block Diagram):

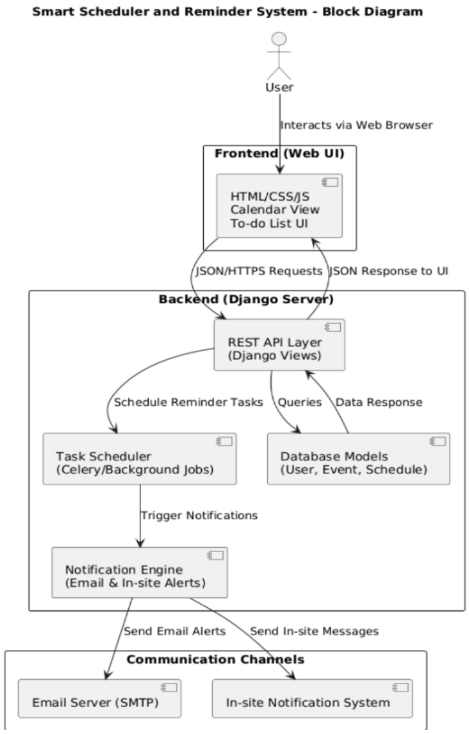


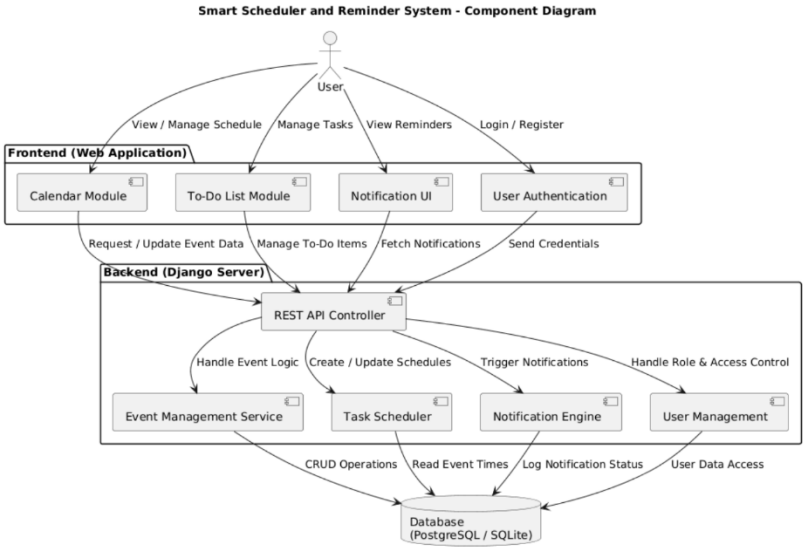
Figure 3.1.6. Block Diagram

3.2 Functional technical implementation

- **User Role Management:** Define permissions for different roles by using the built-in authentication system of Django.
- **Schedule and Event Management**: Through Django's MVT architecture, a complete set of schedule management functions has been provided, covering the entire process from data persistence, business logic processing to user interface presentation.
- **Two-Way Calendar Visibility System:** The front-end renders an interactive calendar view, while the back-end controls visibility through the Django permission model.
- **Reminders and Notifications:** A notification system built based on standard Django functions and driven by events.
- **To-Do List Integration:** The design model supports attributes such as deadlines and dependencies.

- **Participation Tracking and Report Generation:** The system compares the planned time with the actual access time, generates a user's schedule report, and provides a PDF export interface on the backend.
- **Suggestions Function**: Based on popular events or functions, the app will recommended tasks and events.

This component diagram illustrates the layered architecture, clearly presenting the relationship between functions and technologies (Figure 3.2. Component Diagram):



(Figure 3.2. Component Diagram)

3.3 System Architecture

The system adopts the classic Model-View-Template (MVT) architecture (Figure 8. Architecture Diagram), which is the implementation of the MVC pattern by Django. The key components are:

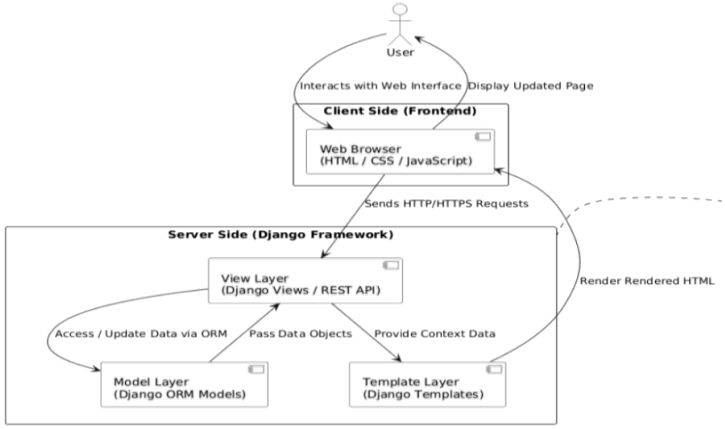


Figure 3.3 Architecture Diagram

- **The Model layer** is responsible for the data structure and business logic. It serves as an abstraction layer for interacting with the database, defining data entities, their relationships, and behaviors. Defined in files like accounts/models.py and campus/models.py. In This Project, it encapsulates the data structure of entities such as User, Event, Reminder, and Campus Event. It handles data validation, storage, and retrieval from the database using Django's Object-Relational Mapper (ORM), allowing developers to use Python code instead of raw SQL. It could include business logic methods, for example, conflict detection within an Event model.

  The User model defines user attributes (username, email, preferences) and authentication logic. The Event model defines fields for title, time, location, and may include methods to check for scheduling conflicts. The Reminder model defines reminder time and sending method.
- **The View layer** contains business logic. It processes user requests, interacts with the Model layer to fetch or manipulate data, and returns an HTTP response. In Django, the View acts as the intermediary between Models and Templates. In this Project, defined in files like accounts/views.py and campus/views.py. View functions or classes receive HTTP requests, process them (e.g., form submission, API call), and return responses. They call upon Model methods to perform CRUD (Create, Read, Update, Delete) operations on data. They typically render a Template by passing data to it in a context dictionary or return JSON responses for API endpoints.

  A login view validates user credentials and redirects upon success. A schedule view retrieves the list of events for the current user from the Event model and passes it to a template for rendering. A reminder settings view saves the user's notification preferences.
- **Template Layer (Template)**: The Template layer is responsible for presenting data to the user. It defines the structure and layout of the final HTML output, using a dynamic template language to display data passed from the View. In this project, templates (.html files) define the look and feel of the pages using HTML and the Django Template Language (DTL). They support template inheritance (e.g., a base.html for common layout) to avoid code repetition and maintain consistency.

3.4 Initial realization of planning and infrastructure setup

The initial implementation path of the core module of the system, which is the foundational work completed for setting up the development environment. Its content focuses on planning and preparation. (Figure 3.4 Figure 3.5)

```
E:\Project\smart-scheduler-main>python --version
Python 3.11.4

E:\Project\smart-scheduler-main>django-admin --version
4.2.24
```

Figure 3.4 Development Environment

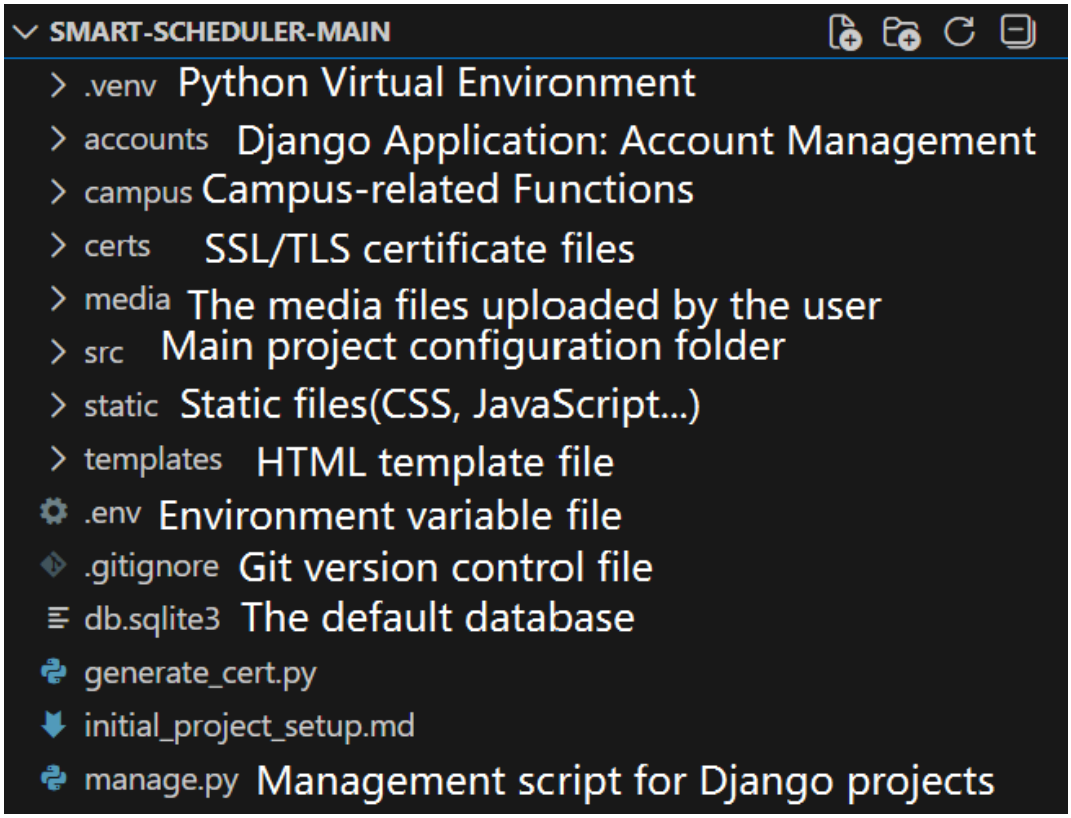


Figure 3.5 Project Framework

4. Expected contribution

This research outlines the design of a targeted solution to a well-defined problem of communication inefficiency. The expected contribution is a working web application that demonstrably reduces the manual effort required for schedule synchronization and increases activity participation rates within its target organizations.

5. Reference

[1] Timewatch. (2024). 50+ Time Management Statistics & Trends. Timewatch Blog. Retrieved from https://www.timewatch.com/blog/time-management-statistics/

[2] Google LLC. (2023). Google Calendar. https://calendar.google.com

[3] Notion Labs, Inc. (2023). Notion Calendar. https://www.notion.so/product/calendar

[4] Microsoft Corporation, “Microsoft Teams and Outlook Calendar Integration,” Microsoft Learn, 2024. [Online]. Available: https://learn.microsoft.com/en-us/microsoftteams/calendar-in-teams

[5] Stigler S, Burdack M. A practical approach of different programming techniques to implement a real-time application using Django[J]. Athens J. Sci, 2020, 7: 43-66.