Automated-TimeTableGenerator Project

# Project Overview

The Automated-TimeTableGenerator will be a web-based application designed for schools, colleges, or private office environments. Its primary function is to automatically generate non-conflicting timetables for classes, sections, staff, and branches. The project will handle various constraints like subject requirements, staff availability, and time slot preferences.

# Key Features

• Multi-Entity Timetable Generation: Timetables for various entities like students, teachers, branches, etc. Avoid conflicts in time slots, subjects, or staff.

• Scalability: The system should scale to handle multiple branches, departments, and subjects.

• User Roles: Admin, Teacher, and Student, with different capabilities for timetable view and management.

• Customization: Different inputs for institutions (like number of periods, holidays, preferred teaching times, etc.).

• Optimization: Ensure efficient allocation of available time slots and avoid empty periods in timetables.

• Hosting and Security: Website hosted with proper authentication for different users.

# Assumptions

• Each staff member can teach one or more subjects.

• Classrooms have capacity constraints and fixed hours.

• Teachers have limited availability and preferences for certain periods.

• Subjects have defined hours per week.

# Workflow

• Input Collection: Institution Details, Constraints (Teacher availability, subject hours, class capacity, etc.).

• Algorithm Design: A Constraint Satisfaction Problem (CSP) solver (such as backtracking or genetic algorithms) will be used for timetable generation.

• Timetable Generation: System ensures no clashes in teacher schedules, classroom allocations, or subject time slots.

• Feedback Mechanism: Allows reviewing, manual editing, and regeneration of timetables.

# Directory Structure

Here's a proposed directory structure for the project:

Automated-TimeTableGenerator/  
├── src/  
│ ├── algorithms/  
│ │ └── timetable\_generator.py # Core logic for timetable generation  
│ ├── models/  
│ │ └── user.py # User model (student, teacher, admin)  
│ │ └── class.py # Class model (subjects, sections, timings)  
│ │ └── timetable.py # Timetable model (periods, days)  
│ ├── services/  
│ │ └── timetable\_service.py # Business logic (scheduling, constraints handling)  
│ ├── controllers/  
│ │ └── timetable\_controller.py # API for timetable actions (CRUD operations)  
│ └── utils/  
│ └── validation.py # Utility functions for validation, error handling  
├── web/  
│ ├── static/  
│ │ └── css/ # CSS styles for front-end  
│ │ └── js/ # JavaScript files for interaction  
│ └── templates/  
│ └── index.html # Main home page template  
│ └── login.html # Login page  
│ └── dashboard.html # Admin/Teacher/Student dashboard  
│ └── timetable\_view.html # Timetable view page  
├── database/  
│ └── migrations/  
│ │ └── create\_tables.sql # SQL migration scripts for creating tables  
│ └── seed\_data.sql # Initial data for branches, subjects, teachers  
├── tests/  
│ └── test\_timetable.py # Unit tests for timetable generation  
│ └── test\_models.py # Unit tests for models  
├── config/  
│ └── settings.py # Configuration (db settings, time limits, etc.)  
├── app.py # Main application entry point  
├── requirements.txt # Python dependencies (Flask, SQLAlchemy, etc.)  
└── README.md # Project documentation

# Tech Stack

Frontend: HTML, CSS, JavaScript (optional: React.js)

Backend: Python (Flask/Django), SQLAlchemy (for database ORM)

Database: PostgreSQL or MySQL

Deployment: Docker for containerization, Nginx for web server, and AWS/GCP for cloud hosting.

# Code Snippets

## 1. timetable\_generator.py (Core Algorithm)

from models import Class, Teacher, Timetable  
import random  
  
def generate\_timetable(classes, teachers, slots):  
 timetable = Timetable()  
 for cls in classes:  
 for subject in cls.subjects:  
 allocated = False  
 while not allocated:  
 time\_slot = random.choice(slots)  
 teacher = random.choice(teachers)  
 if timetable.is\_available(teacher, time\_slot) and timetable.is\_classroom\_free(cls, time\_slot):  
 timetable.add\_entry(cls, subject, teacher, time\_slot)  
 allocated = True  
 return timetable

## 2. timetable\_service.py (Business Logic)

class TimetableService:  
 def create\_timetable(self, school, constraints):  
 classes = school.get\_classes()  
 teachers = school.get\_teachers()  
 slots = school.get\_available\_time\_slots()  
   
 generated\_timetable = generate\_timetable(classes, teachers, slots)  
 self.save\_to\_db(generated\_timetable)  
 return generated\_timetable

## 3. timetable\_controller.py (API Controller)

from flask import Flask, request, jsonify  
from services.timetable\_service import TimetableService  
  
app = Flask(\_\_name\_\_)  
  
@app.route('/generate\_timetable', methods=['POST'])  
def generate\_timetable():  
 data = request.json  
 school = data['school']  
 constraints = data['constraints']  
 timetable = TimetableService().create\_timetable(school, constraints)  
 return jsonify(timetable.serialize())  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)

# Optimization Strategies

1. Efficient Scheduling Algorithms: Use advanced scheduling algorithms like genetic algorithms to improve performance for larger institutions.

2. Cache Results: Use Redis or Memcached to cache results of timetable generation for quick retrieval.

3. Parallelization: Parallelize timetable generation for different branches or departments to reduce wait times.

CHALLENGES  
This document provides a detailed description of the Automated Timetable Generator project.   
It covers the key issues encountered during development, troubleshooting steps, solutions,   
and the necessary steps for deployment and running the project. The content is based on the development   
conversation and logs between the developer and support team.

# Table of Contents

1. Project Overview
2. Setup and Installation Issues
3. Local Deployment Problems
4. Railway Deployment Errors
5. TemplateNotFound Error
6. Final Project Structure
7. Step-by-Step Solutions
8. Conclusion

# 1. Project Overview

The Automated Timetable Generator is a Flask-based project   
 designed to create and manage school timetables automatically. It provides an API that allows   
 users to submit school and constraint data, and it generates timetables based on availability of   
 teachers, classrooms, and time slots.

# 2. Setup and Installation Issues

Throughout the project, we encountered several   
 issues with setting up the PostgreSQL database and configuring it for local development. The main problems were:  
   
- PostgreSQL installation failing due to database cluster initialization issues.  
   
- Missing template files like index.html during Flask app execution.  
   
- Incorrect directory structures and missing files in deployment.  
 The solution was to ensure the templates folder is placed correctly and permission issues are resolved.

# 3. Local Deployment Problems

There were multiple instances where local deployment failed,   
 primarily due to missing dependencies and template files. The issues encountered include:  
   
- TemplateNotFound errors: Flask was unable to locate the index.html file.  
   
- File paths for templates and static files were not set up correctly.  
 Solution: We restructured the project to ensure the templates and static directories were in the correct location   
 and Flask could access them.

# 4. Railway Deployment Errors

The initial deployment to Railway led to frequent app crashes.   
 Logs from the Railway deployment indicated issues related to the Werkzeug library and `url\_quote` import error.   
 Additionally, the deployment logs showed the worker process failing to boot.  
 Solution: Updated the dependencies in the `requirements.txt` file to ensure compatibility with Flask and Werkzeug   
 versions and used proper configuration settings for deployment.

# 5. TemplateNotFound Error

One recurring issue was the `jinja2.exceptions.TemplateNotFound` error.  
 This was because the Flask app could not locate the templates in the designated folder. We found that the templates   
 were either misplaced or the template folder path was incorrectly specified. Solution: Ensure that the templates folder   
 is placed at the root and the Flask app has the correct template folder location.

# 6. Final Project Structure

The final structure of the project is as follows:  
   
- /src/controllers/timetable\_controller.py  
   
- /templates/index.html  
   
- /src/services/timetable\_service.py  
   
- /src/algorithms/timetable\_generator.py  
   
- /src/models/timetable.py  
   
- /Procfile  
   
- requirements.txt  
   
This structure ensures the Flask app can locate all necessary files for generating the timetable.

# 7. Step-by-Step Solutions

Here is a step-by-step guide for resolving the most common issues encountered:  
   
1. PostgreSQL installation failure: Resolve by ensuring no other DB process conflicts and setting proper permissions.  
   
2. TemplateNotFound Error: Ensure templates are located in the correct folder and Flask is pointing to it.  
   
3. Railway Crashing: Update `requirements.txt` and ensure correct deployment configurations.  
   
4. Local Execution Errors: Activate virtual environment and ensure dependencies are installed properly.

# 8. Conclusion

The project encountered multiple issues, mostly related to file paths,   
 template rendering, and deployment. However, with proper structuring and troubleshooting, the project is now   
 fully functional. The key is to ensure all dependencies are properly installed and paths are correctly specified   
 during both local development and deployment.