GROUP M

RESOURCE MANAGEMENT SYSTEM

PROJECT CHARTER

VERSION 1.0

02/03/2025

REVISION HISTORY

Date	Version	Description	Author

DISTRIBUTION LIST

Name	Role	Date of distribution

APPROVALS

Name	Role	signature

CONTENTS

PURPOSE/PROJECT JUSTIFICATION

Abstract / Executive Summary:

Timetable scheduling is a complex optimization challenge faced by educational institutions, where courses, instructors, and classrooms must be allocated efficiently while ensuring no conflicts. The traditional manual scheduling process is often inefficient, error-prone, and time-consuming. This project proposes the development of an intelligent timetable scheduling system powered by a Genetic Algorithm (GA) to automate and optimize scheduling processes.

Project Justification:

The current scheduling methods lack flexibility and scalability, leading to issues such as resource underutilization, instructor overload, and scheduling conflicts. The proposed system will address these challenges by:

- Reducing the time and effort required for scheduling.
- Optimizing resource allocation to maximize efficiency.
- Ensuring fairness in workload distribution among instructors.
- Providing a flexible system that adapts to institutional constraints and modifications.
- Enhancing the accuracy and reliability of timetables with minimal manual intervention.

By leveraging Genetic Algorithms, this project will deliver an adaptive, data-driven solution that outperforms traditional scheduling techniques and meets the dynamic needs of academic institutions.

Timetable scheduling is a complex optimization problem that involves assigning courses, instructors, and classrooms to time slots while ensuring no conflicts and adhering to constraints. Traditional methods are often time-consuming and inflexible. This project aims to develop an automated timetable scheduling system using a Genetic Algorithm (GA), which provides efficient, optimized, and adaptive scheduling.

OBJECTIVES AND SUCCESS CRITERIA

Project Objectives:

The primary goal of this project is to develop an intelligent timetable scheduling system using a Genetic Algorithm (GA) that efficiently assigns courses, instructors, and classrooms to available time slots while minimizing conflicts and optimizing resource utilization.

Success Criteria:

Each objective will be evaluated based on measurable success criteria:

- **Conflict-Free Timetable Generation:**
  - Success is measured by achieving a conflict-free schedule in at least **95%** of generated timetables.
  - Reduction of scheduling conflicts compared to traditional manual methods.
- **Optimization of Resource Utilization:**

- The algorithm should ensure at least **90%** efficient use of classrooms and instructor availability.
- Balanced workload distribution among instructors with minimal overload cases.

❖ **Reduction in Manual Scheduling Effort:**
- The system should reduce manual scheduling time by at least **70%** compared to traditional methods.
- Automated scheduling should require minimal human intervention.

❖ **Flexibility and Adaptability:**
- The system should allow modifications with an **80%** success rate for requested changes without major disruptions.
- Users should be able to input institutional constraints and preferences dynamically.

❖ **User Satisfaction and Adoption Rate:**
- Achieve a **high user satisfaction rating (>85%)** based on feedback from university administrators and instructors.
- Ensure a smooth onboarding process with training materials and support.

REQUIREMENTS

❖ **Functional Requirements**

- **Automated Timetable Generation** : The system should generate schedules for courses, instructors, and classrooms.
- **Conflict Detection & Resolution** : Must ensure no scheduling conflicts occur.
- **Instructor Availability Management** : Faculty preferences and constraints should be accommodated.
- **Resource Utilization Optimization** : Efficient allocation of classrooms, instructors, and time slots.
- **Modification & Flexibility** : Support for changes in scheduling with minimal disruption.
- **User Input & Constraints Handling** : Allow institutions to set rules, preferences, and constraints dynamically.
- **Reporting & Analytics** : Generate reports on schedule efficiency, instructor workload, and resource utilization.

❖ **Non-Functional Requirements**

- **Scalability** : Should handle large datasets for different academic institutions.
- **Performance** : Must generate a timetable within an acceptable time limit.
- **Usability** : User-friendly interface for administrators and instructors.
- **Security** : Access control, data encryption, and user authentication.
- **Integration** : Must connect with Learning Management Systems (LMS) and Institutional Management Systems (IMS).
- **Reliability** : Ensure a high success rate in schedule generation with minimal manual intervention.

PROJECT ASSUMPTIONS AND/OR CONSTRAINTS

Assumptions:

The following assumptions are considered valid for the successful execution of this project:

- Sufficient historical scheduling data is available for algorithm training and testing.
- The system will be deployed in an academic institution with a structured timetable framework.
- The algorithm will run on standard computing environments without requiring specialized hardware.
- Users will provide all necessary constraints, preferences, and scheduling rules before execution.
- Institutional policies and academic regulations will remain stable throughout the development process.
- Stakeholders will actively participate in testing and provide timely feedback.

Constraints:

The project must adhere to the following constraints:

- **Regulatory and Academic Constraints:** The system must comply with institutional policies, faculty workload regulations, and academic calendar constraints.
- **Computational Limitations:** Limited computing resources may impact the complexity and efficiency of the genetic algorithm.
- **Time Constraints:** The project must be completed within the designated timeline, adhering to predefined milestones.
- **User-Defined Constraints:** The system must incorporate specific institutional requirements, such as room capacity limits, instructor availability, and subject-specific scheduling rules.
- **Security and Privacy Considerations:** The system must ensure data confidentiality and restrict unauthorized access to sensitive scheduling information.

PRELIMINARY RISK STATEMENT

- **Algorithm Convergence Issues**:
  - The Genetic Algorithm may not converge to an optimal timetable solution within the expected time frame.
  - Mitigation Strategy: Fine-tune GA parameters such as mutation rate and crossover strategy to enhance performance.
- **Conflicting Constraints:**
  - Institutional scheduling constraints may lead to infeasible timetables.
  - Mitigation Strategy: Implement a priority-based constraint resolution method to handle conflicts effectively.
- **Performance Bottlenecks:**
  - The algorithm may experience slow processing times due to complex constraints and large datasets.
  - Mitigation Strategy: Optimize data structures, parallelize processing, and use efficient memory management techniques.
- **User Adoption Challenges:**
  - Administrators and faculty members may be hesitant to adopt the new system.
  - Mitigation Strategy: Provide comprehensive training, user-friendly UI, and support materials to ease the transition.
- **Data Inaccuracy or Incompleteness:**

- Inaccurate or incomplete historical scheduling data may affect algorithm performance.
- Mitigation Strategy: Validate input data before processing and allow for manual adjustments when necessary.

❖ **Limited Computational Resources:**
- The algorithm may require high computational power, which may not be readily available.
- Mitigation Strategy: Optimize the code for efficiency and explore cloud-based solutions if necessary.

❖ **Security and Privacy Risks:**
- Unauthorized access to scheduling data could lead to misuse or tampering.
- Mitigation Strategy: Implement access control mechanisms, encryption, and secure authentication protocols.

SUMMARY MILESTONE SCHEDULE

Project Milestone	**Forecast Date**	**Owner**
Sprint 0: Stakeholder alignment, backlog creation, environment setup.	Week 1	Development Team
Sprints 1-2: User story mapping, MVP definition, algorithm proof-of-concept.	Week 2-4	
Continuous Sprint Planning: Prioritize backlog, refine user stories each sprint.	Ongoing	
Embedded in Sprints: Refine requirements iteratively via stakeholder feedback.	Week 10-13	
In-Sprint Design: UI/UX prototyping, database schema design per feature.	Weeks 13-16	
Incremental Delivery: Build features sprint-by-sprint (e.g., GA core, API integration).	Week 16-21	
Continuous Testing: Automated tests, performance tuning in each sprint.	Week 21-25	

Detailed Sprint Breakdown

Sprint	**Timeline**	**Focus**	**Deliverables**
0	Week 1	Project kickoff, stakeholder alignment, tool setup (GitHub, Jira).	- Prioritized product backlog - Initial GA research report.
1-2	Weeks 2-4	Core algorithm validation, user story mapping.	- MVP algorithm prototype - Backlog refinement.
3-6	Weeks 5-13	Feature development (e.g., constraint input UI, API integration).	- Functional modules (e.g., conflict detection) - Updated UML diagrams.

7-8	Weeks 14-16	UI/UX finalization, database optimization.	- Interactive UI mockups - Optimized database schema.
9-10	Weeks 17-21	Full-stack integration, end-to-end testing.	- Working prototype with core features.
11-12	Weeks 22-25	Performance tuning, user training, documentation.	- Final system deployment - User manuals and support plans.

PROJECT ORGANIZATION

ROLES AND RESPONSIBILITIES

Team Approach

Our team operates under a **dynamic, collaborative model** where responsibilities are shared equally and rotated based on project needs, skillsets, and sprint priorities. Instead of fixed roles, we emphasize **cross-functional ownership** of work packages. This ensures:

- ❖ **Flexibility**: Members adapt to tasks ranging from UI/UX design to backend development, testing, and documentation.
- ❖ **Equal Workload Distribution**: Tasks are assigned transparently during sprint planning to balance effort and expertise.
- ❖ **Skill Development**: Members gain exposure to diverse aspects of the project, fostering a well-rounded team.

Principles

1. **Sprint-Based Rotation**: Roles shift every sprint based on task priorities and individual bandwidth.
2. **Pair Programming**: Complex tasks (e.g., algorithm tuning) are tackled collaboratively.
3. **Collective Accountability**: All members review pull requests, participate in code reviews, and share ownership of deliverables.
4. **Stakeholder Feedback**: Regular demos ensure alignment with stakeholder expectations despite role fluidity.

APPENDICES

A. PROJECT MANAGEMENT APPROACH

- Adopted Agile methodology with iterative development.
- Used GitHub for task management and Docker for containerization.
- Maintained multiple GitHub repositories for task segregation and documentation (e.g., requirement matrix).

B. TECHNICAL PROGRESS

- **Backend Frameworks:** Evaluated Python Django vs. Node.js; leaning toward Python for algorithm implementation.
- **Algorithm Selection:** Genetic Algorithm (GA) chosen for scheduling, with exploration of hybrid approaches (GA + Constraint Programming).
- **API Integration:** Initiated planning for data retrieval from the faculty's LMS/IMS.
- **Design:** Began UML and use case diagrams for Object-Oriented Analysis and Design (OOAD).

C. COLLABORATION & STAKEHOLDER ENGAGEMENT

- Conducted meetings with senior students for technical insights.
- Planned stakeholder engagement (Deputy Registrar) for requirement validation.

D. KEY CHALLENGES IDENTIFIED

TECHNICAL COMPLEXITY

- Difficulty in implementing advanced algorithms (GA, Constraint Programming) in Python.
- Uncertainty about third-party API limitations (speed, data formats).

DOCUMENTATION & REQUIREMENTS

- Incomplete requirement documentation risks scope creep.
- Need for a structured requirement matrix to align deliverables.

TOOLING & WORKFLOW

- Ensuring an error-free GitHub main branch while following Agile principles.
- Finalizing Docker workflows for consistent deployment.

E. ACTIONABLE RECOMMENDATIONS

TECHNICAL IMPROVEMENTS

- **Proof of Concept (PoC):** Build a small-scale GA prototype to validate conflict resolution and performance.

- **Hybrid Algorithm Testing:** Explore combining GA with Constraint Programming for faster convergence.
- **API Mocking:** Use tools like Postman or Swagger to simulate third-party APIs during development.

DOCUMENTATION & PROCESS

- **Centralized Documentation:** Use a wiki (e.g., GitHub Wiki/Confluence) for requirements, risk registers, and design diagrams.
- **Requirement Matrix Template:** Include columns for Priority, Status, and Stakeholder Approval.

GITHUB & DEVOPS

- **Branching Strategy:** Implement GitFlow to isolate features, bugs, and releases.
- **CI/CD Pipeline:** Set up GitHub Actions for automated testing and deployment.

STAKEHOLDER ENGAGEMENT

- **Weekly Demos:** Showcase progress to stakeholders (e.g., Deputy Registrar) to gather feedback early.
- **Risk Workshops:** Collaborate with senior students to identify mitigation strategies for technical risks.