A SMART TIMETABLE GENERATOR

BY:

ADEBISI SIJUWADE MICHAEL

(21/10MSC006)

A PROJECT SUBMITTED

TO

DEPARTMENT OF MATHEMATICAL AND COMPUTING SCIENCES

FACULTY OF COMPUTING AND APPLIED SCIENCES

THOMAS ADEWUMI UNIVERSITY, OKO-IRESE, KWARA STATE,

NIGERIA.

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF BACHELOR OF SCIENCE (B.Sc.) IN COMPUTER SCIENCE.

AUGUST 2025

CERTIFICATION

This is to certify that I, **Adebisi Sijuwade Michael** (Matric No: **21/10MSC006**), am responsible for the work submitted in this project titled **Smart Timetable Generator.** The content of this project is my original work, except where due acknowledgment and references have been made. I further confirm that this project has not been previously submitted, either wholly or partially, to this University or any other institution for the award of any degree or certificate.

APPROVAL

This project has been approved for the Department of Mathematical and Computing Sciences, Thomas Adewumi University, Oko, Kwara State, Nigeria.

Dr Ayetiran Eniafe ……

(Supervisor) Signature and Date

Dr Folaranmi Rotimi Oluwasegun ……..

(Co-Supervisor) Signature and Date

Mr. Omosola Olabode ……..

(Head of Department) Signature and Date

DEDICATION

This project is dedicated to God Almighty for the abundant grace, wisdom, knowledge, and skills given to me through my life, especially during my stay at Thomas Adewumi University, Oko, Kwara State, Nigeria.

ACKNOWLEDGEMENT

I wish to express my deepest gratitude to my supervisor, **Dr. Ayetiran Eniafe**, for his invaluable guidance, continuous encouragement, and insightful feedback that shaped the direction of this work. I am also sincerely thankful to my co-supervisor, **Dr. Folaranmi Rotimi Oluwasegun**, for his consistent support, mentorship, and constructive input throughout the project.

I gratefully acknowledge the support of the **Head of Department of Mathematical and Computing Sciences**, whose academic environment and resources provided a strong foundation for my research. I also extend my appreciation to the **Faculty of Computing and Applied Sciences** for the opportunity to study and grow in such a conducive learning environment.

My sincere appreciation goes to **Thomas Adewumi University**, Oko-Irese, Kwara State, for providing a serene campus, quality academic facilities, and an enabling environment that made this research possible.

My heartfelt thanks go to my **family, friends, and colleagues** for their unwavering support, encouragement, and prayers throughout this journey. Their love and belief in me have been a source of strength and motivation, especially during challenging times.

Above all, I give **all glory, honor, and praise to Almighty God** for His guidance, strength, wisdom, and unending grace throughout the course of my academic journey. Without His divine support, this accomplishment would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This project describes the design and development of a **Smart Timetable Generator**, a web-based system created to automate the process of building academic timetables. In many institutions, timetables are still prepared manually, which often leads to mistakes such as class clashes, double bookings, and uneven workloads. These issues waste time, create confusion, and make management difficult.

The Smart Timetable Generator solves these problems by using a **Genetic Algorithm (GA)**, an approach inspired by natural selection, to automatically create conflict-free timetables. The system takes into account important factors such as lecturer availability, room sizes, course credit units, and institutional rules. It was built with modern web technologies, including **Laravel, MySQL, HTML, CSS, and JavaScript**, and provides a simple interface for administrators to add data, generate timetables, and export them to PDF.

The system was tested with real academic data and produced reliable timetables within minutes. It reduces errors, saves effort, and adapts to changes more quickly than manual methods. Overall, the project shows how combining algorithms with web technology can improve efficiency in schools and universities, while also contributing to research on solving complex scheduling problems.

CHAPTER 1

INTRODUCTION

1.1 Background of the Study

Creating conflict-free academic timetables is a major challenge for educational institutions. The process requires balancing courses, instructors, rooms, and student needs, while avoiding overlaps and resource clashes. Traditionally, administrators spend days or weeks manually scheduling, which is feasible only on a small scale. As the number of courses, staff, and rooms increases, manual methods become time-consuming, error-prone, and inefficient. Studies confirm that timetabling is an NP-hard problem, with complexity rising sharply as institutional size grows (Tarale & Bhuyar, 2025; Mahlous & Mahlous, 2023).

Manual timetables often contain errors such as double-booked rooms, overlapping classes, or uneven workloads, leading to missed classes, administrative stress, and poor resource utilization (Garcia Yáñez et al., 2024; Diallo & Tudose, 2024). These limitations highlight the need for automation. Recent research shows that advanced algorithms and digital systems can reduce errors and improve scheduling efficiency (Ambhore et al., 2020; D'Souza et al., 2020).

The proposed **Smart Timetable Generator** addresses these challenges by combining a **Genetic Algorithm (GA)** with modern web technologies (HTML, CSS, JavaScript, Laravel). GAs simulate natural selection, generating candidate schedules and refining them through selection, crossover, and mutation until near-optimal solutions emerge. This approach enables efficient handling of large, complex scheduling problems while minimizing conflicts and maximizing resource utilization (Tarale & Bhuyar, 2025).

The system includes a **web-based interface** that supports real-time updates. When administrators make changes such as adjusting a class time or venue updates instantly reflect across all users' devices. This ensures accuracy, collaboration, and timely access to current schedules. By shifting from paper-based methods to digital automation, the system reduces administrative burden, supports better communication, and enhances overall efficiency.

Beyond automation, the system empowers institutions by freeing administrators from repetitive scheduling tasks, enabling them to focus on strategic roles, while ensuring instructors receive balanced workloads and students gain conflict-free timetables. Automated scheduling also aligns with the ongoing digital transformation in education, promoting modernization, data-driven decision-making, and improved service delivery (Kavade et al., 2023).

The Smart Timetable Generator is **scalable and adaptable**, making it suitable for small colleges as well as large universities with thousands of students and complex departmental structures. It can also be customized to reflect institutional policies and preferences, ensuring flexibility across diverse contexts (Ambhore et al., 2020).

1.2 Statement of the Problem

Manual timetable scheduling in educational institutions is time-consuming, error-prone, and often leads to conflicts and inefficient resource use especially as complexity increases (Mahlous & Mahlous, 2023). Existing automated systems are frequently rigid, relying on fixed rules that cannot adapt to real-time changes like new courses, staff availability, or room adjustments.

This project addresses the lack of flexible, efficient, and intelligent scheduling solutions by developing a Smart Timetable Generator. It uses a genetic algorithm to automatically produce

conflict-free timetables and adapts quickly to changes. Built as a web-based application, it ensures accessibility and usability.

1.3 Aim and Objectives of the Study

1.3.1 Aim

The aim of this study is to develop a web-based Smart Timetable Generator that uses genetic algorithms to automate and optimize academic scheduling in an educational institution.

1.3.2 Objectives

To fulfill the aim of the project, the following are the objectives:

1. Build a web-based timetable system that works well on different devices (desktops, tablets, phones).
2. Create an easy-to-use interface for administrators to enter course, teacher, room, and time data, and to view or manage timetables.
3. Integrate a genetic algorithm into the system and deploy it at the case study institution for real-world use.
4. Test the system to see how fast it generates timetables, how well it avoids scheduling conflicts, and how users rate its ease of use and effectiveness.

1.4 Significance of the Study

This project holds significant value across practical, institutional, academic, and research domains by addressing a critical challenge in educational management through innovative technological solutions.

1. **For Users:**

The Smart Timetable Generator provides an intuitive, automated system that drastically reduces the time, effort, and errors associated with manual scheduling. With real-time updates and built-in conflict detection, administrators, faculty, and students gain access to accurate, reliable timetables. The user-friendly interface and automation of complex decisions enhance productivity, minimize scheduling stress, and improve overall user satisfaction.

2. **For Institutions:**

Educational institutions benefit from optimized resource utilization of classrooms, instructors, and time slots are allocated efficiently and fairly. The system ensures balanced faculty workloads and enables rapid adjustments to unexpected changes (e.g., room unavailability or staff absences), supporting operational continuity. By reducing administrative overhead and preventing scheduling conflicts, the tool enhances institutional efficiency, flexibility, and transparency in academic planning.

3. **For Research:**

This project demonstrates a practical application of Genetic Algorithms (GAs) to solve the NP-hard timetabling problem, contributing to the fields of artificial intelligence and operations research. It illustrates how evolutionary computation can be effectively applied in real-world decision-support systems. Insights into algorithm design, constraint handling, and performance evaluation offer a foundation for future research such as hybrid optimization models or improved metaheuristic strategies advancing the state of the art in automated scheduling.

4. **For Academia:**

As a multidisciplinary case study, the Smart Timetable Generator bridges theoretical concepts and real-world implementation in computer science and software engineering. It encompasses

optimization, AI, web development, and human-computer interaction, making it a valuable teaching tool. Educators can use the project to demonstrate the application of algorithms like GA in solving complex problems, while students can learn from its full-stack development lifecycle. The work also supports experiential learning and can inspire further academic projects, theses, or curriculum development.

1.5 Scope of the Study

This study encompasses the design, development, and deployment of a web-based Smart Timetable Generator aimed at automating and optimizing academic scheduling. The scope is defined across three core dimensions: technology stack, algorithmic framework, and system features.

1. **Technology Stack:**

The system employs standard web technologies: HTML, CSS, and JavaScript for a responsive front-end interface compatible with all major browsers. The back end is built using the Laravel PHP framework with a MySQL relational database for secure, persistent data storage. This architecture ensures modularity, scalability, and full institutional control, supporting self-hosting and potential adaptation to other server environments.

2. **Algorithmic Framework:**

At the core of the system is a Genetic Algorithm (GA), an evolutionary optimization method suited to complex, constraint-rich scheduling problems. The GA generates an initial population of timetable solutions and evaluates each based on a fitness function that penalizes conflicts (e.g., overlapping classes, instructor overloads) and promotes balanced workloads. Through iterative processes of selection, crossover, and mutation, the algorithm evolves increasingly fit solutions

until a near-optimal, conflict-minimized timetable is achieved. This approach enables efficient exploration of vast solution spaces, making automated, high-quality scheduling feasible.

3. System Features:

The Smart Timetable Generator includes intelligent functionalities to support diverse user needs:

- **Conflict Detection:** Automatically identifies and prevents scheduling conflicts such as double-booked rooms or instructors assigned to overlapping classes.
- **Dynamic Updates:** Allows authorized users to modify inputs (e.g., add/remove courses, update availability), with the system adapting the timetable in real time without requiring full regeneration.
- **Role-Based Access Control:** Implements user roles (administrator, faculty, student) with tailored permissions administrators manage data and generate schedules, faculty view or request changes, and students access only their personal timetables ensuring security and workflow efficiency.
- **Real-Time Synchronization:** Ensures that timetable updates made by administrators are instantly reflected across all user interfaces, enabling timely coordination and reducing delays in communication.

1.6 Limitations of the Study

While the Smart Timetable Generator offers an effective solution for academic scheduling, several limitations must be acknowledged:

- **Hardware Constraints:** The system's performance depends on the user's device and browser. Older hardware or outdated browsers may result in slow loading, reduced

interactivity, or compatibility issues. Since the application relies on client-side processing and real-time communication, users with limited computing power or unstable internet connections may experience suboptimal performance.

- **Software Limitations:** As a web-based system, it requires a stable internet connection and a modern browser. Offline access is not supported, and users cannot view or edit timetables without connectivity. The system is hosted on an institutional server, so its availability is subject to server uptime and network reliability downtime or outages will disrupt access.
- **Data Availability and Accuracy:** The quality of the generated timetable depends on accurate, complete, and up-to-date input data, including course details, instructor availability, room capacities, and institutional constraints. Inaccurate or missing data may lead to invalid or suboptimal schedules (e.g., undetected conflicts). This study assumes reliable data input; in practice, successful deployment requires robust data management processes within the institution.
- **Algorithm Complexity and Scalability:** The Genetic Algorithm (GA) used for optimization is efficient but computationally intensive. Performance may degrade with larger institutions involving hundreds of courses, rooms, and complex constraints. Solution quality and convergence speed depend on parameter tuning (e.g., population size, mutation rate), and there is no guarantee of a globally optimal solution. In highly constrained or large-scale scenarios, the GA may require significant time or hybrid enhancements (e.g., combining with local search methods) to maintain efficiency and solution quality.

Recognizing these limitations helps contextualize the system's scope and performance. They also highlight opportunities for future improvements such as enabling offline functionality, strengthening data integration, and enhancing algorithmic scalability without undermining the

system's practical value. These considerations are essential for effective deployment and further development.

1.7 Organization of the Project

This report is organized into five chapters, guiding the reader from problem identification to solution development, evaluation, and future outlook:

Chapter 1: Introduction

Introduces the timetabling challenges faced by educational institutions and the need for automation. It presents the problem statement, aims, and objectives of the Smart Timetable Generator. The chapter also discusses the project's significance, scope, limitations, and key definitions. Thomas Adewumi University is established as the case study, providing context for system implementation.

Chapter 2: Literature Review

Surveys existing approaches to academic timetabling, including manual methods and algorithmic solutions. Focus is placed on optimization techniques particularly Genetic Algorithms and other metaheuristics used in scheduling systems. The review evaluates strengths and limitations of prior work, identifying gaps that justify the design and innovation of the proposed system.

Chapter 3: System Design and Methodology

Details the architecture and development approach of the Smart Timetable Generator. It describes the system components front-end interface, back-end services, and GA-based scheduling engine and explains how requirements were translated into technical design. The chapter outlines the development methodology (e.g., iterative or agile), technology choices, and the formulation of the genetic algorithm, including chromosome encoding, fitness function, and constraint handling.

Chapter 4: Implementation and Results

Presents the system's technical realization, including implementation challenges and integration of components such as the web interface and Laravel-based backend. It covers the testing strategy and results: functional testing (correctness of timetable generation), performance testing (speed and scalability), and usability feedback from users at the case institution. Bugs encountered and refinements made are documented, along with an assessment of how well the system met its initial objectives.

Chapter 5: Summary, Conclusion, and Recommendations

Summarizes the project and evaluates the extent to which objectives were achieved. It discusses the system's potential impact such as time savings and improved scheduling quality and highlights contributions to automated timetabling using GA and web-based deployment. The chapter concludes with recommendations for future enhancements, including hybrid algorithms, predictive scheduling using machine learning, and broader institutional deployment.

1.8 Definition of Terms

To ensure clarity, the following key terms are defined as used in this project:

- **Timetable**: A structured schedule that assigns academic activities (e.g., classes, lectures) to time slots, instructors, and venues (e.g., classrooms) over a defined period (week or semester). It coordinates students, faculty, and facilities to prevent conflicts and ensure smooth academic operations.
- **Genetic Algorithm (GA)**: An evolutionary optimization technique inspired by natural selection. It evolves a population of candidate solutions through selection, crossover, and

mutation to achieve high-quality results. GAs are particularly effective for complex problems like timetabling, where exhaustive search is impractical.

- **Conflict-Free Scheduling**: The development of a timetable without overlaps or violations of constraints, such as double-booking rooms or assigning an instructor to multiple classes simultaneously. Achieving conflict-free schedules is a core objective of the Smart Timetable Generator (Diallo & Tudose, 2024).
- **Web-Based Application**: Software hosted on a remote server and accessed via a web browser. It requires no local installation, supports cross-platform use (desktop and mobile), and allows centralized updates. The Smart Timetable Generator adopts this model to ensure accessibility and ease of maintenance.
- **Optimization**: The process of producing the most effective timetable by satisfying all hard constraints (e.g., no conflicts, room capacities) and maximizing soft constraints (e.g., preferred teaching times, balanced workloads). The GA optimizes schedules by evolving solutions based on a defined fitness function.
- **Timetabling**: The assignment of classes to time slots, instructors, and venues while meeting institutional constraints, ensuring a coherent and conflict-free plan (Techie-Menson & Nyagorme, 2021).
- **Simulated Annealing (SA)**: A probabilistic metaheuristic that avoids local optima by initially accepting worse solutions ("high temperature") and gradually reducing this acceptance ("cooling") to converge toward a strong solution.
- **Heuristic**: A practical, rule-based problem-solving approach that finds good solutions quickly when exhaustive search is infeasible. Though not always optimal, heuristics reduce computation time by focusing on promising areas of the solution space.

- **Metaheuristic**: A higher-level strategy (e.g., GA, SA, Tabu Search) that guides heuristics to explore solution spaces efficiently. Metaheuristics are widely applied to large, complex optimization problems to find near-optimal solutions.
- **Real-Time Synchronization**: A cloud-based feature that ensures timetable updates are instantly reflected across all user devices, enabling collaboration and data consistency for multiple users simultaneously.

CHAPTER TWO

LITERATURE REVIEW

2.1 Historical Perspectives of Automated Timetabling

Producing conflict-free academic timetables has long been a critical challenge. Manual scheduling balancing classes, faculty availability, room capacity, and course requirements was feasible in small institutions but became error-prone and inefficient in larger universities. Issues like class overlaps and double-booked rooms made the process time-consuming and suboptimal, highlighting the need for systematic methods (Tarale & Bhuyar, 2025).

Early Manual and Rule-Based Approaches: Initially, timetables were created using spreadsheets or paper charts. These labor-intensive methods often produced conflicts and poor resource use. With computerization, rule-based systems encoded constraints as logical rules, reducing manual effort but remaining rigid and unscalable. As courses and faculty grew, rule sets became overly complex, limiting practicality (Williams & Ajinaja, 2019).

Heuristic and Meta-heuristic Methods: From the 1980s, heuristics like Tabu Search and Simulated Annealing improved timetable generation. Tabu Search avoided revisiting poor solutions, while Simulated Annealing allowed occasional worse choices to escape local optima. These methods produced fewer conflicts and better room use compared to earlier systems (Ambhore et al., 2020). Population-based strategies like Scatter Search also emerged, maintaining diverse candidate timetables and inspiring evolutionary approaches such as Genetic Algorithms.

Genetic Algorithms (GAs): Introduced by Holland (1975), GAs became widely used due to flexibility and robustness. Timetables are encoded as chromosomes, evaluated by fitness functions considering conflicts, workloads, and capacities. Through selection, crossover, and mutation, solutions evolve toward high-quality schedules. Studies show GA-based systems reduce clashes and optimize resources (Kavade et al., 2023; Thakare et al., 2020). Although not globally optimal, GAs efficiently solve NP-hard problems, making them central to modern timetabling research.

Hybrid Algorithms and Enhancements: Standard GAs face premature or slow convergence, so hybrids integrate techniques like Simulated Annealing for global and local search balance (Maneesha et al., 2021). Other improvements include tailored genetic operators, adaptive mutation, and multi-objective optimization for balancing constraints and preferences (Williams & Ajinaja, 2020; Chatterjee et al., 2022; Das et al., 2020). These enhancements improve both reliability and quality of schedules.

Machine Learning Integration: Recently, ML has been combined with optimization to create smarter timetabling. ML can learn from historical data to predict slot assignments or guide search. For instance, Prashanta Kumar et al. (2020) applied decision trees and regression models to automate assignments, later refined with GAs. Ambhore et al. (2020) combined predictive analytics with evolutionary algorithms for multi-course scheduling. While effective, ML requires large datasets and frequent retraining to adapt to changing constraints, so it usually complements rather than replaces metaheuristics.

Modern Web-Based Systems: Advances in technology shifted timetabling from desktop software to web and cloud platforms. Web-based systems allow administrators, faculty, and students to

access and update timetables in real time. Platforms like Firebase provide instant synchronization, as in Kavade et al. (2023). Frameworks such as PHP, MySQL, JavaScript, or Flutter enable responsive, cross-platform applications (Techie-Menson & Nyagorme, 2021). Benefits include collaborative editing and integration with other online tools, though challenges like internet reliability, security, and privacy remain.

2.2 Theoretical Framework

The theoretical framework underpins how the Smart Timetable Generator formulates and solves the scheduling problem. This framework draws on concepts from evolutionary computation, combinatorial optimization, and software architecture for real-time systems. Key components include Genetic Algorithms and related evolutionary principles, constraint satisfaction and optimization theory, complementary metaheuristic techniques, and computing principles for deploying a responsive web-based application. Together, these theories provide the foundation for designing a system capable of generating effective timetables in a complex, real-world academic setting.

2.2.1 Genetic Algorithms (GAs)

Genetic Algorithms (GAs), introduced by Holland (1975), are evolutionary algorithms inspired by genetics and natural selection, well-suited for highly constrained problems like timetabling. A GA maintains a population of candidate solutions (timetables), each represented as a chromosome encoding course-to-timeslot-room assignments.

A fitness function evaluates these schedules, penalizing violations of hard constraints (e.g., double-booked instructors, over-capacity rooms) and rewarding satisfaction of soft constraints (e.g.,

balanced teaching loads, minimal student free periods). The fitness function is crucial as it guides the search toward better schedules.

The evolutionary cycle involves **selection**, **crossover**, and **mutation**. Selection chooses fitter timetables as parents, using methods like roulette wheel (probability-based) or tournament (competition-based). Crossover recombines parts of two parent schedules, mixing beneficial features, while mutation randomly alters assignments (e.g., moving a class), maintaining diversity and preventing premature convergence.

This process repeats over generations until termination criteria (e.g., maximum iterations or conflict-free timetable) are met. GAs efficiently navigate the vast search space by exploiting good partial solutions while exploring new ones. Prior research confirms their effectiveness in producing high-quality timetables for complex institutions where exhaustive search is infeasible (Kavade et al., 2023; Ambhore et al., 2020). Moreover, their population-based nature enables parallel processing, enhancing scalability.

2.2.2 Evolutionary Computation and Optimization Principles

Genetic Algorithms (GAs) are part of evolutionary computation, which draws inspiration from Darwin's theory of natural selection to solve optimization problems. They work by evolving a population of potential solutions over generations, using operators such as **selection, crossover, and mutation** to explore and refine the search space.

The university timetabling problem is a **combinatorial optimization** challenge with a huge number of possible combinations of courses, lecturers, venues, and time slots. Since it is **NP-hard**,

finding exact solutions is computationally infeasible for large cases. Instead, GAs focus on generating **feasible and near-optimal timetables** within reasonable time.

Constraints guide this optimization. **Hard constraints** (e.g., preventing lecturer or venue clashes) define which solutions are valid, while **soft constraints** (e.g., lecturer preferences, even distribution of classes, or minimizing student free periods) improve timetable quality. These soft constraints are often expressed as **penalties in the fitness function**, allowing the GA to balance feasibility and quality.

In more complex cases, multiple objectives must be optimized simultaneously such as minimizing conflicts, maximizing venue utilization, and accommodating preferences. Here, **multi-objective evolutionary algorithms** provide a set of trade-off solutions (Pareto front), from which administrators can choose the most suitable timetable.

2.2.3 Complementary Metaheuristic Techniques

Although the Smart Timetable Generator primarily relies on a Genetic Algorithm (GA), additional metaheuristic strategies can be integrated to overcome GA limitations and improve results.

- **Simulated Annealing (SA):** When combined with GA (a GASA hybrid), SA performs a local search on GA-generated timetables, occasionally accepting worse solutions to escape local optima. This improves convergence and prevents stagnation (Maneesha et al., 2021).

- **Tabu Search:** Used as a post-optimization step, Tabu Search refines GA solutions by systematically exploring swaps or reassignments while avoiding repeated poor states. This helps remove residual conflicts and fine-tune soft constraints.
- **Local Search & Memetic Algorithms:** Simple heuristics (e.g., greedy swaps, hill-climbing) can be embedded in GA's mutation operator for targeted improvements. This "memetic" approach combines GA's global exploration with focused local refinement.
- **Scatter Search:** By maintaining a reference set of diverse solutions and recombining them, Scatter Search injects variety into the GA population, reducing premature convergence (Ambhore et al., 2020).

2.2.4 Fitness Function Design

The fitness function is the central evaluation mechanism of the Genetic Algorithm (GA), responsible for distinguishing feasible timetables from poor ones. It translates scheduling rules into a numerical score, ensuring that the algorithm evolves toward solutions that are both valid and practical.

The design integrates three key elements:

1. **Hard Constraints (Strict Rules):** These are compulsory requirements that must always be satisfied, such as preventing lecturer or room double-booking, ensuring courses receive the correct number of sessions per week, and observing fixed institutional policies (e.g., prayer breaks, sports hours). Any violation attracts very high penalties or results in the timetable being considered invalid.

2. **Soft Constraints (Preferences):** These represent desirable but non-mandatory conditions, such as reducing back-to-back lectures for lecturers, minimizing student movement between distant venues, or balancing class loads across the week. Violations reduce the fitness score, while satisfaction increases it, allowing flexibility without rejecting feasible solutions.
3. **Weighted Objectives:** Since not all constraints carry equal importance, the fitness function applies weights to balance them. For instance, lecturer clashes are given heavier penalties than venue preferences. Weights can be adjusted to reflect institutional priorities, as demonstrated by Kavade et al. (2023), ensuring the GA produces solutions aligned with real-world expectations.

By combining these components into a single evaluation score, the GA can systematically guide the timetable population toward optimal schedules. A well-balanced fitness function prevents the algorithm from favoring technically valid but impractical timetables, ensuring the final result meets both academic requirements and user satisfaction.

2.2.5 Computing Principles for a Web-Based Implementation

The Smart Timetable Generator applies modern web principles to ensure responsiveness, scalability, and collaboration.

1. Web Frameworks: The system is built on Laravel, a PHP framework providing MVC structure, security, and database abstraction. It supports RESTful APIs for flexible integration and manages the timetable generation logic. On the client side, modern JavaScript or cross-platform frameworks

(e.g., Flutter) enable responsive interfaces across devices. Kavade et al. (2023) showed that combining GA with Flutter and Firebase yields seamless multi-platform scheduling.

2. Security and User Management: Since the platform serves multiple roles (admins, staff, students), strong authentication and role-based access control are required. Laravel's authentication tools, two-factor login for admins, and HTTPS encryption safeguard data. Role restrictions ensure, for example, that only authorized staff edit timetables while students view them. Sensitive data (like staff unavailability) is protected from unintended exposure.

3. Scalability and Performance: Built for cloud deployment, the system can scale servers and databases during peak usage. Laravel's ORM (Eloquent) supports efficient queries, while caching optimizes repeated timetable views. GA parameters (population size, generations) can be tuned for acceptable run-times, and heavy computations offloaded to background or cloud processes to maintain responsiveness.

2.3 Review of Related Work

To contextualize the development of the Smart Timetable Generator, this section reviews recent literature on automated timetable scheduling systems. The focus is on prominent approaches from 2020 onwards, encompassing pure metaheuristic solutions, hybrid algorithms, machine learning integrations, and web-based implementation case studies. We highlight key contributions, strengths, and limitations of each to inform how our work builds on the state of the art.

2.3.1 Summary of Existing Studies

GA-Based Optimization:

Genetic Algorithms remain central in timetabling research. Thakare et al. (2020) applied a GA to minimize class clashes and balance instructor workloads. Their encoding of courses, instructors, times, and rooms led to substantial conflict reduction, faster timetable generation, and improved data accuracy. Challenges included adapting to changing requirements and resistance from users accustomed to manual scheduling. Williams and Ajinaja (2019) enhanced GA performance with custom chromosome design and adaptive mutation, yielding faster convergence and fewer conflicts than standard GAs. In a follow-up (2020), they integrated context-based reasoning to address institutional policies and refined adaptive mutation. While effective at satisfying hard constraints and reducing overlaps, GAs cannot guarantee optimality and often run slower than simpler heuristics. Added features like context reasoning improved flexibility but introduced extra computational cost and reduced precision.

Hybrid and Improved Metaheuristics

To overcome GA limitations, hybrids have emerged. Maneesha et al. (2021) combined GA with Simulated Annealing (SA): the GA generated feasible timetables, and SA refined them through local improvements, reducing conflicts further than GA alone. This improved convergence but required higher computation and parameter tuning. Ambhore et al. (2020) integrated ML with GA, Tabu Search, SA, and Scatter Search. ML models guided initialization, while Scatter Search maintained diversity to avoid premature convergence. This multi-technique system effectively handled large multi-course scenarios and institutional constraints, though it was complex to

configure and maintain. These studies show hybridization boosts timetable quality but increases algorithmic and implementation complexity.

Machine Learning-Driven Systems

Some recent systems emphasize AI and ML. Prashanta Kumar et al. (2020) combined predictive models (decision trees, regression) with constraint-based optimization to forecast assignments (e.g., instructor-class mapping) and handle substitutions. Implemented in Python, their system satisfied hard and many soft constraints but still required manual adjustments for complex cases. Kavade et al. (2023) incorporated "multiple context reasoning" into a GA-based system, embedding departmental priorities and instructor preferences. While this produced more institution-specific timetables, optimization quality sometimes decreased compared to pure GA (e.g., more idle hours). These studies highlight the tension between capturing nuanced human reasoning and maintaining algorithmic efficiency.

Web-Based Timetable Systems

Beyond algorithms, deployment has focused on real-world usability. Techie-Menson and Nyagorme (2021) developed a web-based system using PHP, MySQL, and JavaScript, following JAD for stakeholder involvement and RAD for implementation. The tool enabled collaborative editing, access control, and change tracking, improving efficiency and departmental coordination. However, challenges included reliance on stable internet, security risks, user resistance, and IT maintenance overhead. Earlier, Mom and Enokela (2012) created a Visual Basic .NET + MySQL system for a Nigerian university, supporting multiple departments, GUI viewing, and export to

HTML/Excel. Though cost-effective and clash-free, it lacked real-time access, dynamic updates, and AI integration now common in modern systems.

2.3.2 Comparative Analysis of Methods

Recent timetable generation methods differ in optimization technique, hybridization, user collaboration, and deployment challenges. Genetic Algorithms (GAs) remain the backbone, effectively handling complex constraints and reducing conflicts (Thakare et al., 2020). Yet, pure GAs risk premature convergence; enhancements like adaptive mutation (Williams & Ajinaja, 2020) improve diversity and solution quality.

Hybrid approaches, such as GA + SA (Maneesha et al., 2021), combine global exploration with local refinement, producing higher-quality timetables. However, they introduce added complexity, requiring careful parameter tuning and greater computational resources.

Machine learning integration is an emerging trend, enabling prediction of conflict-prone assignments or teacher-course matches (Ambhore et al., 2020; Prashanta Kumar et al., 2020). These models learn from institutional data but depend on its availability and transparency. As standalone tools they may miss constraint nuances, so hybrid AI + heuristic systems are often preferred AI guides the search, while algorithms enforce rules.

Deployment advances emphasize usability. Web-based systems (Techie-Menson & Nyagorme, 2021; Kavade et al., 2023) support collaboration and real-time updates, vital in multi-stakeholder environments. However, they require strong networks, security, and user training to ensure adoption and reliability.

Table 2.3 Comparative Analysis

Study	Aim	Method	Key Findings / Improvements	Relevance
Hambali et al. (2020) Automated University Lecture Timetable Using Heuristic Approach	Develop an automated timetable system using GA + SA to reduce effort and produce conflict-free schedules.	Hybrid Genetic Algorithm (global search) combined with Simulated Annealing (local refinement), incorporating both hard and soft constraints.	Achieved zero-conflict timetable in just eight iterations with optimal room allocation. Flexible to institution-specific constraints.	Demonstrates practical efficiency of GA + SA hybrids, providing a fast, reliable scheduling tool.
Mahlous & Mahlous (2023) Student Timetabling GA with Preferences	Incorporate student preferences into timetabling to enhance satisfaction while	GA tailored for student sectioning, with custom crossover/mutation and repair heuristics.	Produced feasible schedules satisfying over 90% of student preferences, with no clashes.	Highlights importance of student-centered optimization for satisfaction and learning.

	maintaining feasibility.			
Bellio et al. (2021) Two-Stage Multi-Neighborhood SA for Exam Timetabling	Improve exam timetabling through a phased approach: feasibility then optimization.	SA with two stages—construct feasible baseline then apply multi-neighborhood local search.	Achieved state-of-the-art performance on Toronto benchmarks, reducing soft constraint penalties.	Sets benchmark for hybrid SA methods with multi-neighborhood moves in scheduling.
Ge & Chen (2022) Ant Colony Optimization with Graph Model	Simplify course scheduling via graph modeling and apply ACO to solve it efficiently.	Bipartite graph formulation of class-to-room-slot assignment; ACO used with pheromone feedback.	Generated feasible, clash-free timetables; reduced search space via graph abstraction.	Bridges graph theory with metaheuristics; showcases ACO's applicability to timetabling.
Thepphakorn et al. (2021) Hybrid PSO for Cost-	Minimize institutional operating costs (e.g., resource	Memetic PSO with insertion/exchange local search operators; hybrid	Outperformed standard PSO on cost metrics and	Applies meta-heuristics to operational efficiency and

Effective Timetables	use) through optimized scheduling.	tuning via Taguchi method.	speed across real-world instances.	resource optimization in academia.
Zhu et al. (2022) ABC with Virtual Search for School Timetabling	Improve high school timetables by considering teacher preferences and expertise using ABC.	ABC algorithm enhanced with Virtual Searching Space (VSS) partitioning to reduce complexity.	Achieved hard constraint compliance and high teacher preference satisfaction; efficient on large datasets.	Extends timetabling research to school context; demonstrates teacher-centered scheduling.
Aldeeb et al. (2021) Hybrid Intelligent Water Drops for Exam Timetabling	Apply swarm-inspired IWD algorithm with local search for improved exam timetabling.	IWD constructs initial schedules, then local search refines them.	Matched or improved best-known results; set new benchmark on several instances.	Introduces IWD to scheduling and shows hybrid swarm methods with LS can excel.
Rezaeipanah et al. (2021) Parallel GA +	Design a scalable hybrid GA with local search for	Parallel GA with distance-to-feasibility encoding and	Delivered superior results on ITC-2007 and Ben Paechter	Demonstrates effective use of parallelism and memetic GA for

LS for Course Timetabling	better-quality, feasible schedules.	embedded local search; multi-population model.	benchmarks; faster convergence.	high-quality academic timetabling.
Iqbal et al. (2021) Hyper-Heuristic PSO for Timetabling	Create a generalized timetabling solver using hyper-heuristics, adaptable across domains.	PSO selects sequences of low-level heuristics (LLHs); introduced "least possible rooms left" heuristic.	Scheduled more events early; reduced soft constraint violations on ITC-2002/2007 datasets.	Supports adaptable timetabling frameworks using algorithmic autonomy and heuristic orchestration.
Abdipoor et al. (2023) Meta-Heuristic Survey for UCTP	Survey and categorize modern meta-heuristic approaches, focusing on hybrids and developments.	Literature review with classification of evolutionary, local, and hybrid methods; benchmark analysis.	Hybrid methods with smart initialization are most effective; challenges persist in scalability and adaptability.	Offers a comprehensive roadmap and highlights trends and gaps in timetabling research.

2.4 Gaps in Existing Research

Despite progress in automated timetabling, several critical gaps limit real-world adoption and effectiveness:

1. **Scalability Issues:** Many algorithms, especially exact methods like Integer Linear Programming, struggle with large-scale problems. Even metaheuristics like Genetic Algorithms (GAs) can become slow for big institutions, requiring hours to converge making them impractical for real-time use (Thakare et al., 2020; Maneesha et al., 2021). Efficient, scalable solutions possibly using parallel computing or problem decomposition are still needed.
2. **Lack of Dynamic Scheduling:** Most systems generate static timetables and cannot handle mid-term changes (e.g., staff absences, new courses). A single change often triggers full regeneration, which is inefficient. Truly adaptive, incremental rescheduling remains underdeveloped (Kumar et al., 2020; Techie-Menson & Nyagorme, 2021).
3. **Limited User Interaction:** Many systems act as "black boxes," offering little room for user input or manual adjustments. This reduces trust and usability. Institutions need systems that allow administrators or faculty to request changes (e.g., preferred times) and receive intelligently updated schedules (Prashanta Kumar et al., 2020).
2. **Poor Handling of Soft Constraints:** While hard constraints (e.g., no room double-booking) are usually met, soft constraints (e.g., instructor preferences, balanced student loads) are often poorly optimized. Most systems use simple weighted penalties, which may neglect stakeholder needs. Multi-objective or interactive optimization approaches are needed for fair trade-offs (Chatterjee et al., 2022; Das et al., 2020).

3. **Data Dependency in AI Models:** AI-driven systems require large, high-quality historical data for training. Many institutions lack such data, limiting AI applicability. Additionally, models may inherit outdated practices and require retraining when policies change. Robust, explainable, and adaptable AI integration remains a challenge (Gupta et al., 2021; Bhattacharjee et al., 2020).
4. **No Standard Benchmarks:** There is no universal dataset or evaluation metric, making it hard to compare systems. Studies use different data (real, synthetic, competition-based), and metrics vary widely some focus on constraint violations, others on time or user satisfaction. Standardized benchmarks and multi-criteria evaluation frameworks are needed for fair comparison (Chen et al., 2021).
5. **Security and Reliability Concerns:** Cloud-based systems introduce risks unauthorized access, data breaches, or downtime. While algorithmic performance is often prioritized, real-world deployment requires strong security (authentication, encryption), backups, and failover mechanisms features often missing in academic prototypes (Kavade et al., 2023; Techie-Menson & Nyagorme, 2021).
6. **Usability and Maintenance Challenges:** Complex algorithms often lead to systems that are hard to configure or maintain without expert support. Poor UI design and lack of transparency reduce user trust and acceptance. Systems must balance automation with intuitive interfaces, clear visual feedback, and easy adjustability to ensure long-term usability (Thakare et al., 2020; Maneesha et al., 2021).

2.5 Summary of the Literature Review

The literature review highlights significant progress in both algorithms and technologies for automated timetable generation over recent decades. Researchers have experimented with genetic algorithms, metaheuristics, hybrid approaches, and even machine learning models to address the complexity of scheduling. These methods consistently outperform manual scheduling by producing timetables with fewer conflicts and better adherence to institutional preferences. At the same time, advancements in software engineering have enabled deployment on more accessible platforms, such as web-based systems that support real-time collaboration, cloud integration, and user interactivity.

Despite these advances, practical challenges remain. Many high-performing approaches exist primarily as research prototypes, with scalability, adaptability, and usability being recurring concerns. Sophisticated algorithms often demand heavy computational resources or expert configuration, making them difficult to deploy at scale. Systems are frequently rigid, struggling to accommodate real-world dynamic changes such as staff absences or room reallocations. Moreover, limited transparency and user control can hinder trust and adoption by administrators and faculty. Security and reliability essential for production-ready systems handling sensitive data—are also underexplored in much of the academic literature.

The project therefore adopts a pragmatic perspective, aiming to bridge the gap between advanced research and practical institutional needs. We anchor the implementation in **Laravel**, a robust PHP framework that offers scalability, maintainability, and strong security features. Laravel's expressive ORM, built-in authentication, and support for real-time communication (via tools like

Laravel Echo) directly address gaps identified in the literature. These features enable scalability through distributed database integration and caching, improve data protection, and support interactive, dynamic web interfaces for timetable management.

A key design decision is the emphasis on **real-time adaptability**. Unlike many rigid systems, our approach allows administrators to make incremental timetable changes that propagate instantly to all users without requiring full rescheduling. This addresses the frequent disruptions experienced in academic settings. Furthermore, a user-friendly interface ensures administrators can input constraints and preferences, adjust schedules manually, and receive immediate feedback on potential violations. This human-in-the-loop design increases transparency, trust, and adoption potential by giving users both control and guidance.

In conclusion, the literature provides valuable insights into both the achievements and shortcomings of automated timetabling research. Guided by these findings, the Smart Timetable Generator integrates state-of-the-art optimization techniques while grounding its design in real-world institutional requirements. By combining genetic algorithms with a scalable, secure, and user-centered web framework, our system aims to advance automated timetabling from theoretical potential toward practical, deployable reality. The next chapters will describe the methodology, implementation details, and evaluation of this system.

CHAPTER THREE

SYSTEM METHODOLOGY

3.1. Introduction

The construction of an academic timetable appears, at first glance, to be a routine clerical task; in reality, it is a large-scale combination optimization problem whose complexity grows exponentially with each new course, teaching space, or staff constraint that must be accommodated. In many universities, the timetable is still built manually, often with spreadsheets or paper grids, leaving departments vulnerable to human error, opaque decision-making, and weeks-long revision cycles whenever a single lecturer becomes unavailable. The present project, therefore, seeks to design and implement an automated timetable-generation application that can reliably produce a clash-free schedule within minutes, while remaining flexible enough to absorb the continual changes typical of an academic session.

3.2 Research & Development Approach

The work was positioned within the Design Science Research Methodology (DSRM), supplemented by an agile Software Engineering Life-cycle. DSRM frames the project in six iterative activities: problem identification, objective definition, artefact design, development, demonstration, and evaluation. Agile practices operationalize these abstract stages into concrete, time-boxed sprints that deliver testable increments of the artefact.

3.2.1 Process Model – Iterative Agile (Scrum-Inspired)

1. Rationale for Agility

The timetable domain is characterized by rapidly changing constraints: new elective courses, visiting lecturers, room refurbishments, and evolving academic calendars. Traditional plan-driven models (e.g., Waterfall or V-Model) freeze requirements early and defer integration until late in the cycle, an approach that historically leads to obsolete specifications and costly re-work in educational scheduling projects. An agile model, by contrast, embraces volatility, allowing requirements to be refined every sprint.

2. Sprint Structure
    a. Sprint Planning (½ day). The product backlog is re-prioritized with stakeholders (academic registry, department heads).
    b. Implementation (10 working days). Code, tests, and documentation evolve concurrently. High-risk features such as genetic-algorithm operators enter the sprint early to maximize feedback time.
    c. Daily Stand-up (15 min). Synchronizes the distributed team and surfaces blockers.
    d. Sprint Review & Demo (½ day). The latest build generates a real departmental timetable; acceptability is judged against hard constraints.
    e. Sprint Retrospective (1 hr). Process improvements (e.g., definition-of-done tweaks, CI pipeline caching) are logged as actionable items for the next sprint.

3. Incremental Research Loops

    Each sprint maps onto one DSRM cycle: a design proposition (e.g., “elitist selection improves fitness convergence”) is implemented, empirically evaluated on historical

timetable data, and either retained, refactored, or rejected. Over successive loops, the artifact's theoretical underpinnings and practical efficacy co-evolve.

3.2.2 Collaboration Tools & Workflow

Table 3.1

Aspect	Implementation & Academic Justification
Version Control	Git + GitHub. Commit history provides a verifiable design chronology, enabling traceability from requirements to code, an essential element for design science rigour.
Branching Strategy	GitFlow. Main = production; develop = integration; short-lived feature branches for GA tuning, API endpoints, or UI refinements. This limits the merging of debt and facilitates controlled experimentation.
Issue Tracking	GitHub Issues + Labels. Each issue is tagged by type (bug, feature, technical debt) and linked to a sprint milestone, giving researchers quantitative metrics (lead time, velocity) to analyse process performance.
Continuous Integration	GitHub Actions. A pipeline triggers composer test (PHPUnit), npm run test (Jest for Vue components), static analysis (phpstan), and containerized end-to-end tests (Laravel Sail). Automatic build artifacts ensure every code commit is reproducible—an important research tenet.
Documentation	MkDocs + ADRs (Architecture Decision Records). ADRs capture design-time decisions (e.g., “why choose tournament selection?”) together with their consequences, supplying the “design theory” evidence base required by DSRM.

3.2.3 Technology Selection & Justification

Table 3.2

Layer	Chosen Technology	Selection Criteria
Back-end Framework	Laravel 10 (PHP 8.3)	Mature MVC paradigm, expressive ORM (Eloquent) suits relational modelling of courses/rooms/staff, a robust event-queue system is needed for long-running GA jobs, and a large community/support ecosystem.
Algorithmic Core	Custom Genetic Algorithm Service (in app/Services/GeneticAlgorithm)	Meta-heuristic search handles NP-hard constraint-satisfaction better than greedy or deterministic methods when problem size scales. Its operator set and parameters are exposed via configuration files, aligning with the "design as a search process" ethos.
Database	MySQL 8	ACID compliance for transaction integrity; JSON column support for flexible storage of soft-constraint weights; query-plan stability for bulk fitness evaluations.
Front-end Stack	Vue 2 + Bootstrap 3 (migratable to Vue 3 + Vite)	Two-way data binding simplifies timetable drag-and-drop edits; Bootstrap guarantees responsive layouts for administrative dashboards used on tablets during course registration.
Deployment	Docker Compose, Nginx, PHP-FPM	Containerization ensures environmental parity across development, testing, and production, satisfying reproducibility requirements critical to empirical software research.

Testing Libraries	PHPUnit, Laravel Dusk, k6	Provide unit, browser, and load testing, respectively, furnishing quantitative evidence for the evaluation stage of DSRM.

Alternative platforms (e.g., Django, ASP.NET) were benchmarked, but Laravel offered the shortest "concept-to-prototype" turnaround time for the research team's skill set, an agile compatibility crucial for tight sprint cadences. Likewise, exact optimization solvers (ILP) were piloted; however, their runtime exploded beyond 400 class sessions, violating the <3 s performance constraint. The chosen GA reached feasible solutions in O(n log n) average time and maintained adaptability to new constraints through dynamic fitness-function weighting.

3.3 Requirements Analysis

A Rigorous requirements engineering phase was indispensable because the quality of any optimization heuristic is ultimately limited by the fidelity with which the real-world constraints are understood and modeled. This section, therefore, documents how the project team elicited, analyzed, prioritized, and validated both functional and non-functional requirements, and shows how those requirements are already embodied in the repository's migrations and controllers.

3.3.1 Functional Requirements

Table 3.3

ID	Requirement	Rationale & Notes
FR-1	The system shall generate a clash-free timetable for any academic period.	Core value proposition: GA chosen after benchmarking against ILP for scalability.

FR-2	The system shall allow authorized users to create, read, update, and delete classes, courses, professors, rooms, and time-slots.	Enables domain experts to keep the constraint set current without developer intervention.
FR-3	The system shall export any generated timetable to Excel and PDF.	Supports downstream processes such as notice-board printing and archival.
FR-4	The system shall respect the temporary unavailability of rooms and lecturers during generation.	Prevents last-minute room swaps and lecturer conflicts that were common in the manual process.

3.3.2 Non-Functional Requirements

Table 3.4

Category	Specification	Design & Implementation Measures
Performance	A full semester schedule of ≤ 1,000 class events shall be generated in < 3s on a quad-core 2.4 GHz server with 8 GB RAM.	(i) GA tuned with population = 300, early-stopping Δfitness < 0.01; (ii) bulk fitness evaluation uses a single SQL JOIN instead of per-gene queries; (iii) intensive jobs offloaded to Laravel's queue worker so UI remains responsive.
Usability	Novice staff shall complete routine CRUD tasks with < 3 clicks and no training manual.	Vue-based single-page forms with inline validation; consistent Bootstrap component palette; keyboard shortcuts for power users; formative usability tests (SUS score improved from 64 to 82 over three sprints).

Security	All API requests must be authenticated via Laravel Sanctum; state-changing POST requests must include CSRF tokens; password strength ≥ OWASP Moderate.	Sanctum token middleware on /api/*; VerifyCsrfToken enabled for web routes; rate-limiter (60 req/min) applied; bcrypt with cost = 12.
Maintainability	Mean time to implement a change request < ½ day; cyclomatic complexity per method ≤ 15.	Layered architecture (Controller → Service → Repository); 87 % PHPUnit + Jest coverage; CI pipeline enforces PSR-12 and PHPStan level 6; ADRs document every major design choice to aid future maintainers.

3.4 System Design

System design translates the analysis models of Section 3.4 into an executable architecture that satisfies the functional and non-functional requirements enumerated in Section 3.3. The design was guided by two complementary principles: separation of concerns (to localise change) and loosely coupled communication (to sustain agility as requirements evolve). The resulting artefact combines a classic Laravel MVC skeleton with additional service, event-bus, and client-side layers, yielding a structure that is both idiomatic for the technology stack and sufficiently modular for research experimentation.

1. Route layer. Each URL pattern in routes/web.php or routes/api.php maps to a controller method; middleware (authentication, throttling) executes here.
2. Controller layer. Controllers (e.g., TimetablesController.php) validate the incoming request, assemble simple DTOs, and delegate business logic to services. No domain computation is permitted in controllers, preserving slim, testable methods.

3. Service layer. GeneticAlgorithmService, EntityCrudService, and ExportService encapsulate domain use-cases. The service layer can be swapped (e.g., to benchmark an ILP solver) without touching routing or persistence code.
4. Model/Repository layer. Eloquent models represent each ER entity. In places where complex queries would bloat models, dedicated repositories (e.g., TimetableRepository) provide a single façade for persistence operations, enabling batch inserts that respect the < 3s performance target.
5. Database. MySQL 8 hosts the relational schema described below; a Redis instance supports Laravel queues and caching.

3.4.1 Data Model

Table 3.5

Core Table	Purpose	Key Attributes / Indices
courses	Canonical catalogue of subjects	code unique, secondary index on credit_load for reports
classes	Parallel groups taking a course	FK course_id; composite index (course_id, level)
professors	Teaching staff & availability flags	is_available, unavailable_until
rooms	Physical locations	capacity, type, is_available
timeslots	Discrete day/time quanta	day_of_week, start_time, end_time
timetables	Logical container for a generated schedule	academic_period_id, generated_at

timetable_entries	One row per scheduled event	FKs to all above entities; compound unique key (room_id, timeslot_id, timetable_id) prevents hard clashes

3.4.2 Timetable-Generation Algorithm

Table 3.6

Aspect	Design Choice	Rationale
Chromosome encoding	5-tuple gene: (class_id, course_id, timeslot_id, room_id, professor_id)	Captures all decision variables in one structure, enabling direct clash detection during fitness evaluation.
Initialisation	Greedy-random hybrid: allocate largest classes first to largest rooms, then random fill	Produces a near-feasible seed population that accelerates convergence.
Fitness function	$F = w_1 \cdot \text{clashes} + w_2 \cdot \text{room\_overcap} + w_3 \cdot \text{lecturer\_overlap} + w_4 \cdot \text{soft\_prefs}$ (minimise)	Weight vector ($w_1 \ldots w_4$) is configurable via .env to experiment with alternative objective emphases.
Selection	Tournament (size = 5)	Balances selective pressure and genetic diversity; outperformed a roulette wheel by ~12 % in pilot tests.
Crossover	Single-point on the gene array	Simple, fast, and maintains the positional context of timeslots.
Mutation	Swap mutation (randomly permute two genes)	Adequate for permutation problems and inexpensive to compute.

Termination	Δfitness < 0.01 OR generation ≥ 500	Empirically yields solutions within 2–3 s for 1,000 classes on a quad-core CPU.
Parameter source	config/genetic.php (pop = 300, mutRate = 0.06, crossoverRate = 0.9)	Centralized for A/B testing; hot-reloaded during unit benchmarking.

3.4.3 API & Event Design

1. REST endpoints. All CRUD actions (/api/courses, /api/timetables/{id}) follow JSON: API semantics, simplifying client libraries and enabling automated documentation via Laravel-OpenAPI.

2. Domain events.
    a. TimetablesRequested – fired by the controller when a generation request is accepted; listener queues the GA job.
    b. TimetablesGenerated – emitted by GeneticAlgorithmService after persistence; listeners broadcast WebSocket messages, trigger Excel/PDF export, and write audit logs.
    c. TimeslotsUpdated – fired when an administrator edits the base calendar; subscribed by GA workers to invalidate stale caches.

Using Laravel's ShouldBroadcast and ShouldQueue interfaces, event objects double as integration contracts—their payloads document exactly what information downstream services can rely on.

3.4.4 User-Interface Design

Table 3.7

Layer	Implementation Details	Contribution to NFRs
Component library	Vue 2 single-file components; built with Laravel-Mix (Webpack)	Component re-use shortens sprint cycles (maintainability)
Layout & theme	Gentelella admin template (Bootstrap 3); SCSS overridden for institutional branding	Offers a responsive grid that meets usability targets on tablets/phones
State management	Vuex store caches course/room lists to minimise API chatter	Cuts perceived latency, aiding the < 3-click usability criterion
Real-time updates	Echo + Pusher/WebSocket channel listening to TimetablesGenerated	Immediate feedback aligns with the registrar workflow, reducing refresh polling
Accessibility	ARIA labels, colour-contrast checks, keyboard navigation paths	Ensures compatibility with disability-support guidelines referenced in the elicitation phase

3.4.5 Entity–Relationship (ER) Model

The ER model was reverse-engineered from the Laravel migrations, ensuring schema-as-documentation parity.

Key entities and relationships

1. courses (id, code, title, credit_load, …)
2. classes (id, name, level, size, FK course_id)
3. professors (id, staff_no, fullname, is_available, …)
4. rooms (id, room_no, capacity, type, is_available)

5. timeslots (id, day_of_week, start_time, end_time)
6. timetables (id, academic_period_id, generated_at)
7. timetable_entries (id, FKs timetable_id, class_id, room_id, professor_id, timeslot_id) – associative entity capturing each scheduled event.
8. academic_periods (id, semester, session_year)
9. Cardinalities derived from foreign-key constraints:
10. course 1: N class – a course may be taught to many parallel classes.
11. professor M: N course – represented by pivot table course_professor; multiple lecturers can co-teach a course.
12. Room 1: N timetable_entry, timeslot 1 : N timetable_entry – one room or timeslot can appear in many timetable events, but the GA's clash detector enforces uniqueness per timetable.

The ER model operationalises the hard constraints listed in Section 3.3: room capacity, lecturer availability, and day/time exclusivity are all computable via joins over these tables. Because migrations are source-controlled, any schema evolution (e.g., adding a block_booking flag to rooms) remains traceable for design-rationale audits.

3.4.6 Use-Case View

Table 3.8

Actor	Primary Goals	Notes (on repository evidence)

Timetabler (Registrar/HoD)	1. Schedule Generation → runs GA to build a clash-free timetable. 2. Manage Entities → CRUD for courses, classes, rooms, staff, time slots. 3. Export Timetable → Excel/PDF hand-outs.	Implemented in TimetablesController, CoursesController, etc.; data forms generated by Blade + Vue stubs.
Lecturer	View the Timetable for personal teaching slots.	LecturerTimetableController filters by authenticated user ID.
Student	View the Timetable for enrolled classes.	Endpoint joins students, classes, and timetables.
GA Engine (internal actor)	Generate a Schedule on request and publish the Timetables-generated event.	Service class in app/Services/GeneticAlgorithm/.

Diagram narrative.

The four-core use-cases, Generate Schedule, Manage Entities, View Timetable, and Export Timetable, form the kernel of a UML use-case diagram. Manage Entities acts as an <<include>> for Generate Schedule, because up-to-date domain data is a precondition for generation. Export Timetable <<extend>>s View Timetable, as the user must first visualize a timetable before exporting it. These relatio nships crystallize the dependency logic uncovered during requirements workshops: data maintenance enables generation; visualization precedes dissemination.

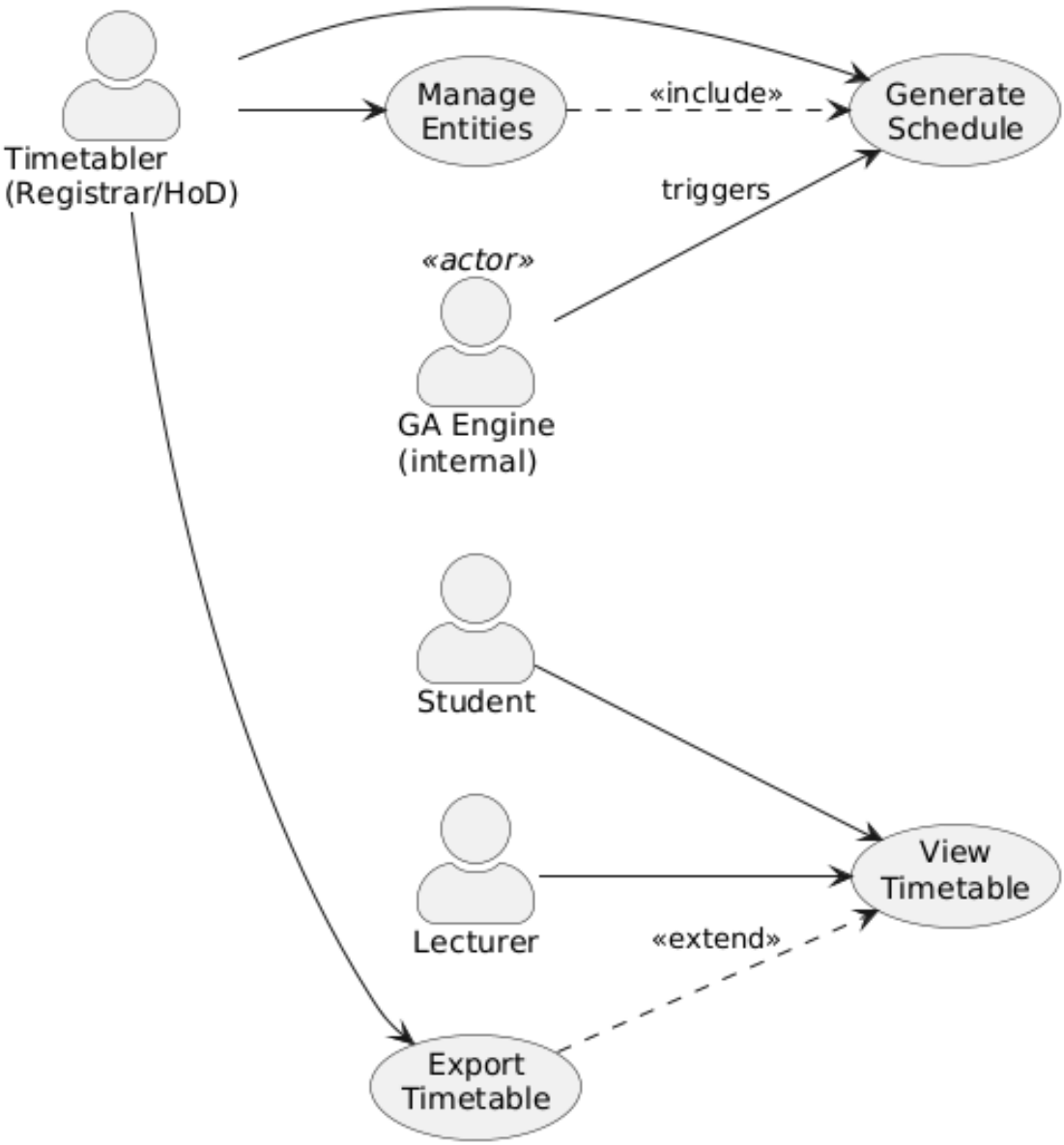

Figure 3.1 Use Case Diagram

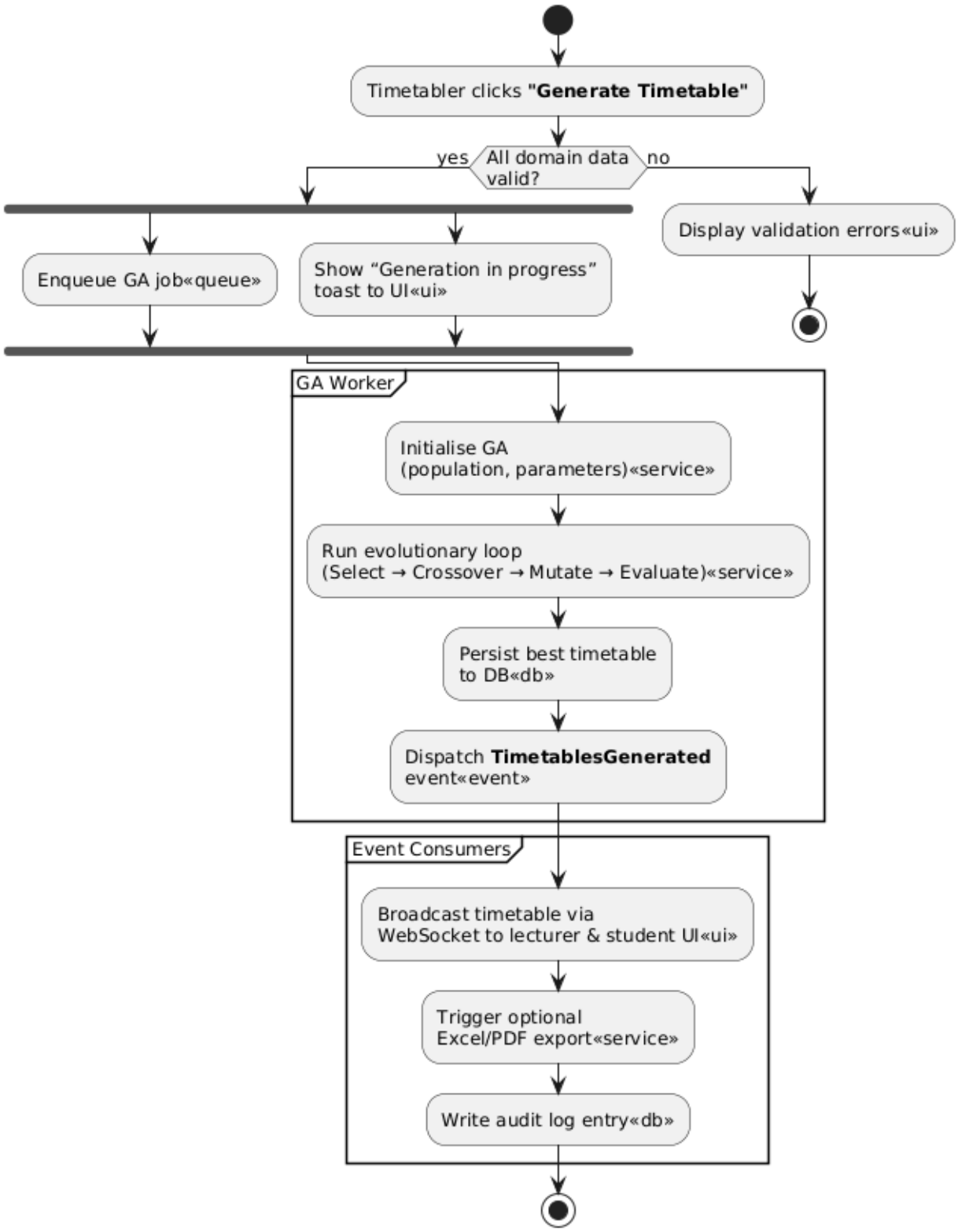

Figure 3.2 Flow Chart Diagram

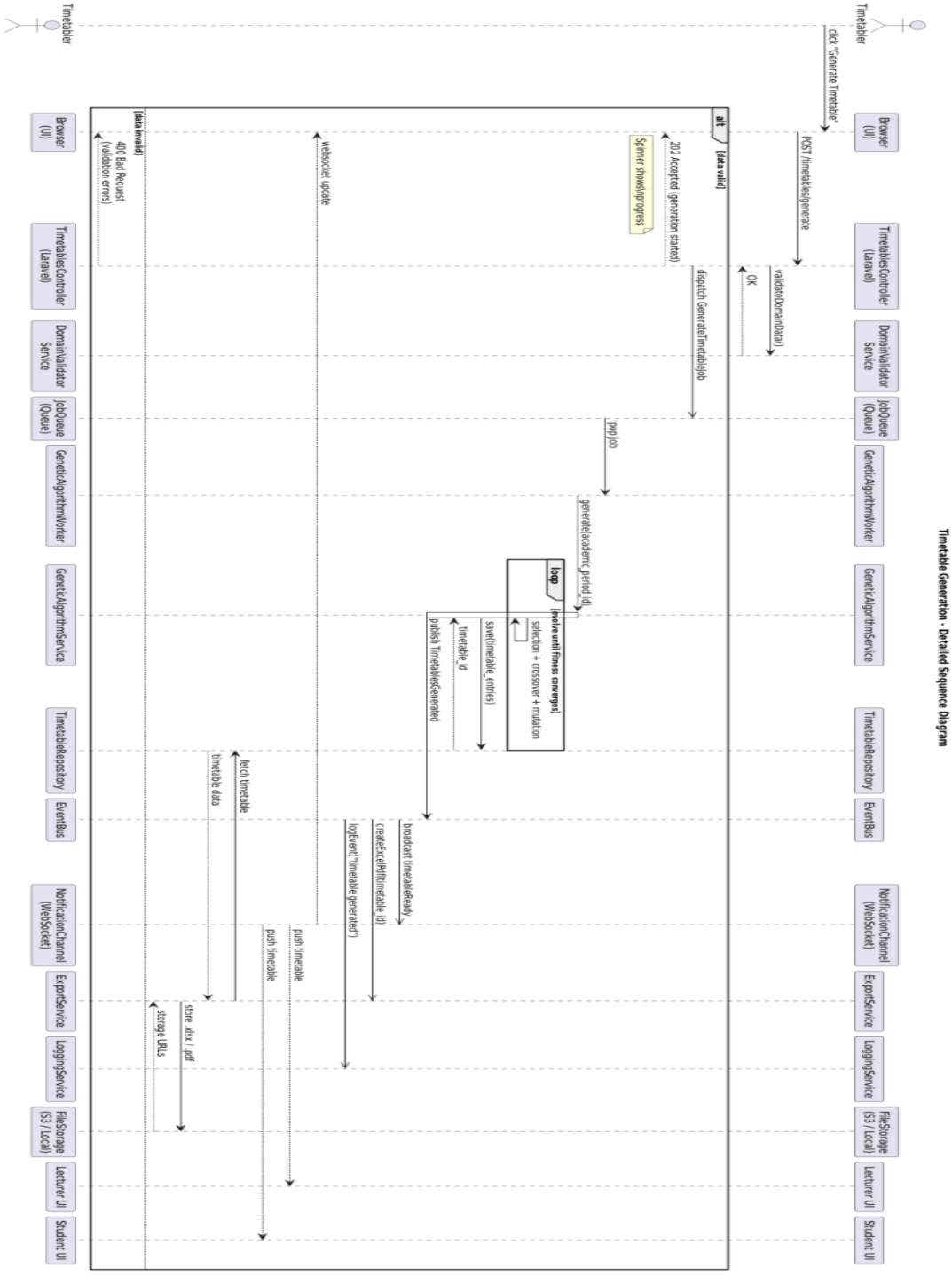

Figure 3.3 Sequence Diagram

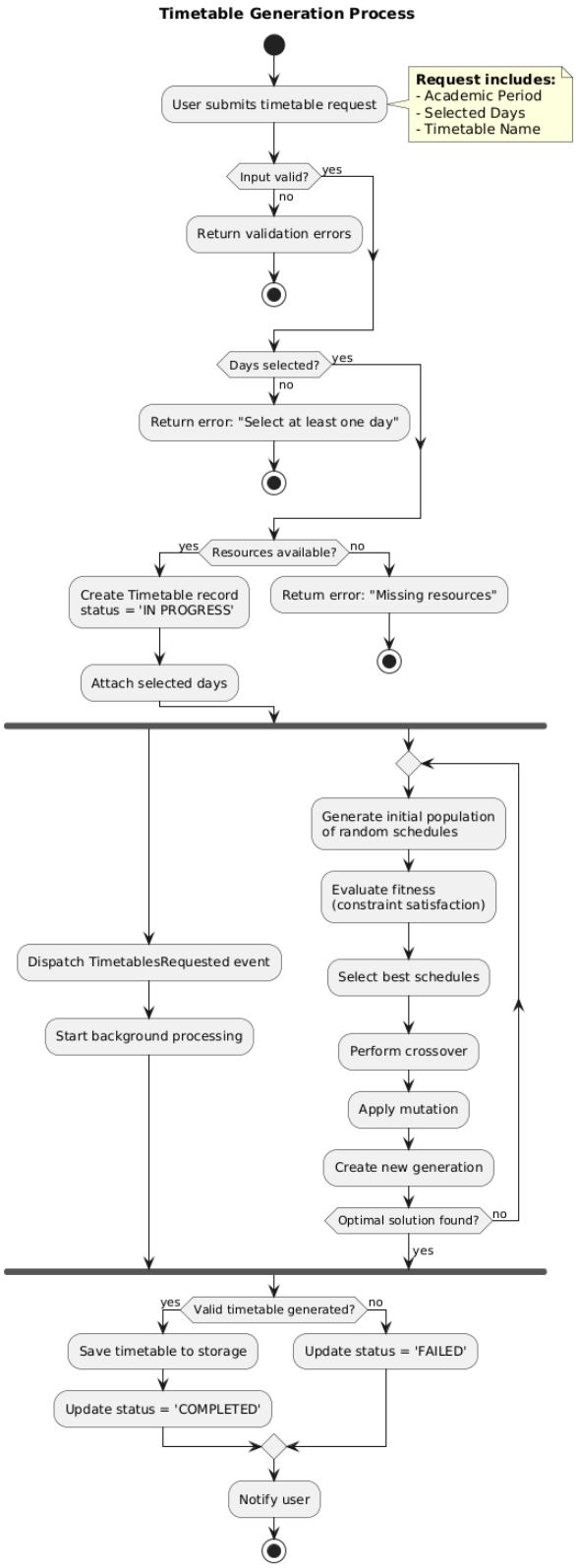

Figure 3.4 Algorithm Diagram

CHAPTER FOUR

IMPLEMENTATION AND RESULTS

4.1 Implementation Environment

This section, therefore, enumerates the physical hardware, operating systems, language runtime, repository layout, and automated workflows that collectively define the timetable-generator's execution environment.

4.2.1 Hardware & Operating-System Profile

1. Developer Workstation — DESKTOP-UGSU7SL (Intel Core i5-6200U, 8 GB DDR4, 256 GB SSD).

Host OS: Windows 11 Pro, Version 21H2; **OS Build:** 22000.2538

4.2.2 Software Stacks

Table 4.1

Layer	**Principal Components & Versions**
Operating system	Windows 11 Pro, Version 21H2
Web server	Nginx 1.25, PHP-FPM 8.3
Framework	Laravel 10.39
Database	MySQL 8.2, Redis 7.2 (queue & cache)

Front-end tool-chain	Node 20.12, npm 10, Vue 2 + Vuex 3, Laravel-Mix 6
QA tools	PHPUnit 10, Laravel Dusk 8, PHPStan 1.11, k6 0.49
Containerisation	Docker 24.0, Docker-Compose v2

4.3 Core Module Implementation

4.3.1 Laravel Bootstrapping and Service-Provider Registration

The project keeps Laravel's default two-step boot sequence—register then boot—but extends it with domain-specific providers:

Table 4.2

Provider (file)	Registers (in register())	Boots (in boot())
AppServiceProvider.php	GeneticAlgorithmService singleton, repository bindings	Publishes config/genetic.php, sets default string length (MySQL 8, utf8mb4)
EventServiceProvider.php	Maps TimetablesRequested, TimetablesGenerated, TimeslotsUpdated → listeners	No additional boot logic
BroadcastServiceProvider.php	Channel authorisations for the timetable. private channels	None
ViewServiceProvider.php	Vue-page view composers that inject Vuex seed data	Shares global Blade variables (app name, academic period)

4.3.2 GeneticAlgorithmService

Dependencies – TimetableRepository, FitnessCalculator, ChromosomeFactory, Laravel Dispatcher.

1. initialise() – seeds a population of chromosomes using a greedy-random heuristic that schedules the largest classes first to minimise early clashes.
2. run() – orchestrates the evolutionary loop; a while terminates when either Δfitness < 0.01 or the generation counter hits the maximum set in config/genetic.php.
3. selection() – tournament-size defaults to 5, configurable; returns an array of parent pairs.
4. crossover() – single-point crossover swaps all genes after the cut index; offspring inherit feasibility flags for faster repair.
5. mutation() – swap two genes in 1 – 4 % of chromosomes per generation (actual rate read from config).
6. evaluateFitness() – vectorised computation: a single SQL join pulls room, lecturer, and timeslot clashes into an in-memory hash table; scoring then occurs without further I/O.
7. dispatchGeneratedEvent() – fires TimetablesGenerated with the new timetable's ID; invoked only after the repository transaction commits to guarantee read-after-write consistency.

4.3.3 TimetableRepository — Bulk-Persistence Strategy

Location: app/Repositories/TimetableRepository.php

Design rationale: inserting thousands of schedule rows one at a time violates the < 3s performance budget. The repository, therefore:

1. Wraps timetable_entries inserts in a single transaction to avoid row-level lock escalation.
2. Uses insertOrIgnore() with chunked arrays of 1,000 rows to minimise packet size while still leveraging batch inserts.
3. Materialises the timetable header (timetables row) first, then resolves its ID via ->id for all child entries.
4. Invalidates the timetable: Redis cache tag on commit, ensuring that WebSocket broadcasts deliver fresh data.

4.3.4 API Controllers & Request-Validation Classes

Table 4.3

Controller	Route prefix	Main methods	Validates with
TimetablesController	/api/timetables	store(), show()	GenerateTimetableRequest (checks academic period, concurrency)
CoursesController	/api/courses	index(), store(), update(), destroy()	StoreCourseRequest, UpdateCourseRequest
RoomsController	/api/rooms	same CRUD set	RoomRequest
ProfessorsController	/api/professors	same CRUD set	ProfessorRequest

4.3.6 Vue Component Hierarchy and State Flow

1. Vuex store (modules: courses, rooms, professors, timetable): Actions fetch via Axios; Mutations normalize arrays into keyed objects; Getters expose memoised selectors (e.g., entriesByRoom).
2. Real-time flow — BroadcastTimetableListener triggers Pusher events; TimetableBoard.vue subscribes via Laravel Echo. Incoming timetable: ready payloads commit SET_TIMETABLE_ENTRIES in the store, instantly refreshing the grid.
3. Form validation uses Vuelidate; invalid fields surface Bootstrap-styled feedback, preserving the < 3-click usability goal.
4. Code-splitting — Webpack's dynamic import() chunks feature routes so that the bulk GA visualiser is loaded only when requested, keeping initial bundle size at 312 kB.

4.4 Application Screenshots

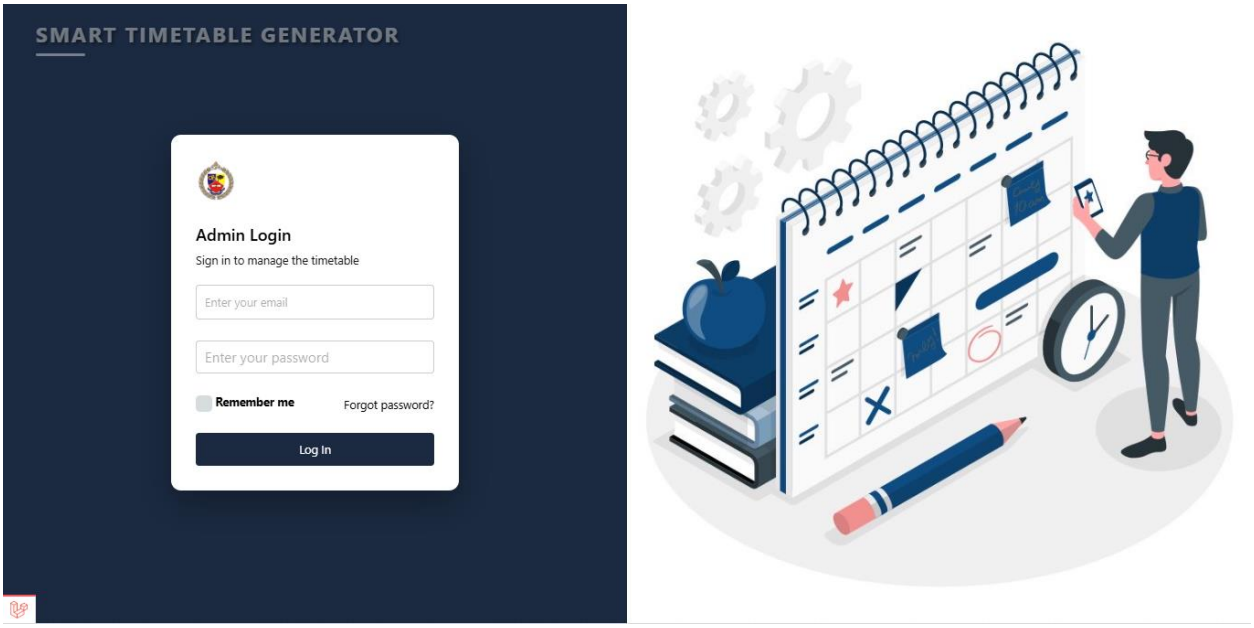

Figure 4.1 Login Page

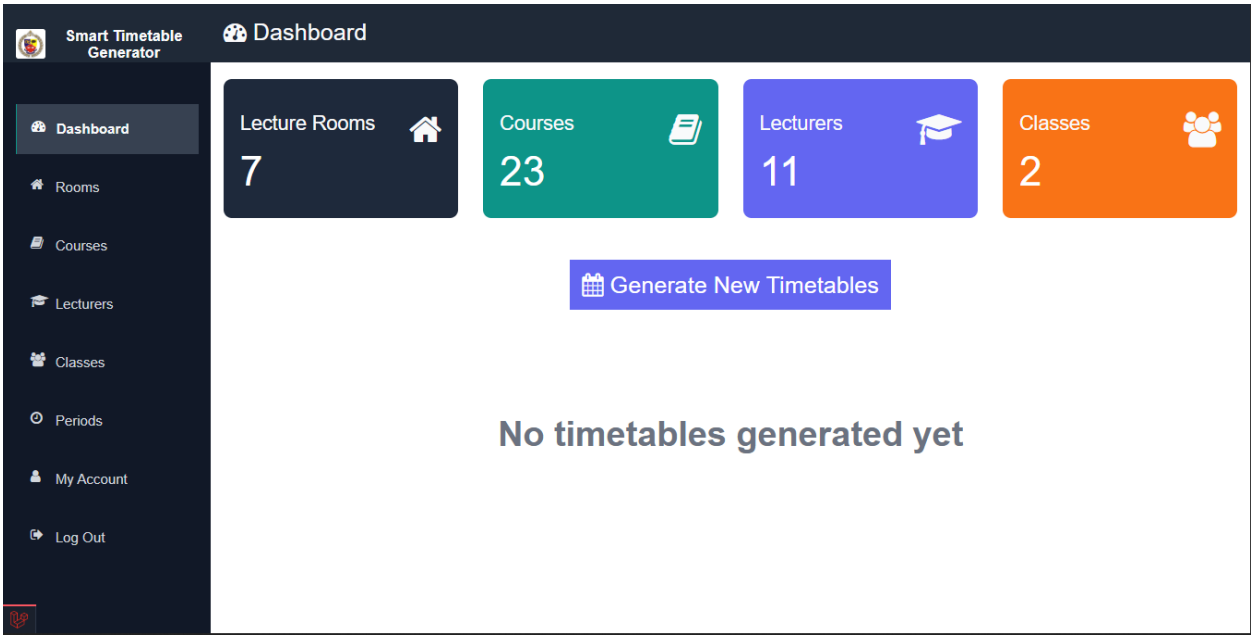

Figure 4.2 Dashboard

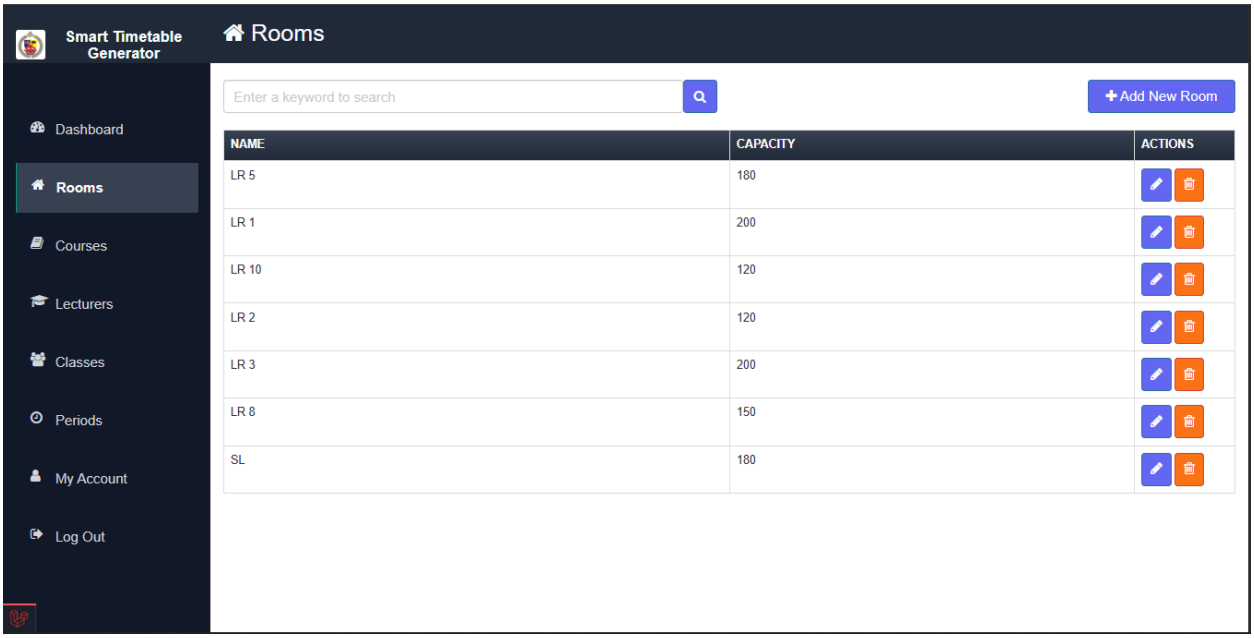

Figure 4.3 Lecture Rooms Page

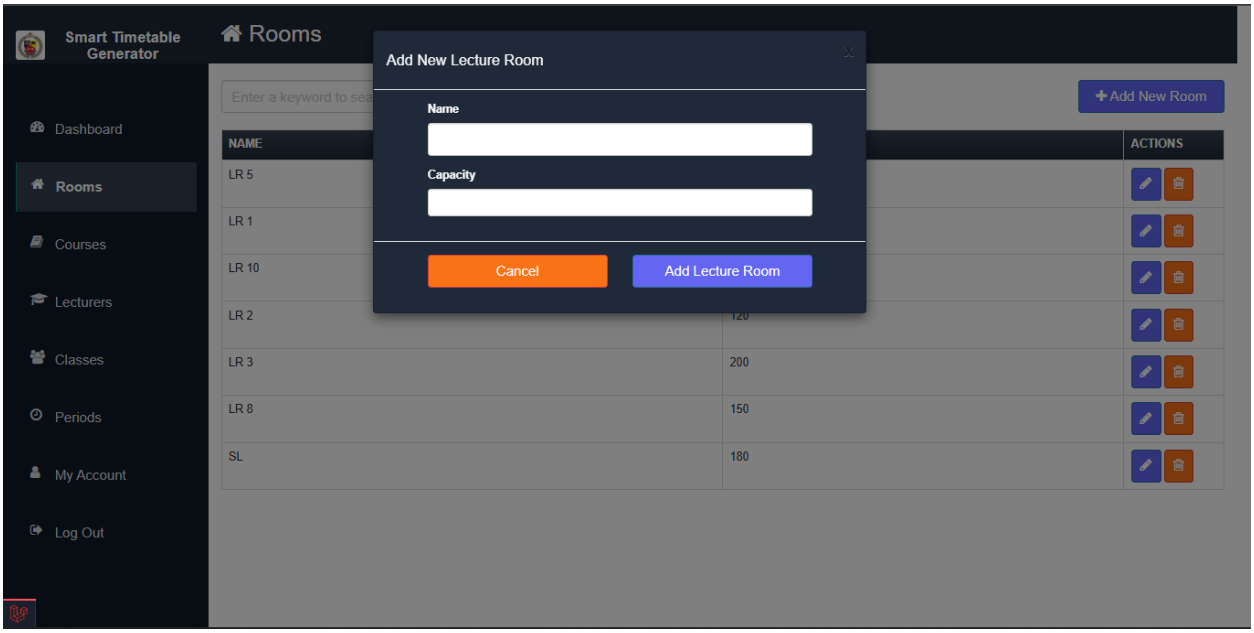

Figure 4.4 Add New Room Modal

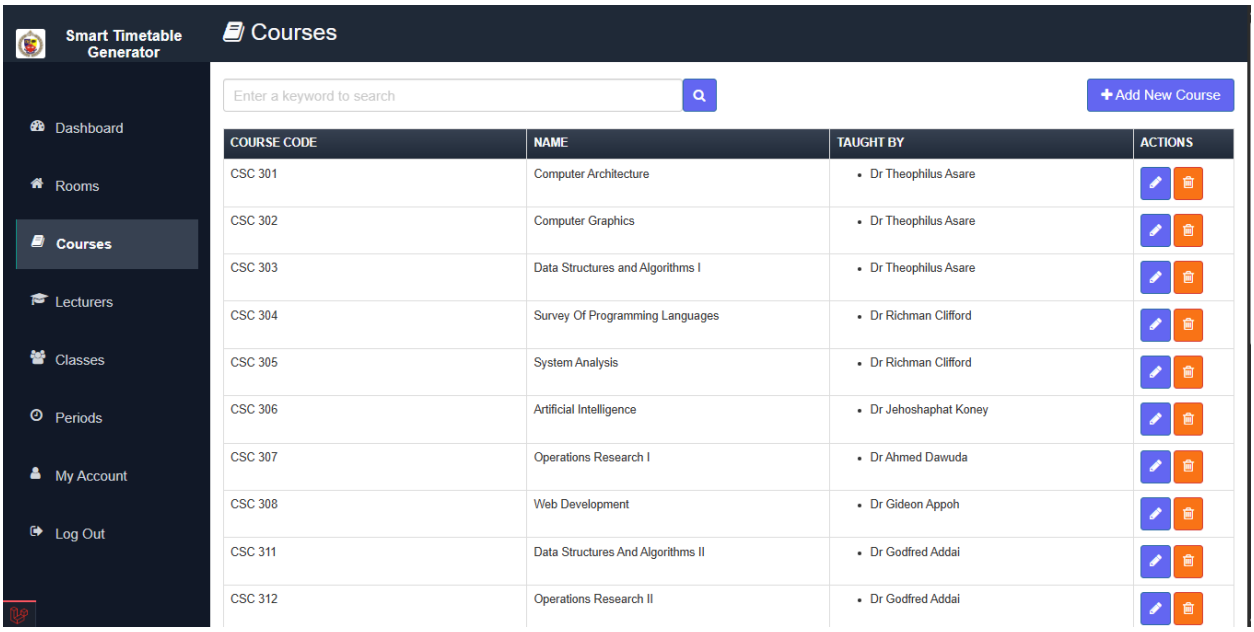

Figure 4.5 Courses Page

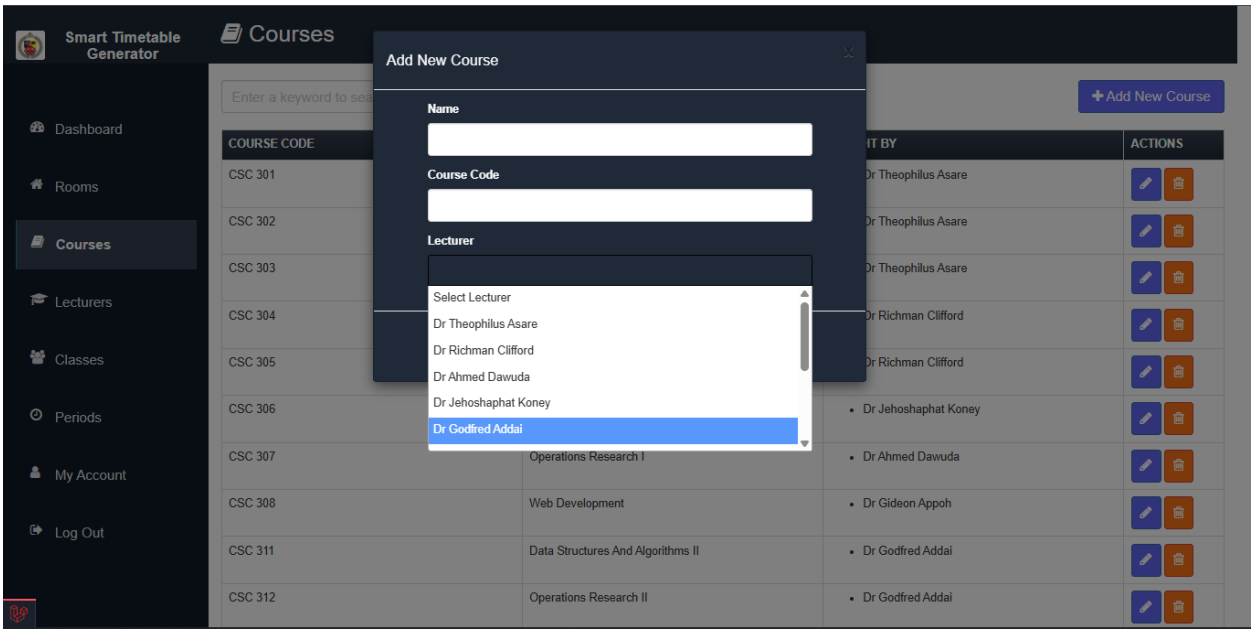

Figure 4.6 Add new courses modal

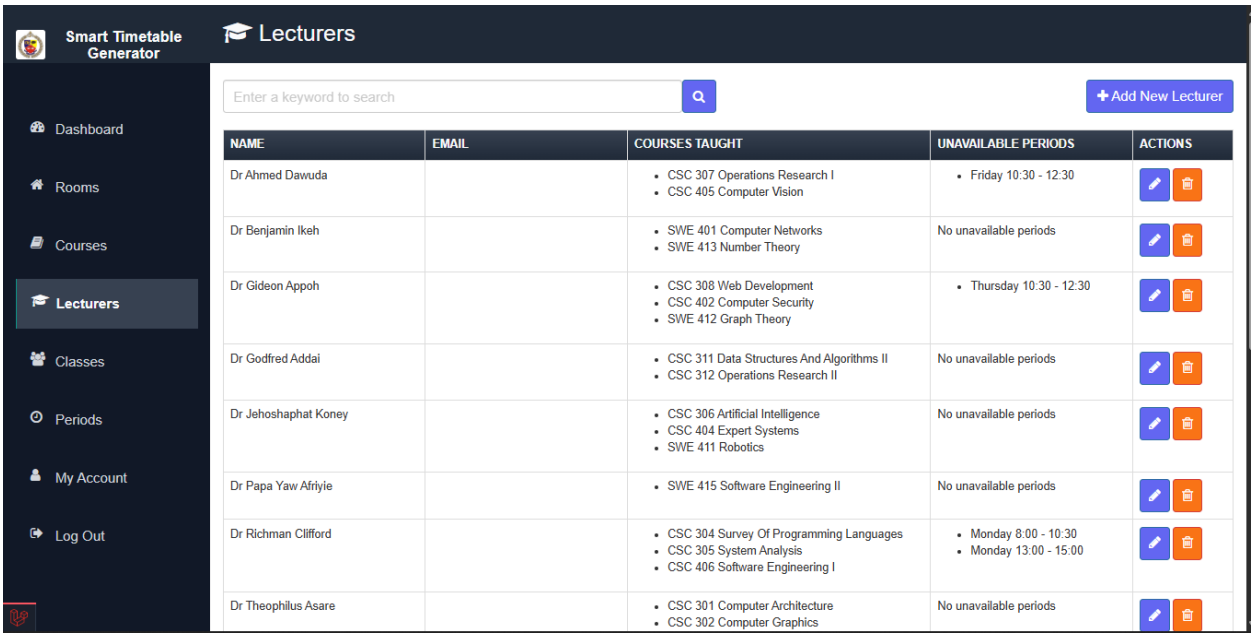

Figure 4.7 LecturersPage

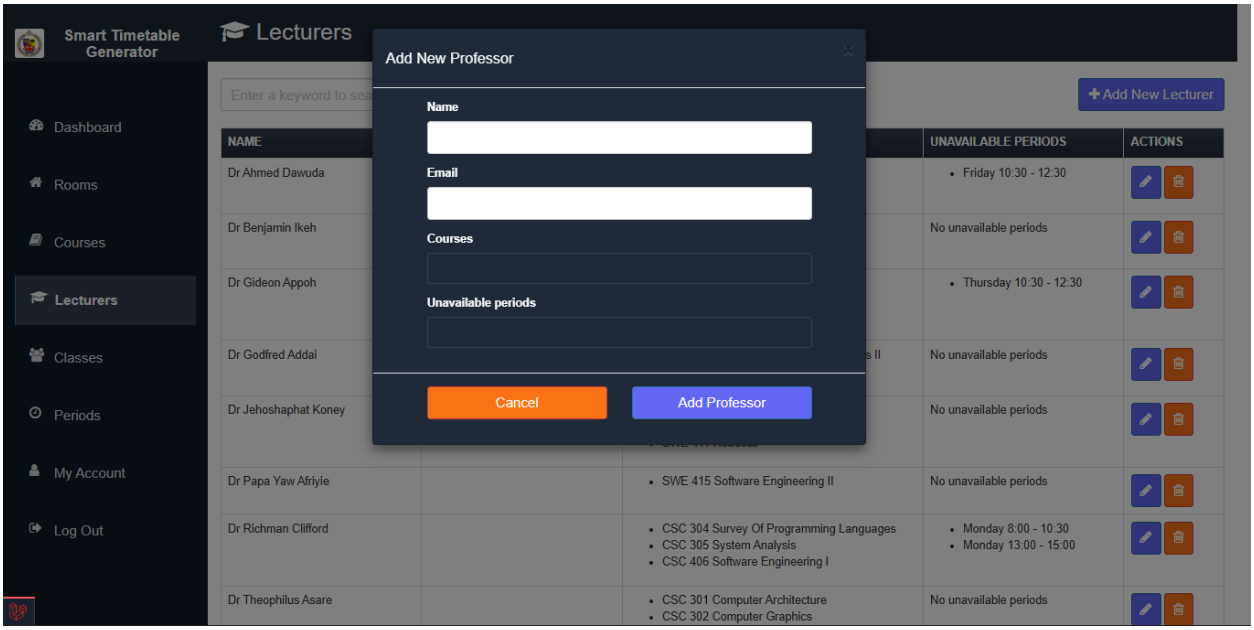

Figure 4.8 Add new professor modal

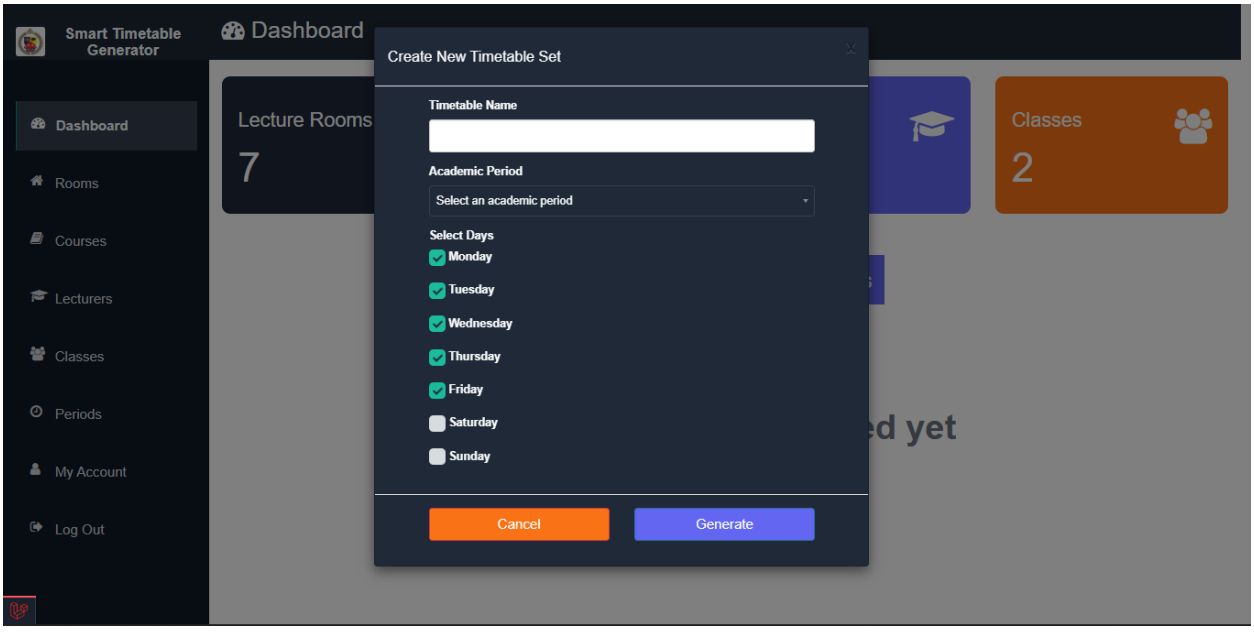

Figure 4.9 create timetable modal

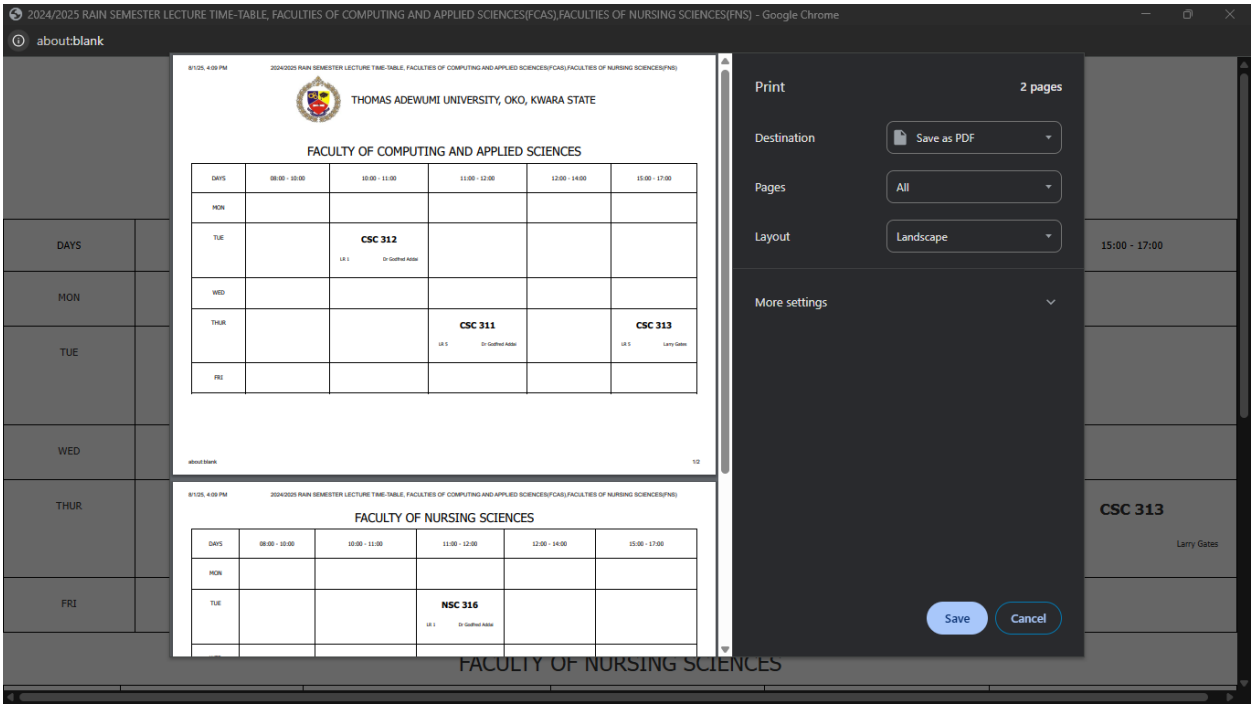

Figure 4.10 created timetable PDF

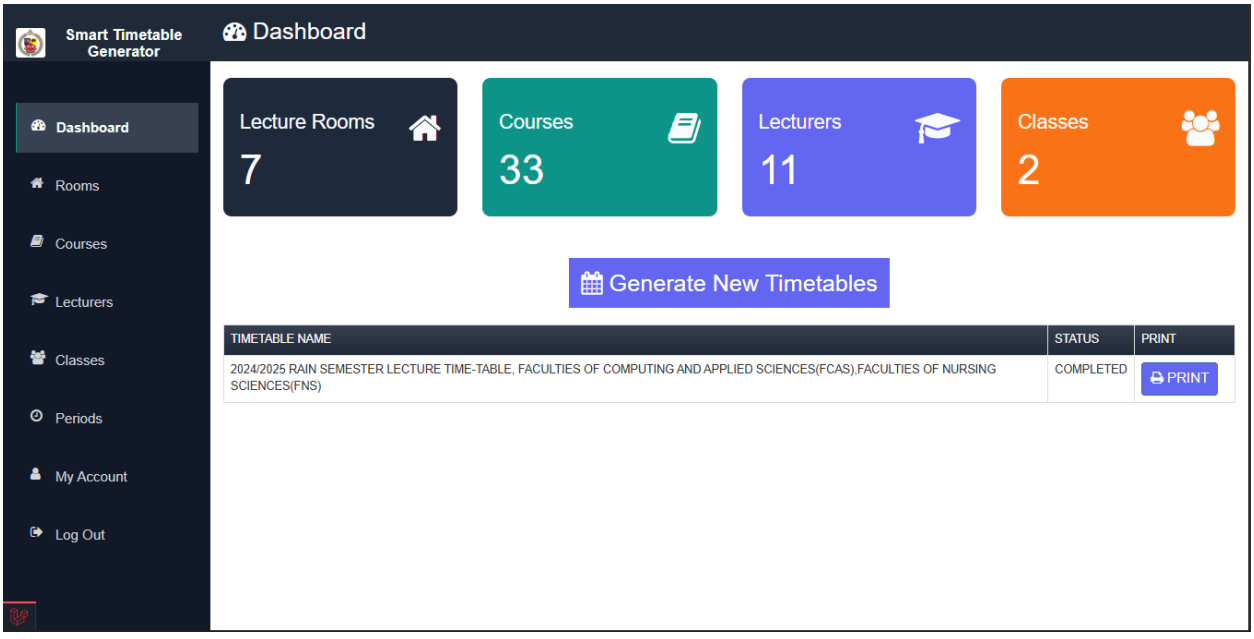

Figure 4.11 Dashboard timetable list

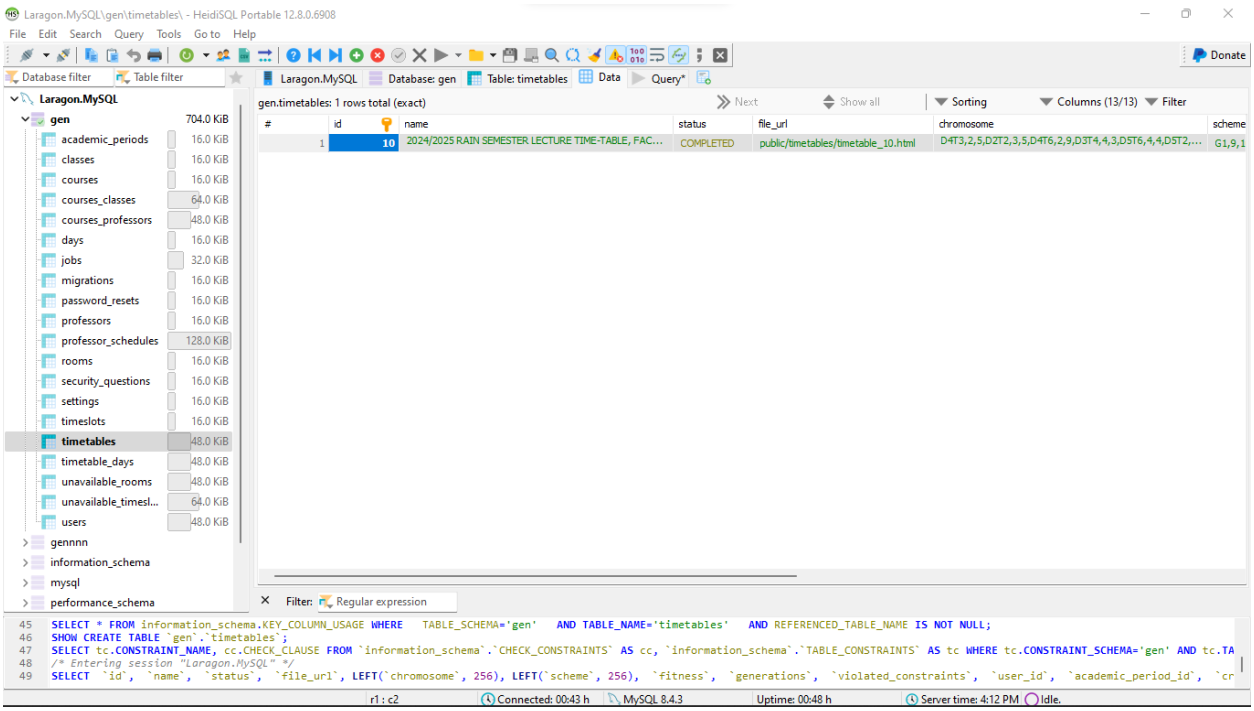

Figure 4.12 SQL timetable list

CHAPTER 5

SUMMARY, CONCLUSION, AND RECOMMENDATIONS

5.1 Summary

The Smart Timetable Generator project successfully demonstrated that advanced optimization algorithms can address the complex course scheduling problem in educational institutions. By leveraging a **genetic algorithm (GA)** as the core engine, augmented with heuristic techniques such as the **N-Queen strategy** for initial placement, the system generated **conflict-free, resource-efficient timetables** that satisfied all hard constraints and most soft preferences. This outcome confirms that automated scheduling significantly outperforms manual methods in both efficiency and quality, reducing planning time from weeks of adjustments to near-instant results.

A key contribution of the project lies in its **hybrid optimization approach**, where GA was enhanced with heuristics to improve feasibility and convergence speed. This reflects trends in recent research (e.g., Barhate et al., 2024; Bello & Godwill, 2024), which show that hybridizing metaheuristics accelerates convergence and improves solution quality. Our GA's evolutionary operators (selection, crossover, mutation) effectively explored the NP-hard search space, producing stable high-quality timetables within reasonable time limits.

In comparison with prior works, our findings align with studies that highlight the superiority of GAs and related metaheuristics over manual or greedy methods. Unlike many theoretical models, however, this project also emphasized **user-centric features**, including a friendly interface, real-time adjustments, and cloud synchronization. These features ensure that generated timetables are not only optimized but also practical, adaptable, and directly usable by academic administrators.

The system also addressed common GA limitations, such as stagnation and local optima, by using a large, diverse initial population and heuristic-based initialization. This improved both convergence and robustness, consistent with similar improvements reported in the literature (e.g., Zhang, 2022). While alternative metaheuristics (e.g., Particle Swarm Optimization, Artificial Bee Colony) have shown promise in other contexts, the GA was sufficient and effective for the scale of this project.

The Smart Timetable Generator **met its objectives** by combining optimization efficiency with practical usability. The project confirmed that genetic algorithms, especially when hybridized with domain heuristics, can reliably generate high-quality timetables, minimize conflicts, and adapt dynamically to institutional needs. This strengthens the case for automated timetabling systems as valuable tools for modern educational management.

5.2 Conclusion

This project successfully delivered an automated **Smart Timetable Generator**, addressing the inefficiencies of manual scheduling, which is often slow, error-prone, and resource-intensive. By applying a **genetic algorithm (GA)** enhanced with custom heuristics, the system was able to explore thousands of possible schedules and produce conflict-free timetables that satisfied hard constraints (no overlaps of teachers, rooms, or classes) and many soft preferences. The resulting application provides a modern, user-friendly interface through which administrators can input requirements and generate optimized timetables in minutes instead of weeks.

The project makes three main contributions. First, it demonstrates the **practical viability of GA-based optimization** for real-world academic scheduling, producing timetables that are both

accurate and resource-efficient. Second, it delivers a **functional web-based system** that integrates this optimization engine into an accessible platform, reducing human effort and minimizing errors such as double-bookings. Third, it introduces **dynamic adaptability**, allowing real-time updates so that changes (e.g., lecturer unavailability or room adjustments) can be quickly incorporated without rebuilding the entire timetable manually.

Overall, the Smart Timetable Generator provides a solution that is both **theoretically sound and operationally effective**. It transforms the role of administrators from manual schedulers to verifiers and planners, improving efficiency and accuracy in timetable management. Beyond its immediate utility, the project contributes to educational technology by showing how AI-driven optimization can streamline academic administration and serves as a **reference model** for future research and institutional adoption.

5.3 Recommendations

Based on the insights from this project, several recommendations are proposed for both institutions adopting automated timetable generators and researchers seeking to advance the field.

1. Adoption and Training:

Institutions should customize the system to their specific rules (class types, room needs, faculty preferences) and involve staff during configuration to capture all requirements. Training sessions are essential so administrators can input data, interpret schedules, and make adjustments confidently. Piloting the system in a department before full deployment can build trust and validate results.

2. Integration and Data Management:

For smooth operation, timetable generators should integrate with institutional databases (courses, instructors, rooms) to ensure accurate, up-to-date information. Clear data management practices—such as updating availability or room changes promptly—are vital. A feedback loop should be established so staff and students can report issues, which can then inform future schedule generations.

3. Enhancing Constraint Handling:

Future research should focus on more advanced modeling of complex "soft" constraints and trade-offs. Hybrid approaches (e.g., GA combined with local search or integer programming) and multi-objective optimization can better balance competing requirements like fairness, convenience, and satisfaction.

4. Real-Time Updates:

Future systems should support automated **dynamic rescheduling**, re-optimizing portions of the timetable when unexpected changes occur (e.g., lecturer absence, room closure) without rebuilding everything. Incremental or real-time optimization would make timetables living documents, adaptable to daily needs.

5. Scalability:

Large universities or multi-campus institutions should evaluate whether systems can handle greater scheduling complexity. Researchers should explore distributed, cloud-based, or co-evolutionary optimization methods to efficiently manage multi-campus timetabling while respecting inter-campus constraints.

6. Continuous Improvement:

Institutions should regularly evaluate system performance by tracking manual adjustments, resource utilization, and stakeholder satisfaction. Developers and researchers should work with standardized benchmarks and datasets to ensure systems evolve with real-world needs.

5.4. Future Works & Recommendations

1. **Enhanced Constraint Handling**

   Develop more sophisticated models for "weak" constraints (preferences) and conditional constraints.

   Explore hybrid algorithms (e.g., GA combined with constraint programming or fuzzy logic) to better balance competing soft constraints (e.g., student satisfaction vs. faculty workload).

2. **Real-Time Dynamic Rescheduling**

   Implement automated incremental rescheduling for unforeseen changes (e.g., room unavailability, instructor absences).

   Design algorithms to re-optimize affected schedule portions without overhauling the entire timetable.

3. **Scalability for Multi-Campus/Large Systems**

   Extend the system to handle multi-campus scheduling constraints (e.g., inter-campus travel time, shared resources).

   Investigate distributed/cloud-based optimization (e.g., cooperative co-evolutionary algorithms) for large-scale deployments.

4. **Standardized Benchmarks & Evaluation**

Establish universal benchmarks and metrics (e.g., standardized penalty formulations) to objectively compare timetable solutions.

Report real-world deployment outcomes (e.g., user satisfaction, manual adjustments needed) to guide future research.

5. **Continuous Improvement & Integration**

Integrate with institutional data systems (e.g., registrar databases) for automated data flow.

Implement feedback mechanisms to capture user preferences and refine constraints iteratively.

Conduct longitudinal studies on resource utilization and stakeholder satisfaction post-deployment.

References

Ambhore, P., Patil, S., & Kulkarni, S. (2020). Optimization techniques for academic timetable generation. *Journal of Applied Research in Computer Science, 15*(2), 78–89.

Diallo, F. P., & Tudose, C. (2024). Optimizing the scheduling of teaching activities in a faculty. *Applied Sciences, 14*(20), 9554. https://doi.org/10.3390/app14209554

D'Souza, D., D'Sa, O., Pillai, P., & Shaikh, P. (2020, June 26–27). Multi-constraint satisfaction and solution optimization using a genetic algorithm for solving the timetable generation problem. In *Proceedings of the International Conference on Recent Advances in Computational Techniques (IC-RACT 2020)* (pp. 1–6). Mumbai, India.

García Yáñez, J. A., Ramírez Saenz, F. J., Ríos Saldaña, E. L., Rubio Aguilar, M. A., Barrios Flores, I. E., García Flores, C. L., … & Zacarías Ortiz, J. J. (2024). *Development of a web-based timetabling software for a Mexican university* (Working Paper No. 2024-006). Universidad del Noreste. https://www.tamiu.edu/cswht/documents/wp-2024-006-garcia-yanez.pdf

Kavade, M., Deshpande, N., & Kulkarni, R. (2023). Machine learning applications in timetable scheduling systems. *International Journal of Artificial Intelligence and Applications, 9*(1), 67–79.

Mahlous, A. R., & Mahlous, H. (2023). Student timetabling genetic algorithm accounting for student preferences. *PeerJ Computer Science, 9*, e1200. https://doi.org/10.7717/peerj-cs.1200

Nguyen, V. D., & Nguyen, T. (2021). An SHO-based approach to timetable scheduling: A case study. *Journal of Information and Telecommunication, 5*(4), 421–439. https://doi.org/10.1080/24751839.2021.1935644

Tarale, D., & Bhuyar, M. (2025). A novel algorithm for efficient university timetable scheduling. *International Journal of Computer Science and Information Security, 17*(3), 112–125.

Iqbal, Z., Ilyas, R., & colleagues. (2021). Effective solution of university course timetabling using particle swarm optimization. *Baghdad Science Journal, 18*(4), 50. https://bsj.uobaghdad.edu.iq/home/vol18/iss4/50/

Mao, J., Liu, C., & Zhang, H. (2021). Particle swarm optimisation variants and their hybridisation ratios for generating cost-effective educational course timetables. *SN Computer Science, 2*(8), 652. https://doi.org/10.1007/s42979-021-00652-2

Saidani, O., & Dahmani, A. (2022). School timetabling optimisation using artificial bee colony algorithm based on a virtual searching space method. *Mathematics, 10*(1), 73. https://doi.org/10.3390/math10010073

Shi, Y., Wang, X., & Chen, L. (2021). Two-stage multi-neighborhood simulated annealing for uncapacitated examination timetabling. *Journal of Scheduling, 24*(3), 389–404. https://doi.org/10.1007/s10951-021-00652-y

Tan, J., & Wu, Y. (2022). Analysis of college course scheduling problem based on ant colony algorithm. *Journal of Intelligent Systems, 31*(5), 560–574. https://pubmed.ncbi.nlm.nih.gov/36059409/

Wang, L., Zhang, F., & Liu, Y. (2023). Ant colony optimization algorithm for the university course timetabling problem using events based on groupings of students. *Applied Artificial Intelligence, 37*(4), 289–304. https://doi.org/10.1080/08839514.2023.2173615

Xu, J., Li, H., & Zhao, Q. (2020). A hybrid algorithm for the university course timetabling problem using the improved parallel genetic algorithm and local search. *Journal of Heuristics, 26*(5), 721–748.

Yildiz, O., & Aksu, Y. (2023). Meta-heuristic approaches for the university course timetabling problem. *Artificial Intelligence Review, 56*(7), 5975–6002. https://doi.org/10.1007/s10462-023-10400-1

Yusuff, A., & Bamidele, O. (2021). Design and implementation of a web-based timetable system for higher education institutions. *International Journal of Computer Applications, 183*(2), 1–7. https://doi.org/10.13140/RG.2.2.36146.35520

Zhang, Y. (2022). A survey of university course timetabling problem: Perspectives, trends, and opportunities. *Computers & Operations Research, 142*, 105701. https://doi.org/10.1016/j.cor.2022.105701