Yasin Lester M00809695

BSc Computer Science Department of Computer Science, Middlesex University

Development of a Cloud-Based Event Management and Booking System for Middlesex University

# Declaration

I, Yasin Lester, hereby attest that the work presented in this report and all other related materials is entirely my own. The text has appropriately cited the information derived from the literature, and a list of references has been provided. No section of this dissertation has been previously presented for another degree or diploma at this or another institution.

Signature: Yasin Lester Date: 12/04/2025

# Abstract

Event ticket administration is an important part of university operations; nevertheless, Middlesex University continues to use an antiquated, inefficient system that requires a lot of staff time and hinders the navigation process for some students. The project's goal is to create a web-based event booking system that will streamline the reservation process, reduce administrative obligations, and improve the experience for both students and staff. AWS services will make the system scalable and stable, and it will use JavaScript, CSS, and HTML. An easy-to-use interface is provided, along with real-time availability statistics, making it accessible from anywhere. Encryption, role-based access control, and multi-factor authentication are used to protect personal data. The goal of this program is to improve event management efficiency. The system makes use of cloud-based approaches to improve the reliability, utility, and scalability of event reservations. This will increase student engagement while also improving operational efficiency.

*Keywords: Event Management System, Web Development, AWS, Cloud Computing, University Booking System, Real-Time Availability, User Experience, Security, Scalability, Full-Stack Development.*

# Acknowledgements

I would like to express my sincere thanks to Mr. Ahmed Eissa, my project supervisor, for being an outstanding source of support throughout this project. He was always approachable, dedicated, and generous with his time. His guidance, patience, and valuable feedback helped me stay focused and make consistent progress, especially when facing technical challenges. I deeply appreciate the effort he invested in supporting my work and development.

I would also like to thank my lecturers and peers at Middlesex University for creating a collaborative learning environment that encouraged growth and innovation.

Table of Contents

[Declaration 1](#_Toc195467368)

[Abstract 1](#_Toc195467369)

[Acknowledgements 1](#_Toc195467370)

[1.Introduction 4](#_Toc195467371)

[1.1 Project Proposal 4](#_Toc195467372)

[1.2 Motivation 5](#_Toc195467373)

[2. Literature Review and background 6](#_Toc195467374)

[2.1 Literature Review 6](#_Toc195467375)

[2.1.1 Advanced Full-Stack Development for Scalable Solutions 6](#_Toc195467376)

[2.1.2 Vision, Challenges, and Future of Web Technologies 6](#_Toc195467377)

[2.1.3 Performance Optimization Techniques in Web Applications 6](#_Toc195467378)

[2.1.4 Data Security and Compliance in Cloud Services 6](#_Toc195467379)

[2.1.5 User Experience (UX) Design Principles in Event Booking Systems 7](#_Toc195467380)

[2.1.6 API Integration in Modern Web Development 7](#_Toc195467381)

[2.1.7 Scalability Challenges and Solutions in Web Applications 7](#_Toc195467382)

[2.1.8 Sustainability Considerations in Software Engineering 7](#_Toc195467383)

[2.1.9 Emerging Technologies: AI and Blockchain in Web Development 8](#_Toc195467384)

[2.1.10 Comparative Analysis of Cloud Platforms 8](#_Toc195467385)

[2.1.11 Future Trends in Web Development 8](#_Toc195467386)

[2.1.12 Case Studies and Real-World Applications 8](#_Toc195467387)

[2.2 First steps 8](#_Toc195467388)

[3.Problem Description and Problem Statement 12](#_Toc195467389)

[3.1 Problem Overview 12](#_Toc195467390)

[3.2 Aims and Objectives 12](#_Toc195467391)

[3.3 Constraints and Assumptions 12](#_Toc195467392)

[3.4 Justification 13](#_Toc195467393)

[4.Theory and advanced preliminaries 13](#_Toc195467394)

[4.1 Full-Stack Web Architecture 13](#_Toc195467395)

[4.2 Cloud Computing and AWS 14](#_Toc195467396)

[4.3 Web Security Fundamentals 14](#_Toc195467397)

[4.4 Responsive Web Design and UX Principles 14](#_Toc195467398)

[4.5 Database Design with MongoDB Atlas 14](#_Toc195467399)

[4.6 API Integration and Communication 15](#_Toc195467400)

[4.7 Authentication and Login Flow 15](#_Toc195467401)

[4.8 Form Validation and Error Handling 15](#_Toc195467402)

[4.9 Environment Setup and Testing Tools 16](#_Toc195467403)

[5.Development and Implementation 16](#_Toc195467404)

[5.1 Tools and Technologies Used 16](#_Toc195467405)

[5.2 Development Process 17](#_Toc195467406)

[5.2.1 Phase 1: Frontend Development (Expanded) 17](#_Toc195467407)

[5.2.2 Phase 2: Backend and API implementation 25](#_Toc195467408)

[5.2.3 Phase 3: Database Integration with MongoDB Atlas 26](#_Toc195467409)

[5.3 Session Management and Access Control 28](#_Toc195467410)

[5.4 Navigation and Dynamic Page Updates 28](#_Toc195467411)

[5.5 Implementation Summary 29](#_Toc195467412)

[6.Testing and Results 29](#_Toc195467413)

[6.1 Testing Methodology 29](#_Toc195467414)

[6.2 Backend API Testing (Postman) 30](#_Toc195467415)

[6.2.1 User Registration (POST /api/register) 30](#_Toc195467416)

[6.2.2 User Login (POST /api/login) 30](#_Toc195467417)

[6.2.3 Event Creation (POST /api/events) 30](#_Toc195467418)

[6.2.4 Get All Events (GET /api/events) 30](#_Toc195467419)

[6.2.5 Book Event (POST /api/book) 30](#_Toc195467420)

[6.2.6 Cancel Booking (DELETE /api/bookings/:id) 30](#_Toc195467421)

[6.3 Frontend Testing 34](#_Toc195467422)

[6.4 Summary of Results 34](#_Toc195467423)

[7.Conclusion 35](#_Toc195467424)

[8. Evaluation and Discussion 36](#_Toc195467425)

[8.1 Limitations 36](#_Toc195467426)

[8.2 Originally Planned Features 37](#_Toc195467427)

[8.2.1 Full AWS Deployment 37](#_Toc195467428)

[8.2.2 Security Measures 37](#_Toc195467429)

[8.2.3 Support Page Functionality 38](#_Toc195467430)

[8.2.4 Interactive Calendar 38](#_Toc195467431)

[8.2.5 Editable User Profile with Picture and Bio 38](#_Toc195467432)

[8.2.6 Forgot Password Functionality 38](#_Toc195467433)

[8.3 Additional Future Enhancements 38](#_Toc195467434)

[8.3.1 Event Capacity Management 38](#_Toc195467435)

[8.3.2 Email Notification System 38](#_Toc195467436)

[8.3.3 Email Domain Verification 39](#_Toc195467437)

[8.3.4 Event Search and Filtering 39](#_Toc195467438)

[8.3.5 Live Calendar View 39](#_Toc195467439)

[8.3.6 Admin Dashboard and Analytics 39](#_Toc195467440)

[8.3.7 Real-Time Updates with WebSockets 39](#_Toc195467441)

[8.3.8 Frontend Styling Enhancements 39](#_Toc195467442)

[9.References 40](#_Toc195467443)

[10.Appendices 40](#_Toc195467444)

[10.1 Command Line Instructions 40](#_Toc195467445)

[10.2 Installed Node.js Packages 41](#_Toc195467446)

[10.3 Project Demonstration Video 41](#_Toc195467447)

[10.4 Ethics Form 41](#_Toc195467448)

# 1.Introduction

Middlesex University has historically taken great satisfaction in being a hive of activity both on and off campus. On the other hand, an antiquated manual booking system makes it far more challenging for the university to properly run its event venues. Event spaces are currently managed in several methods, including actual booking records and email requests. This makes it simple for scheduling errors including last-minute cancellations and multiple bookings to occur. These mistakes sometimes cause uncertainty, frustration, and a lot of work for officials, therefore detracting from the vibrant campus environment.

The recent university-wide initiative to increase student participation in extracurricular events revealed the shortcomings in the present system. This project's comments made it quite evident that consumers needed a quicker and simpler booking system. Teachers and students claimed that delays made planning difficult; office staff members struggled with all the documentation they needed to complete. This input, together with records of bookings occurring concurrently during hectic events like orientation days and test weeks, made it abundantly clear that a streamlined, digital system approach is urgently required.

In response, this project proposes developing a sophisticated web-based booking system to facilitate running Middlesex University's event venues. Although there are several objectives for this new system, its main ones are to significantly reduce the time needed to book rooms, increase the accuracy of data about accessible rooms, and make all users including staff members and students happy. Additionally, simpler use of the ticketing system and improved organisation should encourage more attendance of events. This is so because they can now make reservations and view event specifics without any issues, therefore fostering community and involving more people in university life.

By ensuring that key resources are always readily available, the system also seeks to assist the university's more ambitious objectives of increasing student involvement and promoting innovative approaches of teaching. The fresh approach is supposed to have a broad spectrum of consequences. More efficiency in the process will assist students in both social and academic spheres. Scheduling of lectures and activities will also be simplified for teachers; administrative staff will have much less work to manage regarding booking management.

Using Amazon AWS, the proposed system will be built on a solid basis ensuring it can expand and remain dependable. Real-time updates on accessible rooms, mobile compatibility when using internet browser to fit the reality that many students use smartphones, and user-friendly design concepts to guarantee simple usability. With an eye towards ensuring the system runs well with the university's present digital infrastructure, preliminary research has identified several computer languages and frameworks that would be beneficial in constructing the system.

Apart from simplifying logistics, the motivation behind this project transcends mere convenience. With innovative tools like Amazon AWS, Middlesex University aims to raise its profile for technological innovation. This approach is not only a solution for a problem but also a strategic development that complements the university's objective to establish a useful and effective environment for students to learn and develop.

## 1.1 Project Proposal

The need for a new event booking system at Middlesex University arose due to inefficiencies in the current manual approach. These inefficiencies such as the need for more paperwork and scheduling issues found in market research and demonstrated the need of a simplified digital solution. Thus, a web-based booking system was proposed as a means of enhancing the managing of university event venues by means of improved accessibility, more efficient operation, and better user experience.

Though it has not been implemented yet, Amazon AWS has been intended as the backend technology to guarantee high availability and scalability. AWS services like EC2 for storage, RDS for database management, and Lambda for backend operations will be included to make the system run as soon as these fundamental steps are completed. This designed approach will enable the system to meet evolving requirements while maintaining performance and safety standards high.

The system has an easy-to-use interface and is supposed to provide real-time information on which rooms are accessible, therefore facilitating faster and simpler management of event venues. By using automation, the chances of double bookings and scheduling conflicts should go down a lot. This will make it easier for students and staff to plan events for university and outside of university.

Because this is a web-based method, making it accessible has been a main goal from the start. The platform is being developed using HTML, CSS, and JavaScript, which will make sure that the interface is flexible and easy to use. The system will also work with several different devices, so students and staff will be able to make bookings conveniently from desktop computers, laptops, and mobile browsers. This approach aligns with contemporary technology usage habits, especially at the college level where having access to the internet is necessary for planning events quickly.

Integration with current university systems like academic calendars and student portals will also make event management easier and less reliant on people coordinating things by hand. Bookings that are coordinated with institutional plans will help with organisation, avoid conflicts, and make planning events go more smoothly.

In addition to making things easier, this project aims to help the university's larger goals of getting students more involved. An accessible and effective booking system is expected to increase event attendance, which will make the university community more active and welcoming for everyone. Students will be able to see more activities if there is an event management tool that is well-organised and easy to use. This will make them more likely to participate in both university and extracurricular events.

## 1.2 Motivation

The selection of this project was significantly influenced by a robust interest in web development and cloud computing technologies. An opportunity to explore full-stack development with AWS arose as possibilities were evaluated. This choice was especially useful due to previous experience with MongoDB in earlier web development classes, which established a foundation in database management and cloud-based apps. This project was selected to improve technical skills in creating scalable web-based solutions.

Furthermore, the project provides a significant educational opportunity in system construction, database administration, and cloud architecture. The objective was to apply full-stack development competencies to address event management challenges at Middlesex University. This practical engagement is anticipated to improve proficiency in database integration, security protocols, and scalable cloud solutions, all essential elements of modern software engineering.

This project facilitates the professional and academic development of students and staff while improving the university's operational efficiency and technological relevance. Integrating a bespoke event booking solution would streamline the management of university facilities and improve the overall experience for all university constituents, in accordance with the prevailing digital transformation trends in higher education.

# 2. Literature Review and background

## 2.1 Literature Review

Full-stack web development and reliable cloud services work together to make event management systems better and help them grow, especially in educational settings. A close study of scientific papers and technological reports shows a lot of diverse ways that these technologies can be changed to fit the specific needs of university campuses.

### 2.1.1 Advanced Full-Stack Development for Scalable Solutions

Rajan et al. (2024, p. 830) carefully investigates the possibilities of using Amazon AWS as a full-stack development platform, pointing out that it is a key part of making scalable and reliable business apps. Their research looks in depth at AWS services like Elastic Compute Cloud (EC2) and Relational Database Service (RDS), which are perfect for university event planning systems that deal with a lot of several types of traffic. This strong platform not only allows for expansion, but also guarantees dependability and safety, which are especially important when managing confidential data about students and events.

Traditional ways of teaching full-stack web development are criticised and expanded upon by Petrikoglou and Kaskalis (2019, p. 218). They want to create a new, more immersive model that puts an emphasis on firsthand, project-based learning and is remarkably similar to what the business needs. This study is especially useful because it shows how important it is for engineering teams that build and manage complex systems like the proposed booking system to get firsthand experience and skill development that is relevant to the field.

### 2.1.2 Vision, Challenges, and Future of Web Technologies

Sannakki (2022, p. 3083) gives a broad view of the difficulties and opportunities that lie ahead for full-stack web development. This essay discusses about modern technologies and how web frameworks are continually changing, which could have a huge effect on how the suggested booking system looks and works. Developers may make sure the system is adaptable and works with new web technologies by keeping up with current trends.

Research by Baiskar et al. (2022, p. 1032), for example, looks at how to use the MERN stack to construct scalable web apps. When you mix MongoDB, Express.js, React.js, and Node.js, you get a seamless development experience and the stability you need for data-heavy activities like event management and booking rooms. This stack not only speeds up development, but it also helps you manage data in real time, which is vital for getting immediate updates on room availability and confirmations of bookings.

### 2.1.3 Performance Optimization Techniques in Web Applications

Web apps, especially ones that manage intricate event management activities, need to be able to manage a number of users at once. Techniques like caching, which saves copies of files in a cache, or temporary storage space, can make it faster to access to resources that are used a lot. One method that caching can be used to speed up load times in the cloud is with AWS's Elastic Cache. Load balancing is another key strategy that divides data among numerous servers so that no one server needs to manage too much demand. As a technique to increase application availability and fault tolerance, AWS Elastic Load Balancing handles how user data is spread across Amazon EC2 instances, containers, and IP addresses.

### 2.1.4 Data Security and Compliance in Cloud Services

It is highly crucial that web apps are safe, especially when they deal with personal data like student details in a university booking system. Web-based platforms are exposed to several types of assaults that can impair system security and user privacy. Serve-Side Template Injection (SSTI) is one of these threats. If it is not properly confined, it can cause remote code execution (RCE), information leakage, and unauthorised access (Pisu et al., 2024, p. 6). Mishandling of dynamic material in web apps is a common cause of SSTI vulnerabilities, which is why it is so crucial to think about this while creating safe systems.

Secure coding and rigorous input validation is essential to prevent malicious individual from taking advantage of web templates in this way. On top of that, security frameworks and best practices like web application firewalls (WAFs), role-based access control (RBAC), and encrypting personal data can help make assaults less likely. Identity and Access Management (IAM) and Cognito are two technologies that AWS provides for handling security policies and authentication. However, it is equally as critical to patch basic security gaps in apps. The event booking system can reduce risks by putting in place initiative-taking security measures during the development phase. This safeguards both data privacy and system stability.

### 2.1.5 User Experience (UX) Design Principles in Event Booking Systems

Making ensuring that online services are accessible and easy to use entails making sure that the user experience is improved through effective design. As seen in the use of the MERN stack, React is a JavaScript tool for developing user interfaces that enables adding new features to web apps that users see easy, rapid, and adaptive. Developers can build systems that are easy to use and comprehend by following user-centred design principles. The total pleasure of users increases higher with this strategy, and it also encourages more users, especially those with disabilities, to adopt it and get active.

### 2.1.6 API Integration in Modern Web Development

APIs are particularly significant in current software development because they make it easier for different systems to talk to one other. When it comes to a university event booking system, APIs make it easier for the booking system to talk to other institution databases, like academic calendars and student websites. Kurth et al. (2022, p. 5) talk about how vital well-designed APIs are in hybrid computing. They underline how APIs make systems more portable and make it easier for sophisticated systems to work together. In the same manner, APIs make it easier to share data in a web-based system. This makes sure that the booking system changes in real time, offering users the most up-to-date information on events that are available, and boosting the general efficiency of operations.

### 2.1.7 Scalability Challenges and Solutions in Web Applications

Scalability is a crucial aspect of the architecture of current web-based apps, especially ones that need to accommodate varied amounts of user demand. AWS Auto Scaling and other cloud platforms have traditionally let apps alter their resources on the fly to keep performance high. Web Assembly (Wasm), on the other hand, can be used to improve scalability at the program level. Web Assembly lets web browsers run code at speeds that are virtually as fast as native code. This allows web apps run faster without requiring too much server-side labour.

Sipek et al. (2021, p. 4) discuss about how Web Assembly may make the best use of resources by making backend systems' duties easier. Wasm lowers server load by doing operations that need a lot of computing power in the browser. This divides job processing between the client and the server. Particularly for systems like platforms for scheduling events at colleges that must manage a lot of people, this approach can enable applications far more rapid and scalable.

### 2.1.8 Sustainability Considerations in Software Engineering

In software engineering, sustainability goes beyond just environmental consideration. It also covers social and financial aspects including the lifetime of software systems and their maintenance frequency. By applying cloud services and adhering to well-known best practices in software engineering, developers can create systems that span years and are simpler to administer and improve over time. This reduces the whole cost of ownership as well as the environmental damage running server farms causes.

### 2.1.9 Emerging Technologies: AI and Blockchain in Web Development

AI integrated to systems can automate mundane tasks and provide predicted insights, therefore altering data handling. AWS provides several machine learning tools that might be included to web projects to increase their value. One can build, train, and apply machine learning models using Amazon SageMaker, for instance. Blockchain might also fundamentally alter the safety and clarity of online programs. Although it is not used in booking systems, blockchain helps in cases when high degrees of accuracy and auditability are required since it maintains unchangeable records that enhances security.

### 2.1.10 Comparative Analysis of Cloud Platforms

Though AWS has attracted a lot of interest, considering it next to Microsoft Azure and Google Cloud Platform will help you understand why it could be the best option for this project. Every platform can be used for a different reason since they include different cost and set of capabilities. This comparison research would support AWS's low cost, worldwide infrastructure, and service offerings in helping to justify its choice.

### 2.1.11 Future Trends in Web Development

Latest trends including serverless architectures, progressive web applications (PWAs), and the need of designing for mobile devices first will define the age of web development going forward. These technologies hold enormous potential to make web apps scalable, efficient, and more easily available. For next developments and enhancements to the university booking system, this could be quite crucial.

### 2.1.12 Case Studies and Real-World Applications

Brigham Young University and Northern Arizona University demonstrate the application of cloud-based event management tools using the Coursedog platform. Operations are now significantly more efficient because of this technology. For example, the time required to comprehend curriculum concepts has decreased by ninety percent, and room arrangements are now computerised. It is evident from these enhancements that cloud-based technologies have the potential to facilitate the seamless operation of educational environments.

These systems addressed common issues, such as system integration and user acceptance. By engaging all stakeholders in strategic planning and comprehensive training, the universities ensured that the transition was seamless. Addressing change at the outset reduces resistance and fosters a more conducive work environment for employees.

These examples demonstrate the significance of selecting technology that is not only compatible with the institution's objectives but also has a logical use and integration strategy. This level of strategic planning and stakeholder interaction enables colleges to implement cloud-based event management solutions, which facilitate event management and attract a greater number of participants.

## 2.2 First steps

Various platforms were extensively researched to identify what features were crucial and what issues would arise before beginning to work on the university event managing system. The study provided insights into effective approaches for developing a functional system with good functionality and simplicity. Balsamiq was used to create a collection of wireframes illustrating the layout and user movement across the system. These formed the cornerstone of the system's design.

Important portions of the site are shown in these wireframes: the home page (Figure 2.1), the page with events (Figure 2.2), the calendar view (Figure 2.3), the user profile (Figure 2.4), the page with support (Figure 2.5), and the page where you register or log in (Figure 2.6). Before actual development begins, they may be tested and refined since they represent the layout, interaction patterns, and navigation flow.

To create the responsive user experience of the front end, HTML, CSS, and JavaScript were combined with the wireframes. On the homepage, a dynamic event carousel (Figure 2.1) informs students about forthcoming events and provides an entertaining and interactive homepage. Figure 2.2: The event listing page features a "Book Now" button that facilitates users' reservation making and event details. Looking through the calendar view (Figure 2.3) makes it simple to arrange events depending on their occurrence. This front end was created using Visual Studio Code since it is user-friendly and features a built-in Live Server capability enabling real-time testing and the implementation of updates to improve functionality.

For the backend to hold and control the data of the system in a dependable and scalable manner, it is planned that Amazon Web Service (AWS) will be used (AWS). While Amazon RDS is supposed to make database management simple so that event and user data can be managed fast, AWS EC2 is meant to run the web server and provide dynamic scalability depending on traffic level. The selected services are dependable, secure, and compatible with other AWS products. Nevertheless, since the project is still in its early phases of development, the backend has not yet been configured. At present, the system design and front-end development are the primary focus.

Although not the primary objective, security has been a consistent consideration from the outset. SSL encryption was intended to be applied to guarantee that data moved across the site was safe. Advanced Encryption Standards (AES) were also considered as a means of safeguarding personal data, safeguarding user privacy, and preventing unauthorised access into systems. By restricting access to certain capabilities to only approved users, Role-Based Access Control (RBAC) was supposed to improve security even further. Data breaches would thus be less likely.

Users of the profile page (Figure 2.4) can manage their personal information and event record. Its simple dashboard also facilitates the tracking of participant information. The whole user experience is enhanced when users may easily obtain help from the support page (Figure 2.5). Personalised event management is set up by the easy account creation and safe entry made possible by the login and registration page (Figure 2.6). Multi-factor authentication (MFA) could be included as an additional security step in next revisions.

Strong security policies and AWS services will help develop the booking system to be compatible with Middlesex University's security demands. It is also designed to be easily accessible for staff and students, hence dependable. The wireframes are quite crucial since they ensure that every component is produced with usability and functionality in mind. This guarantees flawless progress of the development process.

A screenshot of a computer

AI-generated content may be incorrect.Figure 2.1 homepage wireframe.

A computer screen with many different types of text

AI-generated content may be incorrect.

Figure 2.2-events page wireframe

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2.3 calendar wireframe

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2.4 profile wireframe

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2.5 user support wire frame

A computer screen shot of a computer

AI-generated content may be incorrect.

Figure 2.6 user login and register wireframe.

# 3.Problem Description and Problem Statement

## 3.1 Problem Overview

The existing event booking procedure at Middlesex University is primarily reliant on out-of-date manual methods such as email-based enquiries, spreadsheets, and fragmented departmental communication. These processes are inefficient and prone to human mistake, especially at busy times like induction weeks, revision seasons, and university-wide events.

Staff frequently struggle to coordinate room usage, while students encounter delays and uncertainty while attempting to reserve seats. This lack of automation leads to multiple bookings, scheduling conflicts, and lost resources. Furthermore, it adds an extra strain to administrative workers, who must manually track and confirm each request.

These inefficiencies are addressed with the implementation of a streamlined, web-based booking system. The goal is to modernise the university's approach, decrease human workload, and provide users with a simple, accessible, and real-time platform for event administration.

## 3.2 Aims and Objectives

The project's original goal was to create and build a responsive, secure, and scalable event booking platform using cloud infrastructure. While Amazon Web Services (AWS) was intended for backend deployment and scalability, full AWS integration was not achieved due to technical integration issues and the complexity of implementing new technologies that were still being investigated during the development process.

Despite this, the project remained focused on its core aims:

* To design a user-friendly, web-based event booking system
* To automate and simplify the room/event reservation process.
* To reduce the risk of double bookings and scheduling conflicts
* To allow real-time updates and dynamic event management
* To explore scalability options using cloud platforms like AWS
* To enforce secure login and basic role-based access functionality
* To reduce administrative burden and support student engagement

## 3.3 Constraints and Assumptions

Constraints:

* The project was developed within a limited academic timeframe.
* Full integration with AWS infrastructure was originally planned but could not be implemented due to backend compatibility issues and project time constraints.
* Development was focused solely on a browser-accessible web platform.

Assumptions:

* Users (students and staff) would access the system via standard web browsers.
* The university would provide server support (e.g., AWS or institutional hosting) for future deployment.
* Booking-related data used in the system reflects typical university scenarios.
* Formal user testing will be conducted once the final system is complete, to ensure the design is intuitive, functionality meets expectations, and the platform performs well under real conditions.

User testing is regarded as critical to the project's success. Although not yet implemented, it is intended for the next stage of development. Before a full rollout, the testing procedure will include obtaining feedback from potential users (students and staff) to assess usability, identify faults, and improve the interface and booking flow. This will help ensure that the system effectively addresses the identified booking concerns.

## 3.4 Justification

This project addresses the urgent need for digital change in how Middlesex University handles event venues. The present manual systems are slow, error-prone, and provide a terrible user experience. Introducing a web-based booking tool increases operational efficiency, enhances accessibility, and encourages greater student and staff engagement.

Although cloud infrastructure such as AWS was not included in the final build, the project was planned with scalability and security in mind. Future connection with AWS or comparable platforms remains possible and encouraged. This long-term perspective makes the project more than just an academic exercise; it also serves as a practical foundation for real-world deployment.

The project provided opportunity to use full-stack web development abilities in a realistic development setting, establish secure user authentication, and deal with real-world restrictions that arise in software projects. This report emphasises the project's outcomes and identifies opportunities for future development.

# 4.Theory and advanced preliminaries

This chapter covers the theoretical and technical foundations that will support the creation of the cloud-based event booking system. It discusses essential principles in full-stack architecture, cloud computing, web security, responsive design, database modelling, API communication, authentication, form validation, and testing, all of which influenced the system's fundamental operation.

## 4.1 Full-Stack Web Architecture

A full-stack web application is made up of three main layers: the frontend (client-side interface), backend (server-side logic), and database (data storage and management). These components communicate via HTTP protocols and APIs to ensure a consistent user experience across devices and browsers.

The interface was built with HTML, CSS, and JavaScript to allow users to interact with the platform, browse events, and make bookings. The backend handles logic such as booking requests, user session management, and data validation. This backend interfaces with the database to retrieve and store information in response to system queries made by users.

The separation of responsibilities across various levels promotes maintainability, scalability, and modular development, all of which are essential characteristics of modern online applications. It also assures that each component of the system can evolve independently as needed.

## 4.2 Cloud Computing and AWS

Cloud computing provides scalable, adaptable, and cost-effective hosting for online applications. Platforms such as Amazon Web Services (AWS) enable developers to create backend servers using EC2, manage cloud-based databases using RDS, and access tools for authentication, monitoring, and automatic scaling.

AWS is one of the most popular cloud providers due to its worldwide reach, dependability, and diverse ecosystem. The system created for this project was designed with future cloud deployment in mind, providing for scalability and high availability as user demand increases. Using cloud platforms also eliminates the requirement for physical infrastructure and allows for faster disaster recovery.

## 4.3 Web Security Fundamentals

Security is essential for protecting personal user data and ensuring system integrity. Several key security practices were incorporated into the system design:

* Login authentication: Only registered users can access protected features like booking.
* Hashed passwords: Passwords are encrypted before storage using hashing algorithms.
* Secure data handling: User credentials and booking information are not exposed directly.
* HTTPS (SSL/TLS): The platform is intended to run over secure HTTPS connections during deployment.
* Session control: Users stay logged in temporarily through saved session data in the browser.

While advanced techniques like role-based access control (RBAC) and multi-factor authentication (MFA) were not implemented, the system was structured with extensibility in mind.

## 4.4 Responsive Web Design and UX Principles

Responsive design ensures that websites appear correctly on a variety of screen sizes and devices. CSS was used to implement responsive layout approaches that accommodated multiple browser window widths, making the site useable on both desktops and laptops.

Good UX design enhances user satisfaction by focusing on usability, clarity, and feedback. The event booking system applies these principles through:

* A clean and structured interface
* Clear button placement and form labels
* Real-time feedback (e.g., error messages, booking confirmations)
* Navigation simplicity to reduce user effort.

These design principles improve user flow and ensure that students and staff can interact with the system efficiently.

## 4.5 Database Design with MongoDB Atlas

MongoDB Atlas is a cloud-hosted NoSQL database that stores data in flexible, JSON-like documents. This structure is highly suited to modern web applications, as it allows for unstructured and dynamic data.

The following structure was used:

* Each user document includes login and personal information.
* Each event document holds data such as title, date, and description.
* Bookings are recorded by linking user IDs to specific event entries.

MongoDB inherently supports one-to-many relationships, making it an ideal choice for systems that allow users to register for many events. It also supports quick querying and scalability without requiring complicated joins, making it suitable for agile web development.

## 4.6 API Integration and Communication

APIs (Application Programming Interfaces) are essential in connecting various parts of a system. In web development, APIs function as communication bridges between frontend and backend services. RESTful APIs provide a standard structure using HTTP methods like GET, POST, PUT, and DELETE.

Internal API endpoints are used to:

* Register and log in users.
* Submit and retrieve booking data.
* Fetch available events for display.
* Allow users to cancel events

Each API route manages requests and responds with structured data (typically in JSON format). While external integrations such as university timetable APIs were not implemented, the system is architecturally prepared to support them with minimal changes. This flexibility enables future growth and interconnectivity.

## 4.7 Authentication and Login Flow

Authentication is the process of validating a user's identity before granting access to protected system regions. LocalStorage was used to manage authentication and keep users signed in after they logged in.

When a user enters credentials, the backend checks them against the database. After a successful login, basic user data (such as a username or user ID) is saved in localStorage. This allows users to remain logged in across several pages until logout is initiated.

Passwords are salted before storage to avoid plain-text access, and login feedback is purposefully unclear, for example, an invalid username or password results in the same error message. This method makes it more difficult for attackers to determine legitimate login credentials using trial and error.

The system is designed such that future enhancements, such as Express-based sessions or JWT authentication, can be simply implemented.

## 4.8 Form Validation and Error Handling

Validation is a key part of maintaining data integrity and preventing user mistakes. In this system, both client-side and server-side validation are used to ensure smooth and secure interactions.

During registration:

* The system checks whether the username, email, phone number, or university ID is already in use.
* Required fields are validated before submission.
* Users receive clear messages when input is invalid or unavailable.

During login:

* A generic message is shown if login fails regardless of whether the username or password is wrong.
* This helps protect against account guessing attempts.

Validation logic was tested through the frontend and via Postman, a widely used tool for backend route testing. This approach ensured consistent responses and robustness in error handling.

## 4.9 Environment Setup and Testing Tools

The system was developed and tested locally, with Node.js for the backend and a live browser environment for the frontend. MongoDB Atlas was utilised as the cloud database, allowing for real-time data storage and retrieval during testing.

Key tools included:

* Postman: For testing API endpoints like login, register, and bookings
* Browser dev tools: For testing form validation and frontend functionality
* Console logs and error messages: To debug backend responses.

This environment allowed thorough testing of both isolated backend routes and full system flow. It also provided flexibility to simulate real-world scenarios, including failed logins and duplicate user registration.

# 5.Development and Implementation

This chapter covers the development process for the cloud-based event booking system, including the tools, technologies, and processes used to construct and test the system. It also goes into the steps of implementation, the problems faced, and the final system capabilities. Screenshots from MongoDB Atlas and Postman were used to ensure that the backend API worked properly. Screenshots from the frontend UI are given to show how each main functionality appears visually to the end user.

## 5.1 Tools and Technologies Used

The system was built using a modern full-stack approach that integrates:

* Frontend: HTML, CSS, JavaScript (with interactive elements like form validation, modal popups, event cards, and calendar integration)
* Backend: Node.js with the Express.js framework (RESTful API)
* Database: MongoDB Atlas using Mongoose ODM
* Development Tools: Visual Studio Code (coding), Postman (API testing), Git (version control), MongoDB Atlas (cloud database)
* Authentication/Security: bcryptjs (password hashing), dotenv (environment management), CORS (Cross-Origin Resource Sharing)

Additional frontend components like search bars, login/registration alerts, and conditional display logic were managed using JavaScript and localStorage.

## 5.2 Development Process

The development of the system was conducted in three major phases:

### 5.2.1 Phase 1: Frontend Development (Expanded)

The user interface was developed first to establish the visual structure and client-side functionality of the system. Each screen was created using HTML, CSS, and JavaScript, focusing on responsiveness, validation, and user experience. JavaScript was also used for DOM manipulation and managing session state via localStorage.

Each page and its role in the system are described below:

Login Page (Figure 5.1):  
This screen allows users to enter their username or email and password to access the system. It includes input validation to ensure fields are not left empty and displays real-time error messages for incorrect login attempts. If login is successful, session data is stored in localStorage, and the user is redirected to the . The form interacts with the /api/login backend route using fetch () to verify credentials.

Registration Page (Figure 5.2):  
The registration page collects all user information, including email, username, password, phone number, university ID, and gender. JavaScript is used to validate input before submission, such as checking for a valid email address and matching mandatory fields. When the data is successfully submitted, it is delivered via POST request to the /api/register endpoint, and the user receives success or error feedback.

Support Form (Figure 5.3):  
This page allows users to submit support enquiries. It has a simple form that verifies for empty data and, in a later version, might be linked to an admin dashboard or email notification system. The help website was created with future integration in mind; while backend connectivity was not built, the frontend interface was successfully finished to demonstrate the desired capability.

Forgot Password Page (Figure 5.4):  
Users who have forgotten their credentials can use this page to request a password reset. While this feature is not yet connected to a live backend, the user interface has been fully designed, including input fields for the registered email address and a button for sending reset instructions, preparing the system for future password recovery implementation.

Profile Page (unauthenticated and authenticated) (Figures 5.5 and 6.6):  
The profile page behaves differently based on user status:

* If unauthenticated, the user is shown a warning message and blocked from viewing the page contents. This ensures secure access control by preventing unauthorised access to personal or sensitive data.
* If logged in, the user’s profile information (username, email) is dynamically loaded and displayed. Logged-in users can also create new events via a form. The event creation form sends data to the /api/events endpoint and reloads the event list on success.

DOM manipulation and localStorage are used to control what content is shown depending on login state.

Homepage with Event Carousel (Figure 5.7):  
The homepage features a JavaScript-powered event carousel that dynamically displays forthcoming events from the backend. Events are retrieved via a GET request to the /api/events endpoint and displayed using auto-scrolling capability. The design is mobile-responsive and graphically emphasises featured or newly added events.

Calendar Page (Figure 5.8):  
This website shows a monthly calendar with clickable dates. Events that have been booked or created are highlighted, providing users with a fast snapshot of forthcoming activities. While this version does not include live event dates, the mechanism is in place to incorporate dynamic filtering or categorisation in future releases.

• Registration & Login Validation Feedback (Figures 9–12):  
The system provides real-time feedback for registration and login outcomes:

* Figure 5.9: Displays a message if an email or username is already in use.
* Figure 5.10: Confirms a successful registration with a success message.
* Figure 5.11: Displays an error for incorrect login credentials.
* Figure 5.12: Displays a login success message after authentication.

These validation messages are essential for improving user experience and guiding users through the authentication process.

Events Page with Map (Figure 5.13):  
This page displays all accessible events in card format, as retrieved from the /api/events endpoint. Each event has a "Book Now" button, title, date, and description. Middlesex University's location map helps users understand where activities are taking place. This structure also paves the way for future features like as filters, categories, and real-time booking statistics.

Implementation Notes

All frontend pages were styled using CSS for layout and appearance. JavaScript was used for:

* Form submission and validation.
* Handling API responses
* Access control via login checks in localStorage.
* Dynamic page updates (e.g., profile reloads, event lists)

This approach allowed for a responsive user interface without the use of a frontend framework like React. As a result, navigation, state, and conditional rendering were all managed manually using the DOM.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 5.1: Login page interface where users enter their credentials to access the system.

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure 5.2: Registration page with form fields for user details including email, username, Phone number , and university ID.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 5.3: Support form allowing users to submit queries directly through the system.

A screen shot of a logo

AI-generated content may be incorrect.

Figure 5.4: Forgot password page providing users with the option to reset their login credentials.

A logo on a blue background

AI-generated content may be incorrect.

Figure 5.5: Profile access restriction for unauthenticated users.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.6: Profile page showing authenticated user details and event creation option.

A screenshot of a website

AI-generated content may be incorrect.

Figure 5.7: Homepage carousel displaying upcoming events dynamically.

A screenshot of a calendar

AI-generated content may be incorrect.

Figure 5.8: Calendar view highlighting upcoming event dates.

A screen shot of a login form

AI-generated content may be incorrect.

Figure 5.9: Error message indicating the email is already in use during registration.

A screen shot of a computer screen

AI-generated content may be incorrect.

Figure 5.10: Confirmation message after successful user registration.

A logo with a white square with a blue background

AI-generated content may be incorrect.

Figure 5.11: Login failure message when credentials are incorrect.

A screen shot of a logo

AI-generated content may be incorrect.

Figure 5.12: Login success message confirming user authentication.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.13: Events page showing dynamically loaded event cards and embedded location map.

### 5.2.2 Phase 2: Backend and API implementation

The backend was implemented using Node.js and Express.js, providing a server-side logic layer that communicates with the database and frontend. The core server logic was maintained in server.js, which managed routing, middleware, database connections, and API operations.

Each route was created following RESTful standards and included:

* User Registration: Accepts new user data, validates fields (email, username, phone, university ID), hashes passwords using bcryptjs, and saves data to MongoDB atlas user collection.
* User Login: Authenticates user via either email or username, checks password hash, and returns minimal session-related info.
* Event Creation and Listing: Allows users to create events (title, date, description, createdBy), which are then retrievable via a GET endpoint.
* Booking System: Events can be booked by submitting event information, name, and email. Bookings are stored in the MongoDB Atlas Bookings collection with a timestamp and can be retrieved or deleted through the frontend cancellation feature.

Backend middleware included CORS settings for frontend-backend connection, as well as JSON processing using express.json().

The API was modular and cleanly defined, allowing for simple debugging and extension. Authentication was stateless and frontend-driven, utilising localStorage. Although AWS hosting was initially intended, deployment stayed local due to time constraints and configuration complexity during the project's duration.

Database Design Reference: The architecture of routes and endpoints was inextricably tied to the MongoDB database structure, which was detailed in Phase 3. Screenshots from MongoDB Atlas (e.g., Users, Bookings, and Events collections) are useful for demonstrating how API logic maps to document storage.

### 5.2.3 Phase 3: Database Integration with MongoDB Atlas

The backend functionality of the system was closely integrated with the database via MongoDB Atlas, a managed cloud database solution. All data interactions were handled by Mongoose, which supported structured schema definitions and model-based queries. This connection guaranteed that data flowed consistently and securely between the frontend, backend, and storage levels. Three basic collections were established, each having a distinct function inside the system:

Users Collection:

* Fields: email, username, password (hashed), phone, universityId, gender
* This collection holds all registered user data. Passwords are securely hashed using bcryptjs before being stored.
* User data is retrieved for authentication and displayed on the profile page.
* Figure 5.14 shows the users collection as displayed in MongoDB Atlas.

Events Collection

* Fields: title, description, date, createdBy
* This collection stores all events created by users. Events are dynamically fetched and displayed on the homepage, events page, and user profile.
* Events are created through a POST request and retrieved using a GET request.
* Figure 5.15 presents the stored event documents in the events collection.

Bookings Collection

* Fields: username, email, eventType, eventDate, notes
* This collection captures user bookings for events. Each document stores the booking details and the associated user and event.
* Bookings can be created, fetched, and deleted through dedicated endpoints.
* Figure 5.16 displays the booking records in the MongoDB Atlas interface.

These collections served as the foundation for the system's data model, allowing the application to function properly, from user management to real-time event booking. Their simple yet scalable framework allows them to add new features in the future, such as user roles, event categories, and reporting tools.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.14: Users collection in MongoDB Atlas displaying registered user documents. Note that passwords are securely stored using bcrypt hashing.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.15: Events collection highlighting event documents created by users, including fields such as title, description, date, and creator information.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 5.16: Bookings collection illustrating event booking records linked to users and event types, with timestamps for each submission.

## 5.3 Session Management and Access Control

Login state was managed using the browser's localStorage API instead of token-based or server-side session management. This method was selected for its simplicity and its ability to meet basic access control requirements.

* On successful login, user data (email, username) is stored in localStorage.
* Each protected page checks login state before rendering restricted content.
* If login state is not found, the user is redirected or shown an access warning.
* On logout, session data is cleared from localStorage, and the page is refreshed to ensure all content is reloaded in an unauthenticated state.
* A confirmation popup is also displayed after logout to inform users they have been successfully logged out.

This logout function is a critical feature that was not originally included in the wireframes. By clearing the session and refreshing the page, any cached or residual data is prevented from being accessible, ensuring that the interface behaves safely during the next user session. The popup feedback enhances the user experience by clearly confirming the logout activity.

## 5.4 Navigation and Dynamic Page Updates

As no frontend framework was used, navigation and interactivity were implemented manually with JavaScript:

* Links between pages were implemented using standard anchor tags.
* Content was dynamically updated by DOM manipulation after successful fetch responses.
* Event creation and booking forms trigger real-time page updates upon submission.
* Event cards, profile info, and booking lists are all loaded dynamically based on login state or user action.
* Pop-up modals and form validation messages are shown and hidden using custom JavaScript functions.

All updates and routing are performed without requiring page reloads, creating a seamless user experience despite using basic tools.

## 5.5 Implementation Summary

The final implementation of the system includes:

* A multi-page frontend with secure login, registration, and event features
* A backend server managing all CRUD logic for users, events, and bookings.
* A well-structured database with separate collections for users, events, and bookings
* Lightweight session handling using localStorage.
* DOM-based navigation and interface updates without a framework

Together, these components comprise a full-stack system designed to manage university events in a streamlined and user-friendly manner. Evaluation and testing are covered in the next chapter.

# 6.Testing and Results

This chapter provides a full explanation of the testing procedure used to assure the proper installation and functionality of the cloud-based event booking system. Unlike Chapter 5, which focused on system implementation, this chapter determines if the technological components function properly. Testing took place locally with Postman, MongoDB Atlas, and browser-based validation techniques.

## 6.1 Testing Methodology

A systematic testing process was used to ensure consistent performance and fault handling. All tests were run locally prior to frontend deployment. Backend testing was done with Postman to replicate API calls, and MongoDB Atlas was used to validate database changes. Frontend testing was carried out manually using browser developer tools and console logs.

The cancelling capability was included later in the frontend development process. Due to timing restrictions, the functionality was tested immediately on the frontend and validated by monitoring data changes in MongoDB. Instead of using Postman, which was the primary technique for accessing other API endpoints.

Test tools and environments used:

* Postman: Used for testing API endpoints with different request scenarios to ensure proper backend functionality.
* MongoDB Atlas: Used to confirm successful data insertion, updates, or deletions within the database.
* Browser Console: Used to check for errors and validate form handling through JavaScript in the frontend interface.

## 6.2 Backend API Testing (Postman)

Each API endpoint was tested with multiple data combinations. The following endpoints were tested, along with their corresponding outcomes:

### 6.2.1 User Registration (POST /api/register)

* Function: Accepts new user data, hashes the password, and saves it to MongoDB.
* Result: User successfully added to the database and visible in the "users" collection.
* Screenshot Reference:
  + Figure 6.1: Successful registration test in Postman
  + Figure 6.2: User successfully added to MongoDB collection.

### 6.2.2 User Login (POST /api/login)

* Function: Validates credentials and returns user information.
* Result: Correct login returns a user object; incorrect credentials return an error.
* Screenshot Reference:
  + Figure 6.3: Login success response

### 6.2.3 Event Creation (POST /api/events)

* Function: Allows users to create new events.
* Result: Event saved correctly and appears in the "events" collection.
* Screenshot Reference:
  + Figure 6.4: Event successfully created via Postman.
  + Figure 6.5: Event successfully saved in MongoDB.

### 6.2.4 Get All Events (GET /api/events)

* Function: Fetches all event documents.
* Result: Returned list includes all created events.
* This endpoint was not tested in Postman; instead, its functionality was confirmed through visual verification of dynamic event loading on the events and homepage interfaces.

### 6.2.5 Book Event (POST /api/book)

* Function: Submits a booking request for a selected event.
* Result: Booking saved successfully in the "bookings" collection.
* Screenshot Reference:
  + Figure 6.6: Booking success response in Postman.
  + Figure 6.7: Booking saved in MongoDB Atlas

### 6.2.6 Cancel Booking (DELETE /api/bookings/:id)

* Function: Deletes a booking document from the database.
* This endpoint was excluded from Postman testing due to its late-stage implementation. The cancellation feature was tested via the frontend interface, and its functionality was validated through observed data changes in MongoDB Atlas.

A screenshot of a computer program

AI-generated content may be incorrect.

Figure 6.1: Successful registration request sent via Postman to the /api/register endpoint, with confirmation message and stored user object returned in the JSON response.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6.2: Registered user document as displayed within the user’s collection in MongoDB Atlas, confirming successful data insertion and field structure.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6.3: Login request tested in Postman using valid credentials at the /api/login endpoint. Backend returned a success message and relevant user data in the response.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6.4: POST request to /api/events tested through Postman, resulting in a successful event creation with appropriate response payload.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6.5: Visual representation of the events collection in MongoDB Atlas, showing multiple stored events including title, description, date, and creator.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6.6: Event booking submitted through the /api/book endpoint in Postman. The backend confirmed the booking was saved successfully.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6.7: Booking records shown within the bookings collection in MongoDB Atlas, verifying that event bookings are stored correctly with relevant fields.

## 6.3 Frontend Testing

Manual testing was carried out using the web interface, utilising browser tools and visual feedback systems. JavaScript was utilised to validate data on both the login and signup forms. These validation techniques were personally evaluated to ensure they provided accurate replies to both valid and invalid inputs..

Test examples:

* Registration Form Validation: JavaScript-based validation correctly triggered warnings for empty fields or incorrect input formats (e.g., missing email).
* Login Form Feedback: If incorrect credentials were entered, an error message was displayed without sending a request to the backend.
* Booking Flow: Successfully booking an event updated both the frontend booking list and the backend bookings collection.
* Event Creation: Submitting a new event updated the homepage and events page without a full page reload.
* Cancel Booking: Triggered from the frontend UI, removed the booking from the interface and was confirmed in the MongoDB collection.

These frontend validations complemented backend error handling and helped provide a smooth user experience during testing.

## 6.4 Summary of Results

The testing phase demonstrated that the system is stable, functional, and performs as intended. The following is a summary of the key outcomes observed during testing:

• API Endpoint Verification: All major backend API endpoints, including user registration, login, event creation, and booking, were thoroughly tested using Postman. Each produced the correct response for valid inputs and appropriate error messages for invalid data.

• Database Accuracy: Database Accuracy: Every API activity, such as adding a new user, establishing an event, or making a booking, was checked in MongoDB Atlas. This ensured that data was properly stored, accessed, and destroyed across the appropriate collections.

• Frontend Functional Testing: The user interface responded dynamically to user activities. Event listings were updated once they were created, reservations were reflected right away, and validation messages were presented correctly in response to input problems or backend feedback.

• Cancellation Feature: This capability was included later in the development process, but it was successfully tested on the frontend. When a user cancelled a booking, the change was seen and recorded in the database. Due to time constraints, this was not tested using Postman.

• Edge Case Handling and Error Messages: The system gracefully handled unexpected inputs, avoiding major failures. For example, the registration form prevented users from entering multiple email addresses or usernames, and the login form displayed appropriate errors when credentials were wrong.

• Overall Stability and Integration: No major flaws were discovered during the testing procedure. The coordination between the frontend, backend, and database levels was seamless, demonstrating that the system's components are properly integrated.

Test results show the system's dependability and preparedness for deployment in a real-world university context, with a strong design capable of properly managing user interaction and data under normal settings.

# 7.Conclusion

The development of this cloud-based event booking system has given a comprehensive learning experience that extends beyond achieving functional needs. The project was created to solve real-world issues such as inefficient event administration, multiple bookings, and limited accessibility of event information, particularly in a university context. The finished system met its initial aims by using a full-stack development strategy and iterative testing, including a responsive frontend, a secure backend, and an integrated cloud database to monitor user activity and bookings.

One of the implementation's primary merits was its simplicity and maintainability. Rather of adopting a framework like React, the frontend depended on JavaScript, HTML, and CSS to handle dynamic rendering and user interaction. While this presented certain issues, notably in routing and session management, it also allowed for greater control over logic and user flow. JavaScript's localStorage was utilised to efficiently persist session data, and all page navigation was done manually via the DOM. This design decision enabled the system to remain lightweight, accessible, and simple to alter.

Backend development was another area where critical development occurred. The Node.js and Express.js servers handled all API operations with explicit and modular logic. Each route was constructed with proper validation, error handling, and database interaction using Mongoose. This project represented the first time both Mongoose and MongoDB Atlas were used in the development process; while initially difficult, this component proved to be one of the most gratifying. Understanding Mongoose's schema-based approach and how to make efficient asynchronous queries were tough at first, but familiarity with these tools increased with time. Understanding how to define data schemas, perform asynchronous database operations, and visually evaluate cloud-stored data with Atlas was a significant step forward in my technological development.

MongoDB Atlas proved to be an exceptionally excellent cloud integration database. All collections were available and managed through the web interface, and modifications were validated in real time as API calls were handled. Structuring the collections around users, events, and bookings meant that data was neatly segregated and that all important entities had clear duties. This configuration also makes the system scalable, allowing for future enhancements such as admin user roles, event filtering, and improved reporting capabilities.

Testing was an important element of this project, and it was handled using Postman for backend routes and browser-based validation for frontend operations. While most backend endpoints were successfully tested using Postman, the cancellation functionality had to be manually tested on the frontend owing to time restrictions, as it was implemented late in the development process. Despite this, all important functionality performed as intended, and database entries were validated using MongoDB Atlas. The robustness of the validation mechanism on both the backend and frontend meant that duplicate registrations, empty fields, and incorrect credentials were properly managed.

This project brought about a few obstacles. One was time management, particularly when adding things late in the project. The cancellation API was introduced after frontend development, which constrained the time available for thorough backend testing. Similarly, the logout option was not included in the initial wireframes but was later added to improve the user experience. The logout button removes session data from localStorage, refreshes the page, and shows a message verifying that the user has logged out. Another problem was maintaining uniform styling and behaviour across all pages without the use of a framework. However, overcoming these hurdles led to the development of technical skills and problem-solving ability under real-world deadlines.

Some constraints were also discovered during development. Features including full AWS deployment, interactive calendar integration, and help page functionality were initially intended but had to be postponed owing to time restrictions and the necessity to master unknown technologies. While the system's current version achieves its primary goals, these limitations suggest areas for future development and refinement.

In conclusion, the project met all of the essential objectives while also providing significant insights into full-stack web development. It gave hands-on expertise with database integration, frontend validation, API design, and testing methods. More importantly, the project demonstrated the value of preparation, adaptability, and user-centred design. The end product is a practical and scalable platform that may be further developed for real-world applications, particularly in educational or institutional settings. Future work could involve deploying to cloud platforms such as AWS or Render, adding user roles and admin dashboards, and implementing real-time capabilities via WebSockets or email notifications. Overall, the system met its primary goals as specified at the start of the project, delivering a functional, user-friendly, and scalable solution that may serve as the foundation for future growth.

# 8. Evaluation and Discussion

While the event booking system fulfilled its essential functions, various improvements might be done to make the platform more comprehensive, user-friendly, and scalable. This chapter discusses features that were originally planned but were not implemented owing to time restrictions or the difficulty of unknown technology, as well as potential future enhancements to the system.

## 8.1 Limitations

While the system successfully met its primary objectives, certain limitations affected its overall implementation and scope:

* Deployment Limitations: The system was not fully deployed to the cloud. Although MongoDB Atlas was used for cloud-hosted storage, the frontend and backend were hosted locally due to time constraints and unfamiliarity with AWS configuration.
* Security Gaps: Advanced security measures such as SSL encryption, role-based access control, and AES encryption were planned but not implemented due to limited experience and project scope.
* Feature Incompleteness: Functionalities like the calendar page, support page, and forgot password mechanism exist only as UI components without backend logic.
* Late Integration: Some features such as cancellation and logout were implemented late, which limited the time for thorough backend testing.
* Frontend Scalability: The use of JavaScript rather than a modern frontend framework made routing and state management more complex, reducing long-term scalability.

These limitations provide a clear understanding of the system's current boundaries and serve as a foundation for future improvement.

## 8.2 Originally Planned Features

The following features were part of the original vision for the system, as outlined in the original system specification and design documentation. However, these were not completed due to time constraints and the complexity of modern technologies during development.

### 8.2.1 Full AWS Deployment

* Amazon EC2 was intended to host the backend server, providing high availability, scalability, and flexibility.
* Amazon S3 was planned for hosting the frontend, ensuring global access and seamless static content delivery.
* AWS RDS would have managed the database instead of MongoDB Atlas, offering relational data structure options.
* AWS Lambda was considered for serverless execution of backend functions like event creation and booking.

These services were part of the rollout strategy to make the system completely cloud-based. However, due to the complexity of setting up cloud infrastructure and AWS's high learning curve, implementing these functionalities was not viable within the project deadline. As a result, the system was tested and run locally, with the exception of MongoDB Atlas, which was cloud-based.

### 8.2.2 Security Measures

* SSL encryption was proposed to secure data transmission.
* AES encryption was planned for storing sensitive user data securely.
* Role-Based Access Control (RBAC) would have limited access based on user roles (admin, student, etc.).
* Multi-Factor Authentication (MFA) was intended as a future addition to enhance login security.

Due to time constraints and lack of experience with advanced security protocols, these features were not developed. However, they remain important considerations for future implementation.

### 8.2.3 Support Page Functionality

* A support form was included in the wireframes and developed as a frontend page.
* The plan was to save user queries into a "support" collection in MongoDB.
* Admin users would view, manage, and respond to tickets via a future admin dashboard.

Although the support page UI was developed, backend integration remains pending.

### 8.2.4 Interactive Calendar

* A monthly calendar interface was designed to display upcoming events.
* Clicking "Learn More" would direct the user to the event’s detailed view.

The layout for this feature exists, but the calendar is not functional and does not yet integrate with dynamic event data.

### 8.2.5 Editable User Profile with Picture and Bio

* A user dashboard was designed to allow editing of profile picture and bio/about section.
* The profile page currently only displays static user details like email and username.

Time limitations prevented the implementation of this editable feature.

### 8.2.6 Forgot Password Functionality

* A password reset form was designed in the frontend.
* The intended functionality was to generate a secure token and send a password reset link via email.

The backend logic for generating and handling password reset tokens was not completed.

## 8.3 Additional Future Enhancements

Aside from the features that were originally planned, the following suggestions have been identified to improve the system further.

### 8.3.1 Event Capacity Management

* Admin or event creators could define a maximum number of attendees.
* Remaining seats could be displayed on each event card or booking page.
* The booking logic would update capacity in real-time.
* Once an event is full, the "Book Now" button would be disabled.
* If a user cancels, capacity will automatically increase.

This feature would be especially beneficial for room-limited events and reduce administrative burden.

### 8.3.2 Email Notification System

* Send confirmation emails when a user registers or books an event.
* Send reminder emails the day before the event.
* Notify users about cancellations or event changes.
* Use services like NodeMailer or SendGrid to automate communication.

Email communication would improve professionalism and reduce no-shows.

### 8.3.3 Email Domain Verification

* Ensure users register only with their Middlesex email (e.g., ending in @mdx.ac.uk).
* Reject public domain emails like Gmail or Yahoo.
* Display validation messages to inform users.

This would help ensure that platform access is limited to verified university members.

### 8.3.4 Event Search and Filtering

* Implement a search bar on the events page.
* Introduce category-based filters (e.g., "sports", "careers", "social").
* Add a date filter to display upcoming events in specific ranges.

These features would improve usability and help users find relevant events easily.

### 8.3.5 Live Calendar View

* Show all upcoming events using an interactive calendar grid.
* Booked events would appear in a distinct colour.
* Clicking on a calendar date could show a list of that day’s events.
* Booking and cancellation actions could be managed directly through the calendar.

This would offer users a high-level overview of their schedules and simplify event planning.

### 8.3.6 Admin Dashboard and Analytics

* Add role-based access for admin and moderators.
* View charts of event participation, cancellations, and user registrations.
* Manage users, events, support queries, and system settings.

This would centralise management and improve reporting.

### 8.3.7 Real-Time Updates with WebSockets

* Notify users in real-time when new events are added.
* Automatically update the booking counter without page refresh.
* Alert users of last-minute event changes or slot availability.

Real-time features would create a dynamic, modern web experience.

### 8.3.8 Frontend Styling Enhancements

* Refactor and improve CSS for a more polished and professional appearance.
* Use consistent colours, spacing, and UI patterns across all pages.
* Apply hover effects, responsive design tweaks, and custom icons.
* Enhance the interface to be modern, visually consistent, and user-friendly.

Together, these features provide a clear roadmap for future development. With extended development time and further technical advancement, the system could evolve into a fully cloud-based, secure, and feature-rich university event platform ready for real-world deployment.

# 9.References

1. Baiskar, Y., Paulzagade, P., Koradia, K., Ingole, P. & Shirbhate, D. (2022) ‘MERN: A Full-Stack Development’, International Journal for Research in Applied Science & Engineering Technology (IJRASET), 10(1), pp. 2321–9653. Available at: <https://www.ijraset.com/research-paper/mern-full-stack-development> (Accessed: 6 February 2025).
2. mrth, A., Forsberg, B. & Benini, L. (2022) ‘HEROv2: Full-Stack Open-Source Research Platform for Heterogeneous Computing’, arXiv preprint. Available at: <https://arxiv.org/abs/2201.03861> (Accessed: 10 February 2025).
3. Petrikoglou, A. & Kaskalis, T.H. (2019) ‘Full Stack Web Development Teaching: Current Status and a New Approach’, Proceedings of the 11th International Conference on Computer Supported Education (CSEDU). Available at: <https://www.scitepress.org/Papers/2019/80662/80662.pdf> (Accessed: 2 February 2025).
4. Pisu, L., Maiorca, D. & Giacinto, G. (2024) ‘A Survey of the Overlooked Dangers of Template Engines’, arXiv preprint. Available at: <https://arxiv.org/abs/2405.01118> (Accessed: 9 February 2025).
5. Rajan, A.T., Aravind, S., Shukla, S., & Gupta, R.K. (2024) ‘Leveraging AWS Full Stack Development Platform for Scalable and Reliable Enterprise Applications’, International Journal of Intelligent Systems and Applications in Engineering, 12(17s), pp. 830–. Available at: <https://www.ijisae.org/index.php/IJISAE/article/view/6930> (Accessed: 27 January 2025).
6. Sannakki, S.S. (2022) ‘Full Stack Web Development: Vision, Challenges and Future Scope’, International Research Journal of Engineering and Technology (IRJET), 9(4), pp. 2395–0072. Available at: <https://www.irjet.net/archives/V9/i4/IRJET-V9I4398.pdf> (Accessed: 3 February 2025).
7. Sipek, M., Muharemagic, D., Mihaljevic, B., & Radovan, A. (2021) ‘Next-generation Web Applications with WebAssembly and TruffleWasm’, arXiv preprint. Available at: <https://arxiv.org/abs/2112.08270> (Accessed: 23 February 2025).

# 10.Appendices

## 10.1 Command Line Instructions

How to Run the Backend Server:

To start the backend server locally, follow these steps in your terminal:

cd Backend

node server.js

This command starts the Express server and connects to MongoDB Atlas.

## 10.2 Installed Node.js Packages

Below is a list of all the Node packages installed and used for this project:

| Package | Version | Purpose |
| --- | --- | --- |
| bcryptjs | 3.0.2 | For securely hashing user passwords before storing in the database |
| cors | 2.8.5 | To enable Cross-Origin Resource Sharing between frontend and backend |
| dotenv | 16.4.7 | To load environment variables from a .env file securely |
| express | 4.21.2 | Web framework used to create API routes and manage server logic |
| Mongoose | 8.13.1 | ODM (Object Data Modelling) library used to interact with MongoDB Atlas |

## 10.3 Project Demonstration Video

A video demonstration of the cloud-based event booking system can be accessed at the following link:  
YouTube: <https://youtu.be/C_5wb7Pu1X4>

This video walks through the core features of the system.

## 10.4 Ethics Form

Ethics Approval Form:

A close-up of a questionnaire

AI-generated content may be incorrect.

A paper with text on it

AI-generated content may be incorrect.

A screenshot of a application form

AI-generated content may be incorrect.