

**SCHOOL OF ARCHITECTURE, COMPUTING & ENGINEERING**

**Department of Computer Science and Digital Technologies – CDT**

CN7021 – Advanced Software Engineering

**SCHOOL ACTIVITY BOOKING SYSTEM - ACADEMIC REPORT**

Group: **3.B**

Students Name & ID:

**Sanchit Kaushal (2823183)**

**Chichebendu Blessed Umeh (2823112)**

**Mohd Sharjeel Mohd Saquib Khan (2823311)**

**Shiva Kasula (2822121)**

Tutor: Manasa Yegamati

Module Leader: **Dr Hisham AbouGrad**

**December 2025**

Table of Contents

[1. Introduction 3](#_Toc217284536)

[1.1 Purpose 3](#_Toc217284537)

[1.2 Document Conventions 3](#_Toc217284538)

[1.3 Intended Audience and Reading Suggestions 3](#_Toc217284539)

[1.4 Product Scope 3](#_Toc217284540)

[1.5 Requirements Reference Documents 4](#_Toc217284541)

[2. Software Project Description, Methodology, and Methods 5](#_Toc217284542)

[2.1 Software Product Purpose, Functions, and Use Cases 5](#_Toc217284543)

[2.2 User and Stakeholders 6](#_Toc217284544)

[2.3 Operating Environment 6](#_Toc217284545)

[2.4 Software Architecture and Methodology 6](#_Toc217284546)

[2.5 Design and Implementation Constraints 8](#_Toc217284547)

[2.5.1 System Design 8](#_Toc217284548)

[2.5.2 Implementation and System Development 8](#_Toc217284549)

[2.6 User Documentation 10](#_Toc217284550)

[2.7 Assumptions and Dependencies 10](#_Toc217284551)

[3. External Interface Requirements 10](#_Toc217284552)

[3.1 User Interfaces 10](#_Toc217284553)

[3.2 Hardware Interfaces 12](#_Toc217284554)

[3.3 Software Interfaces 13](#_Toc217284555)

[3.4 Communication Interfaces 13](#_Toc217284556)

[4. System Functional Requirements 13](#_Toc217284557)

[4.1 Parent Registration & Authentication: F1 13](#_Toc217284558)

[4.2 Activity Booking Management: F2 15](#_Toc217284559)

[5. User Stories and Scenarios 24](#_Toc217284560)

[6. System Nonfunctional Requirements 37](#_Toc217284561)

[6.1 Performance Requirements 37](#_Toc217284562)

[6.2 Safety Requirements 37](#_Toc217284563)

[6.3 Security Requirements 37](#_Toc217284564)

[6.4 Software Quality Attributes 37](#_Toc217284565)

[6.5 Other External Requirements 37](#_Toc217284566)

[6.6 Other External Requirements 37](#_Toc217284567)

[7. Software Testing and Test Plan 37](#_Toc217284568)

[7.1 Test Suite T1: Functional Requirements Validation 38](#_Toc217284569)

[7.2 Test Requirement NF1: Nonfunctional Requirements Verification 39](#_Toc217284570)

[8. Project Management 41](#_Toc217284571)

[8.1 Effort Calculation (Using the COCOMO Model) 41](#_Toc217284572)

[8.2 Operational & Infrastructure Budget (Annual) 42](#_Toc217284573)

[8.3 Project Management Board (Notion) 43](#_Toc217284574)

[44](#_Toc217284575)

[9. References 45](#_Toc217284576)

[10. Appendices 46](#_Toc217284577)

[10.1 Appendix A: Contributions Table 46](#_Toc217284578)

[10.2 Appendix B: Agreement of Participation 48](#_Toc217284579)

[10.3 Appendix C: Glossary 49](#_Toc217284580)

[10.4 Appendix D: Analysis and Design Models 49](#_Toc217284581)

[10.5 Appendix E: To Do List 51](#_Toc217284582)

[10.6 Appendix F: Source Code 51](#_Toc217284583)

# Introduction

## Purpose

The product specified in this document is the **School Activity Booking System, Version 1.0**. This SRS covers the **entire software system**, including web interface, database, and reporting. The system replaces manual processes with a digital platform for parents, tutors, and admins.

Key Objectives: -

* Providing a **centralized platforms** for booking activities.
* Enabling **role-based access control (RBAC)**.
* Automating **capacity management** to prevent double-bookings.

## Document Conventions

This report is based on IEEE SRS norms and scholarly rules to specify technical terms to all the interested parties. There is a High-Medium-Low priority system of requirements. Database schemas are written with standard notation, and the code is written with PEP 8. Formatting and organisation of headings are followed, and all external references are cited in Harvard.

## Intended Audience and Reading Suggestions

This specification is significant to all the stakeholders in the software lifecycle. Project assessors and tutors should read all sections to evaluate compliance with the coursework brief. Developers will find Section 2.5 (Design) and Section 3 (Interfaces) critical for implementation details, while testers should focus on Section 7 (Testing) for validation strategies. The document is structured to allow both chronological reading for a general overview and job-based navigation for technical specifics.

## Product Scope

The School Activity Booking System provides a centralized platform for managing after-school activities, enhancing parent engagement and reducing administrative overhead. The system automates booking workflows, enforces capacity constraints to prevent double-booking, and generates automated PDF invoices. Built with Flask 3.0, SQLAlchemy ORM, and Bootstrap 5, the application ensures scalable, transparent management of activities, bookings, and participant records.

## Requirements Reference Documents

The following documents and resources serve as the authoritative references for this Software Requirements Specification:

A.  Source Code Repository

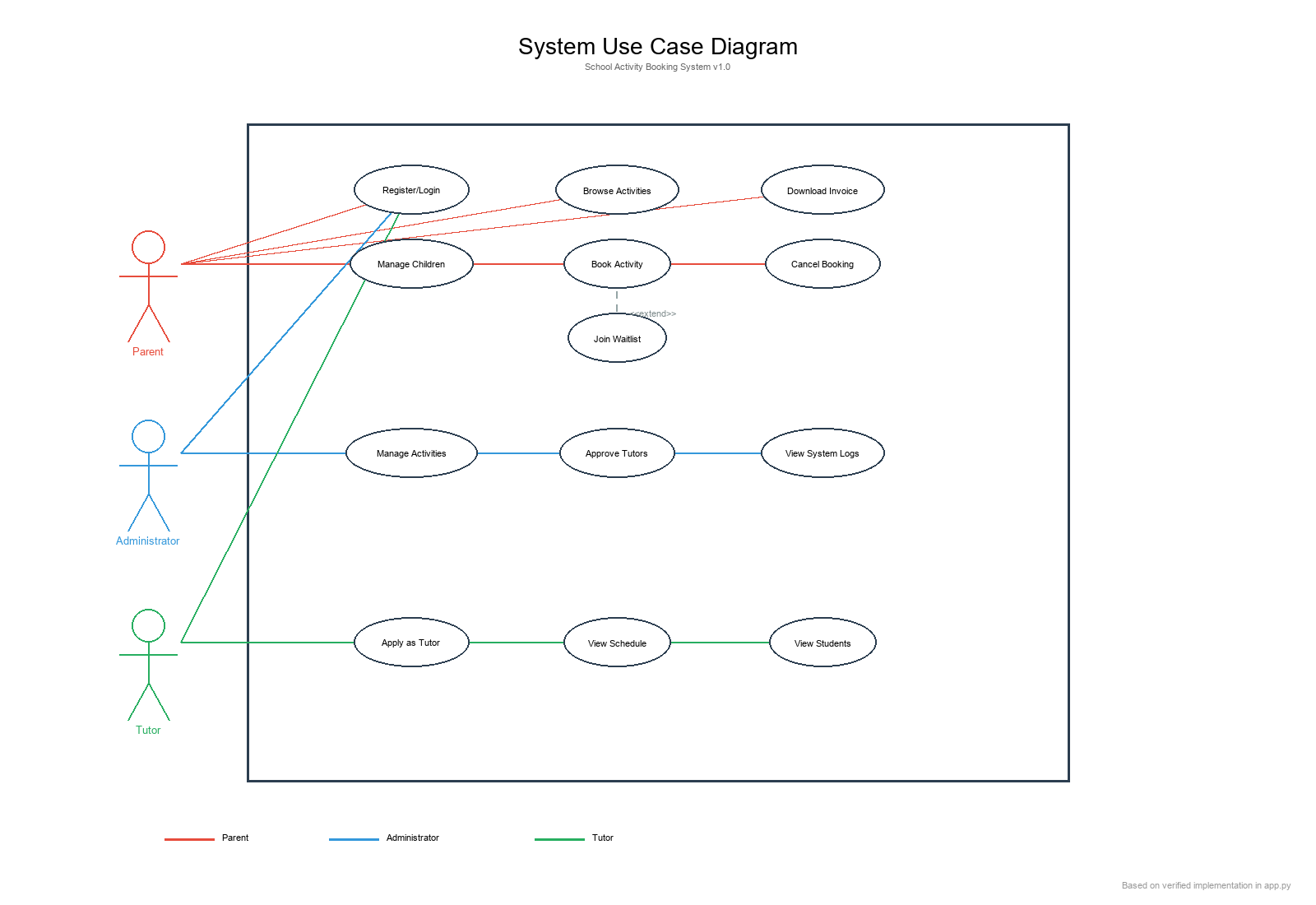
* **Title:** School Activity Booking System
* **Location:** GitHub Repository (https://github.com/sanchitmahant/School-Activity-Booking-System)
* **Version:** v1.0 (Released Dec 2025)

B.  Standards & Style Guides

* **IEEE Std 830-1998:** IEEE Recommended Practice for Software Requirements Specifications. New York: IEEE.
* **Python PEP 8:** Style Guide for Python Code. (Van Rossum & Warsaw, 2013).
* **Bootstrap 5.3 Docs:** User Interface Style Guide. (https://getbootstrap.com/docs/5.3/)

# Software Project Description, Methodology, and Methods

## 2.1 Software Product Purpose, Functions, and Use Cases

****

**Figure 1: System Use Case Diagram**

**(Source: Developed Using Python and Flask Framework)**

The School Activity Booking System allows parents to manage participation in after-school activities easily and securely with authentication, child profile management, activity browsing, and a conflict-free booking system. Database restrictions prevent double-booking, and administrators organize activities and capacities effectively. The automated PDF invoices provide complete booking, child, and cost information to ensure proper documentation. Parent, Administrator, and Tutor are identified in the use case diagram in terms of authentication, child management, activity exploration, booking, invoicing, tutor application, and system administration, focusing on the dependency of processes, automatic invoice generation, and waitlist management based on capacity validation.

## 2.2 User and Stakeholders

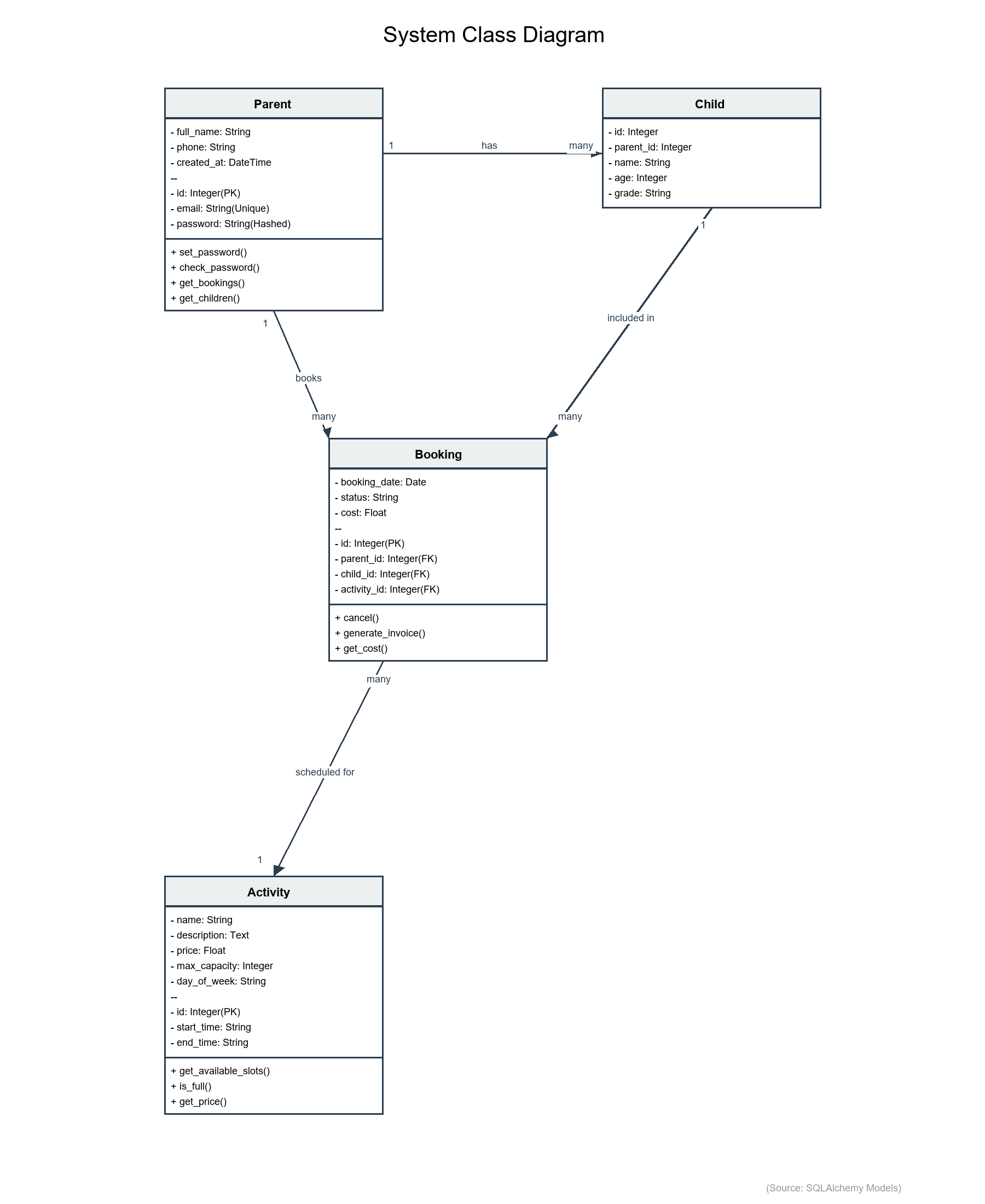
The system serves three primary stakeholder groups. **Parents** represent the primary user base, managing child profiles and activity bookings with basic technical proficiency. **Administrators** function as system managers, requiring advanced technical skills to oversee activity catalogs, approve tutor applications, and monitor system operations. **Tutors** operate as service providers, utilizing intermediate technical capabilities to access their teaching schedules and view enrolled student information.

## 2.3 Operating Environment

The application operates in a **cloud-native environment** optimized for Platform-as-a-Service (PaaS) deployment while maintaining portability for local development. Client-side requirements include any modern web browser (Chrome, Edge, Safari, Firefox) with JavaScript enabled. The server environment utilizes Python 3.12+ running Flask 3.0, with PostgreSQL 16 for production databases and SQLite for development/testing. Deployment targets include and Heroku PaaS platforms, with Docker containerization support for consistent cross-platform execution.

## 2.4 Software Architecture and Methodology

The project follows Agile methodology with iterative development cycles, prioritizing core booking functionality before implementing notification and waitlist subsystems. The architecture implements a Model-View-Template (MVT) pattern via Flask, with clear separation between data models (SQLAlchemy ORM), business logic (route handlers), and presentation (Jinja2 templates). Key architectural components include the Parent, Child, Activity, Booking, Waitlist, Admin, and Tutor models, interconnected through normalized database relationships ensuring referential integrity and cascade operations for dependent records.



**Figure 2: System Architecture Class Diagram**

**(Source: Based on SQLAlchemy Models in app.py)**

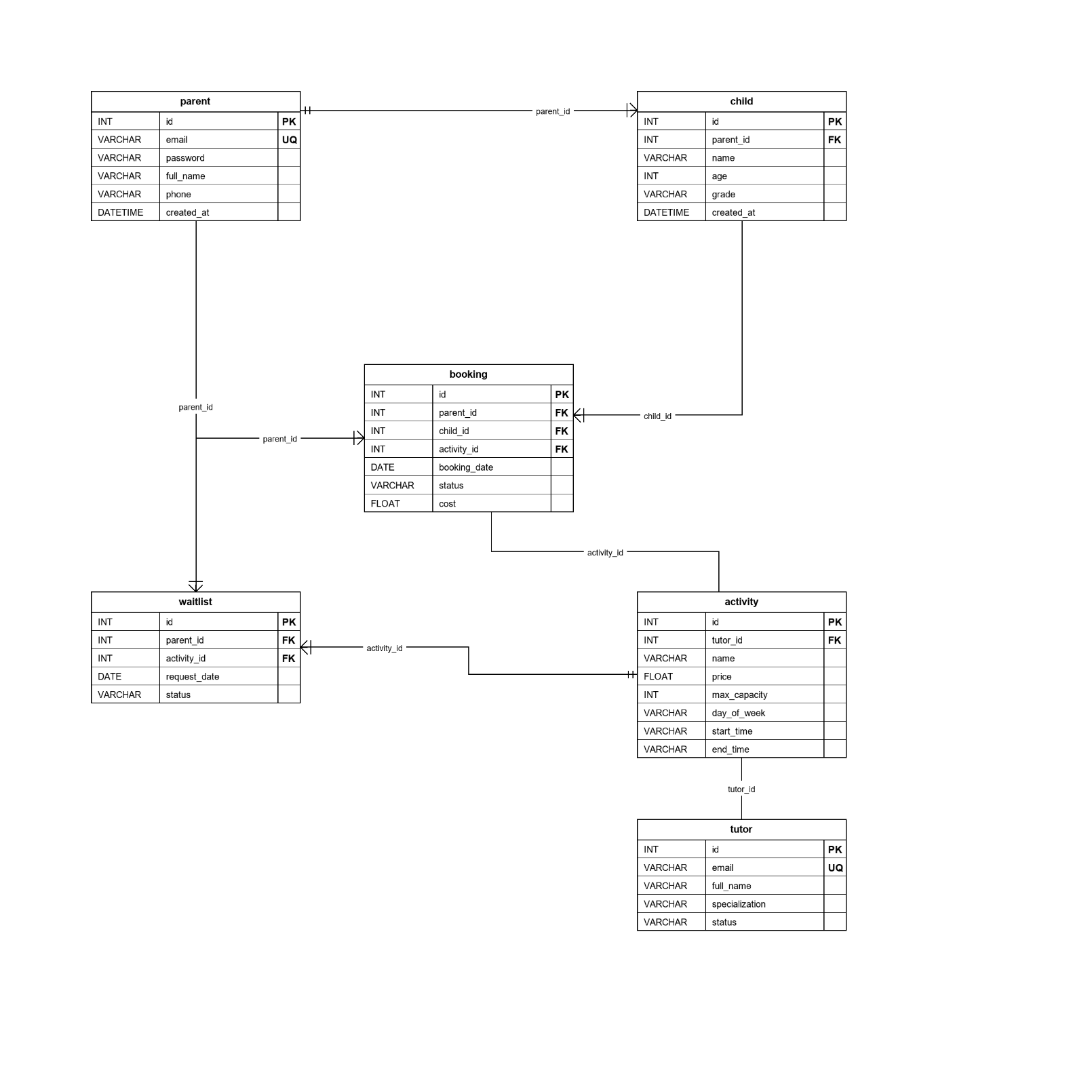
## 2.5 Design and Implementation Constraints

### 2.5.1 System Design

The system adheres to strict security, scalability, and maintainability standards. Data integrity is enforced through a Third Normal Form (3NF) PostgreSQL database schema utilizing foreign keys and cascading deletes. Authentication security is implemented using Werkzeug's secure password hashing (scrypt) and httpOnly session cookies to prevent XSS. To prevent double-bookings, database-level unique constraints are applied to the Booking entity (child\_id + booking\_date). The interface utilizes a responsive Bootstrap 5 grid system to adapt fluidly to desktop, tablet, and mobile (>=320px) viewports. Performance objectives include maintaining page load times under two seconds and database query execution times under 100 milliseconds.

### 2.5.2 Implementation and System Development

The application utilizes a robust relational database schema implemented via SQLAlchemy ORM in Python. Key entities include Parent, Child, Activity, Booking, Waitlist, Tutor, and Attendance, ensuring comprehensive data management for all stakeholders. The Booking entity enforces a unique constraint (child\_id + booking\_date) to strictly prevent double-bookings at the database level. Developing on Flask 3.0 allows for a modular Model-View-Template (MVT) architecture, where business logic is decoupled from data models. The system supports Role-Based Access Control (RBAC) through distinct Admin and Tutor models, securitized with password hashing and session management.



**Figure 3: Database Entity-Relationship Diagram**

**(Source: Database Design Documentation)**

The application is developed using Python 3.12 and the Flask 3.0 web framework, chosen for its lightweight and modular MVT architecture. Data persistence is managed by SQLAlchemy 2.0 ORM, ensuring database-agnostic code compatible with both SQLite (dev) and PostgreSQL (prod). The user interface is built with HTML5/CSS3 and Bootstrap 5.3 components. PDF generation for invoices uses the ReportLab library. The development workflow follows industry best practices including PEP 8 style guidelines, Git version control, and comprehensive unit testing to ensure code quality.

## 2.6 User Documentation

Comprehensive documentation includes a README.md providing step-by-step installation instructions for non-technical administrators. In-application help features contextual tooltips on complex forms, particularly for booking conflict resolution and capacity management. An automated SETUP\_AND\_RUN.bat script streamlines Windows deployment by handling dependency installation and database initialization. System dependencies require internet connectivity for initial pip package installation and Google Fonts CDN resources during runtime.

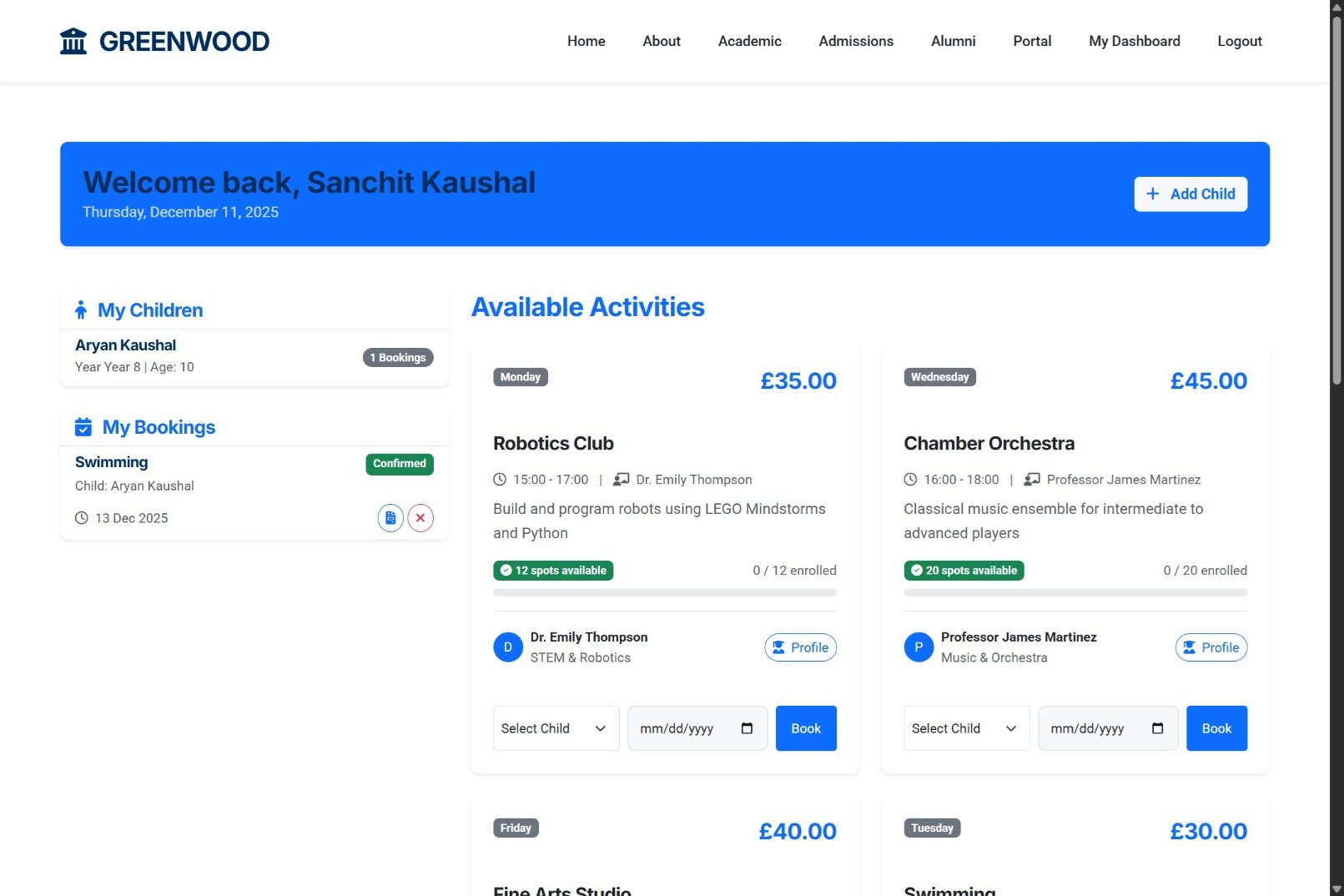
## 2.7 Assumptions and Dependencies

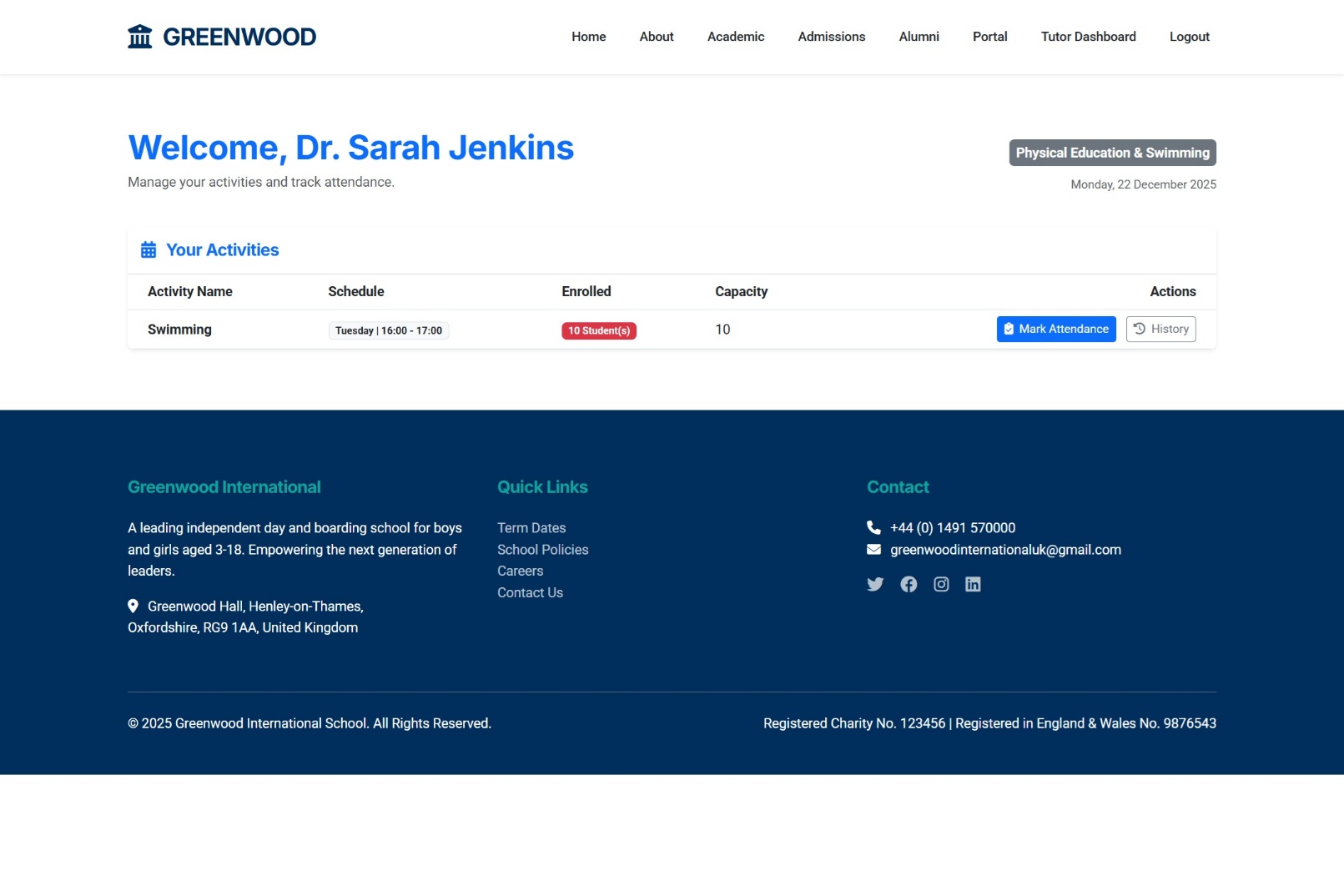
The system assumes reliable internet connectivity for client browsers to access cloud-hosted instances and load external resources. User authentication depends on valid email addresses for account recovery and notification delivery. Booking operations assume accurate system clock synchronization to prevent timestamp conflicts in concurrent booking scenarios. External dependencies include SMTP mail servers for automated notifications, PostgreSQL database services for production data persistence, and browser compatibility with HTML5, CSS3, and ECMAScript 6 standards.

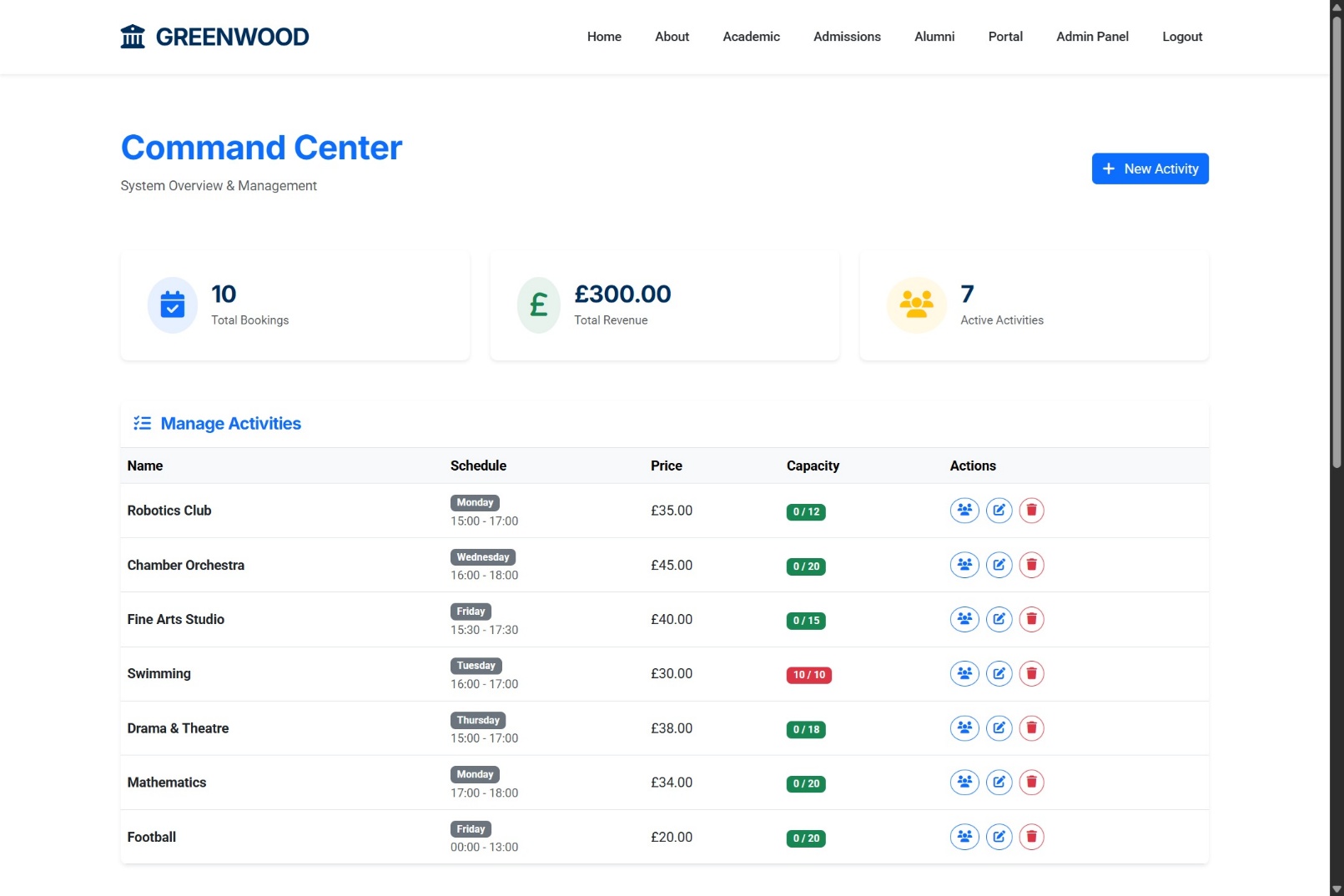
# External Interface Requirements

## User Interfaces

The interface follows a mobile-first design philosophy using Bootstrap 5.3, ensuring responsive rendering across screen sizes from 320px (mobile) to 1920px+ (desktop). Navigation employs a persistent top bar with role-adaptive menu items (guest users see Login/Register, authenticated users access Dashboard/Logout). User feedback mechanisms include Bootstrap Alert components for success/error messages and real-time form validation with inline error indicators. Accessibility compliance targets WCAG 2.1 Level AA standards through semantic HTML, ARIA attributes, and sufficient color contrast ratios (minimum 4.5:1 for normal text).







**Figure 4: Parent, Tutor and Admin Dashboard Screenshot**

**(Source: Application Interface Implementation)**

****

**Figure 5: Public Landing Page**

**(Source: Application Interface Implementation)**

## Hardware Interfaces

Client hardware requirements specify any device capable of executing modern web browsers, with 4GB RAM recommended for optimal PDF rendering performance. Server-side specifications include minimum 1 vCPU at 2.0 GHz, 512MB RAM (1GB recommended for production loads), and 500MB storage for application code and database files. Network bandwidth requirements assume minimum 1 Mbps for standard operations, with higher throughput beneficial for concurrent user scenarios and large PDF downloads.

## Software Interfaces

The application integrates with PostgreSQL 16 via the psycopg2-binary driver for production database operations, while SQLite serves development environments through Python's built-in sqlite3 module. Web server interfacing occurs through Werkzeug (development) or Gunicorn (production WSGI server). Critical library dependencies include Flask-SQLAlchemy for ORM abstraction, Flask-Mail for SMTP integration, and ReportLab for programmatic PDF generation. Browser compatibility requires HTML5, CSS3, and JavaScript ES6+ support for client-side functionality.

## Communication Interfaces

Client-server communication utilizes HTTP/1.1 for development environments and HTTPS with TLS 1.2+ for production deployments to ensure data confidentiality. Data interchange formats include HTML5/CSS3 for rendered pages, JSON for AJAX API responses (booking availability checks, dynamic form updates), and multipart/form-data for file uploads. Email notifications employ SMTP protocol over ports 587 (STARTTLS) or 465 (SSL/TLS) for registration confirmations, booking receipts, and waitlist promotion alerts.

# System Functional Requirements

The functional requirements are organized by the core system features: Parent Registration and Activity Booking.

## Parent Registration & Authentication: F1

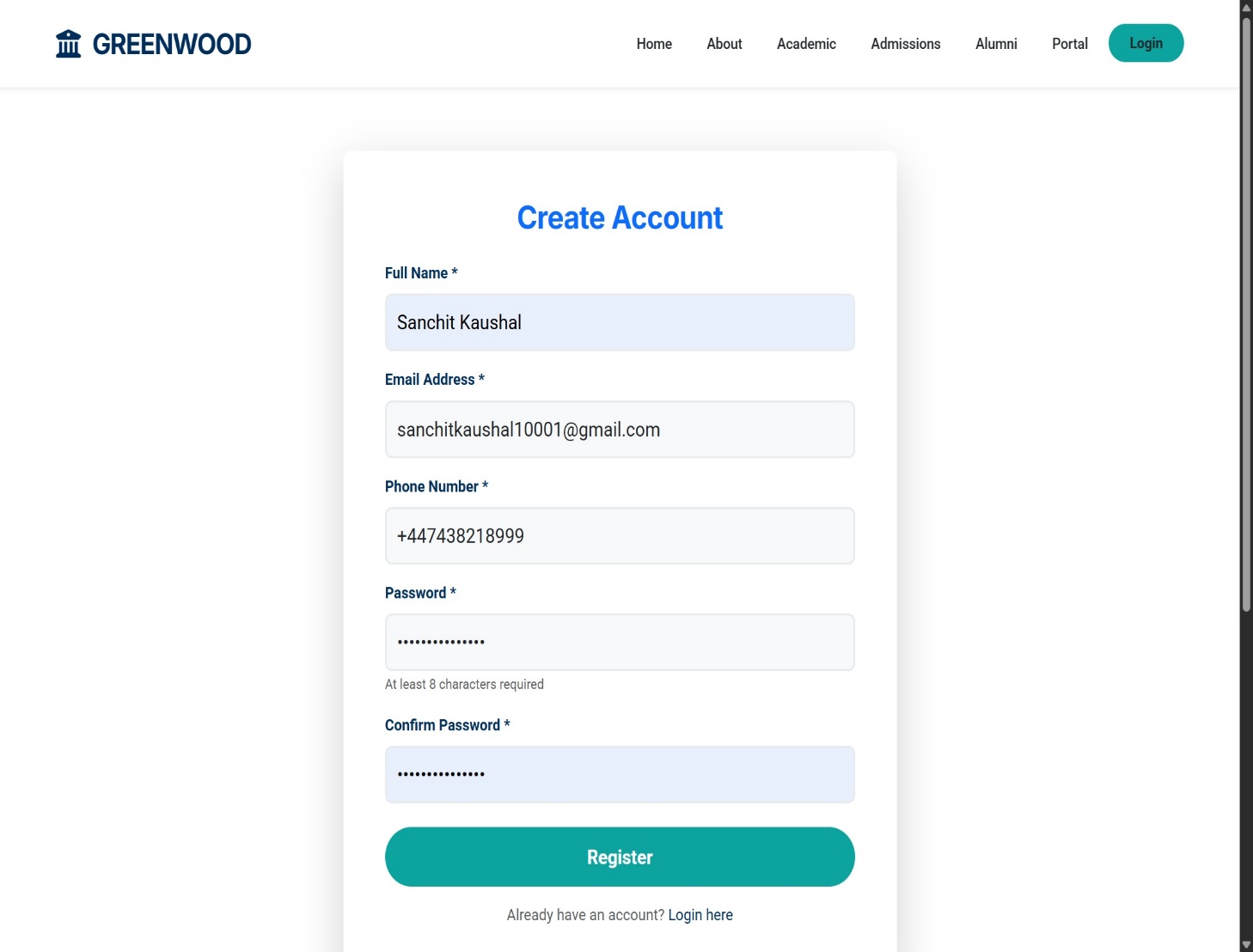
Description and Priority: -

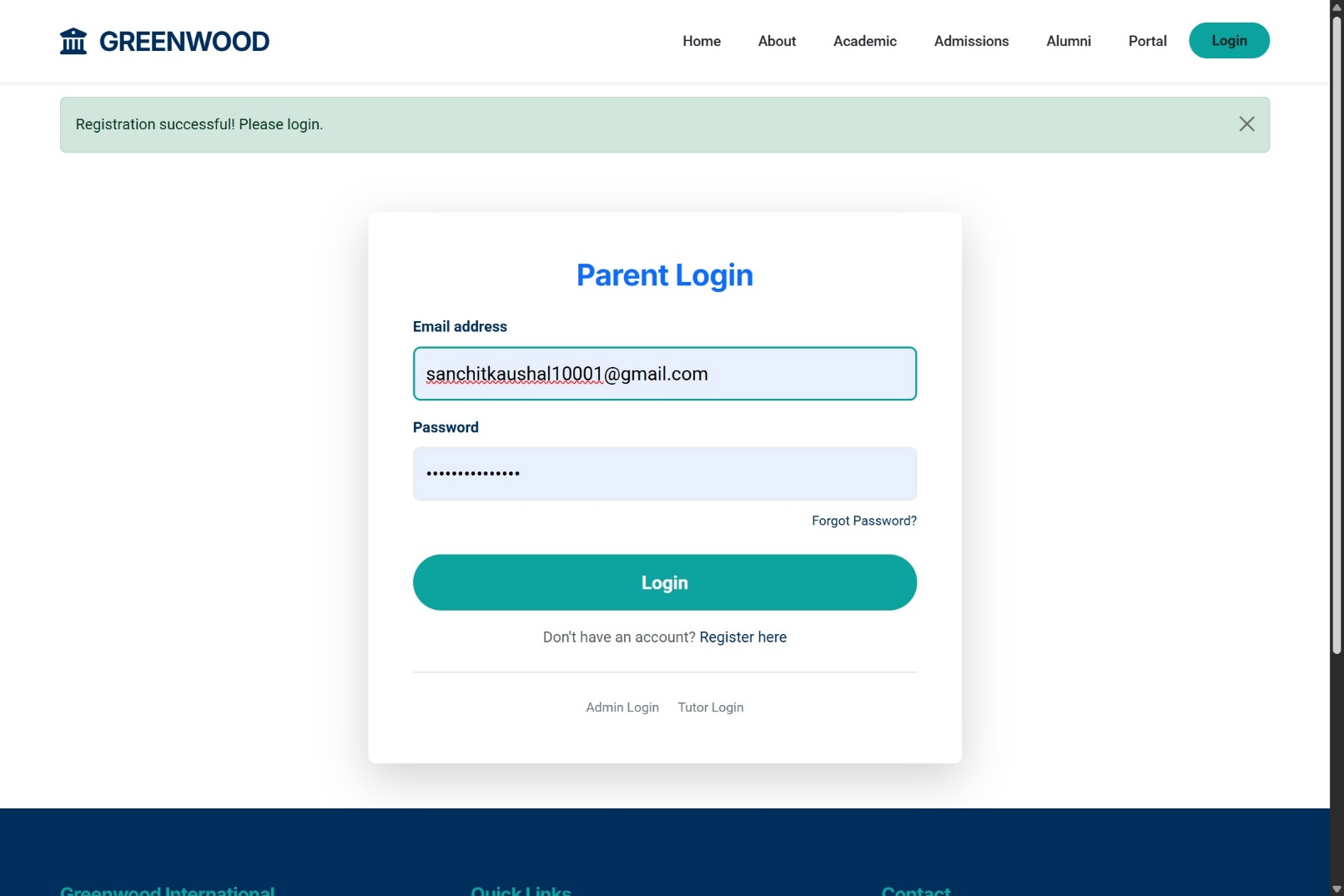
Allows parents to create accounts and log in securely.

Priority: High (Essential for system access).

Input/Output Sequences: -

* Registration: User submits details -> System validates, hashes password, create account.
* Login: User logs in -> System creates session, redirects to Dashboard.





**Figure 6: Registration and Login Interface**

**(Source: Application Interface)**

Functional Requirements: -

* F1.1: Validate email format and uniqueness in the database.
* F1.2: Securely hash passwords (scrypt) before storage.
* F1.3: Enforce 30-minute session timeouts.

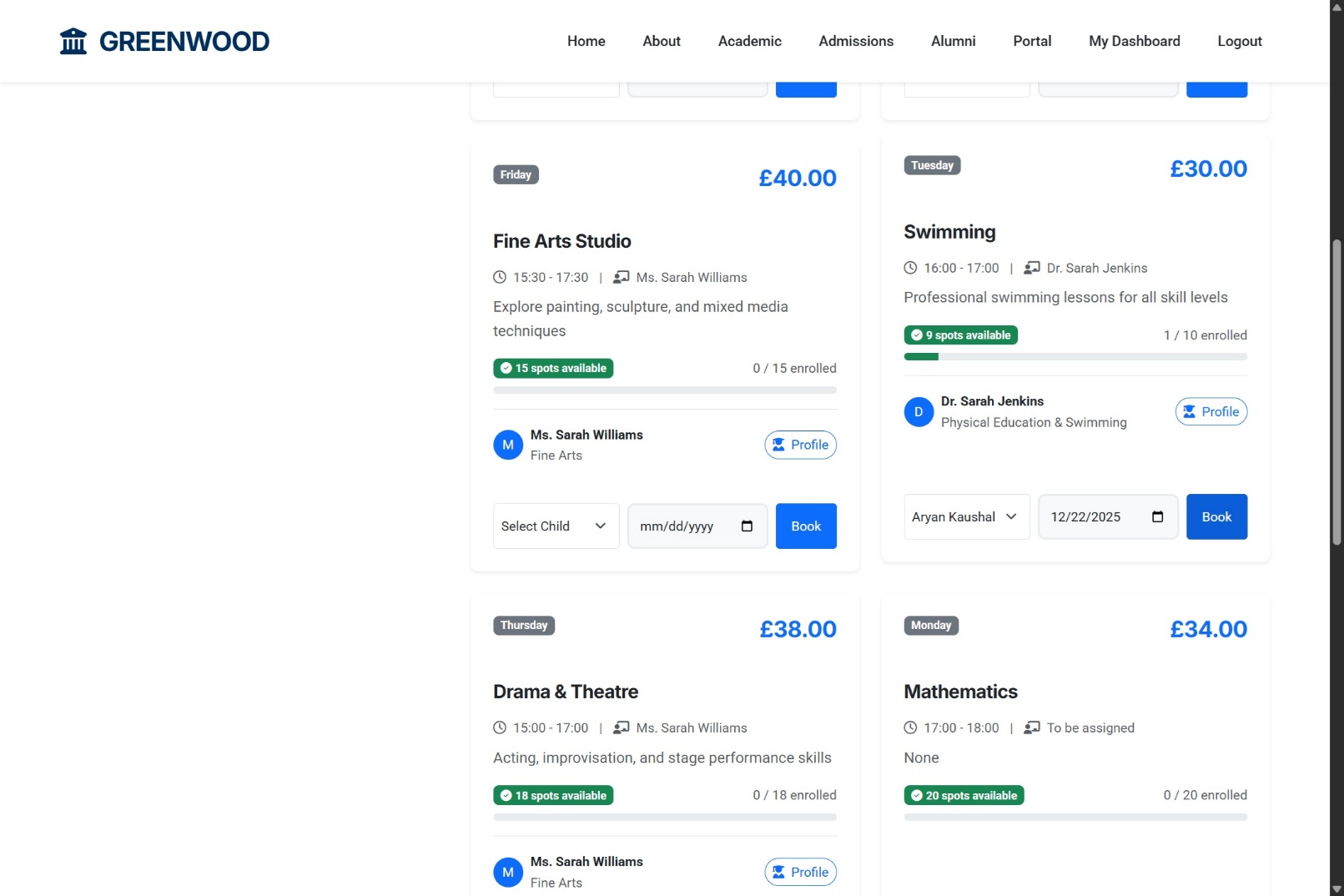
## Activity Booking Management: F2

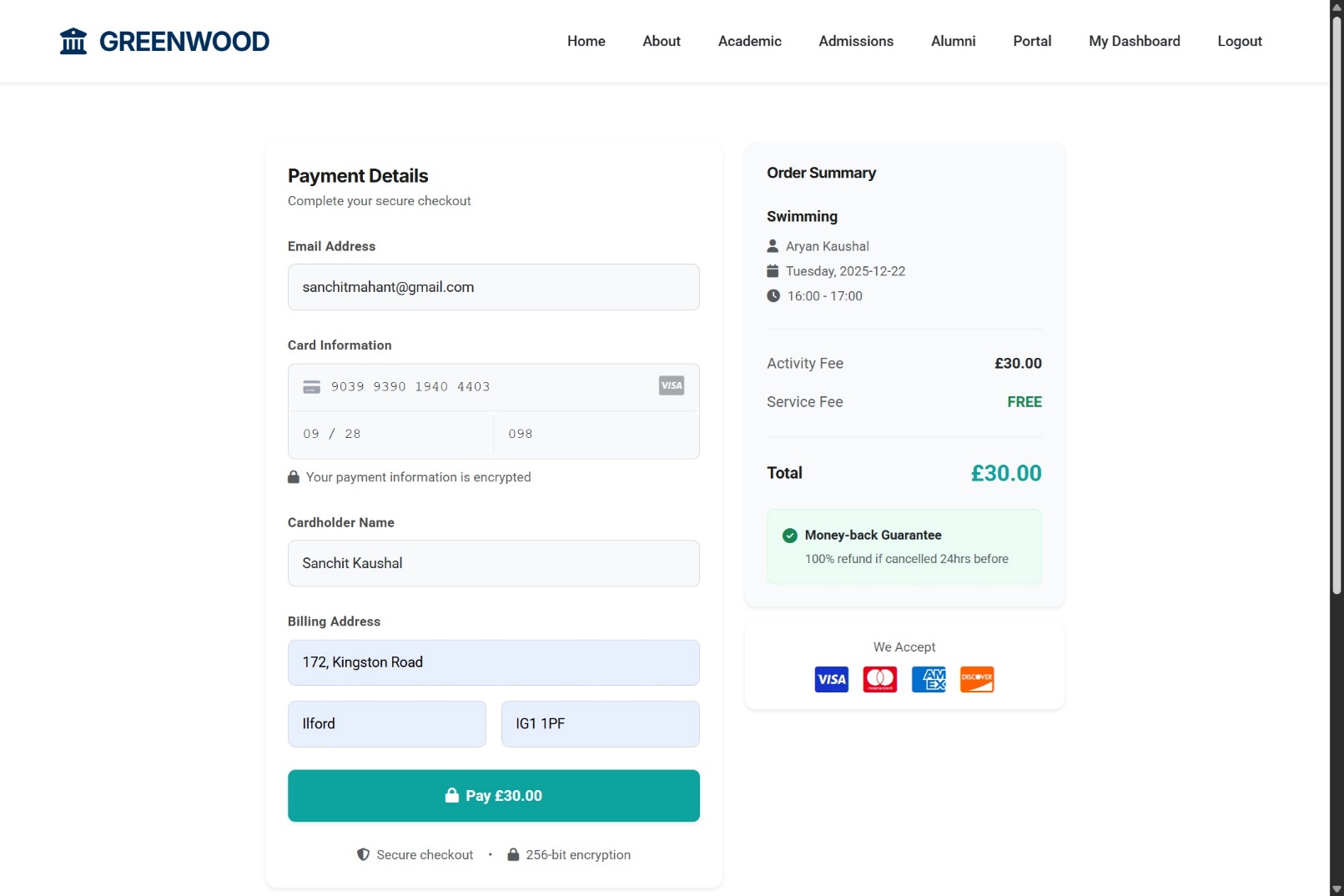
Description and Priority: -

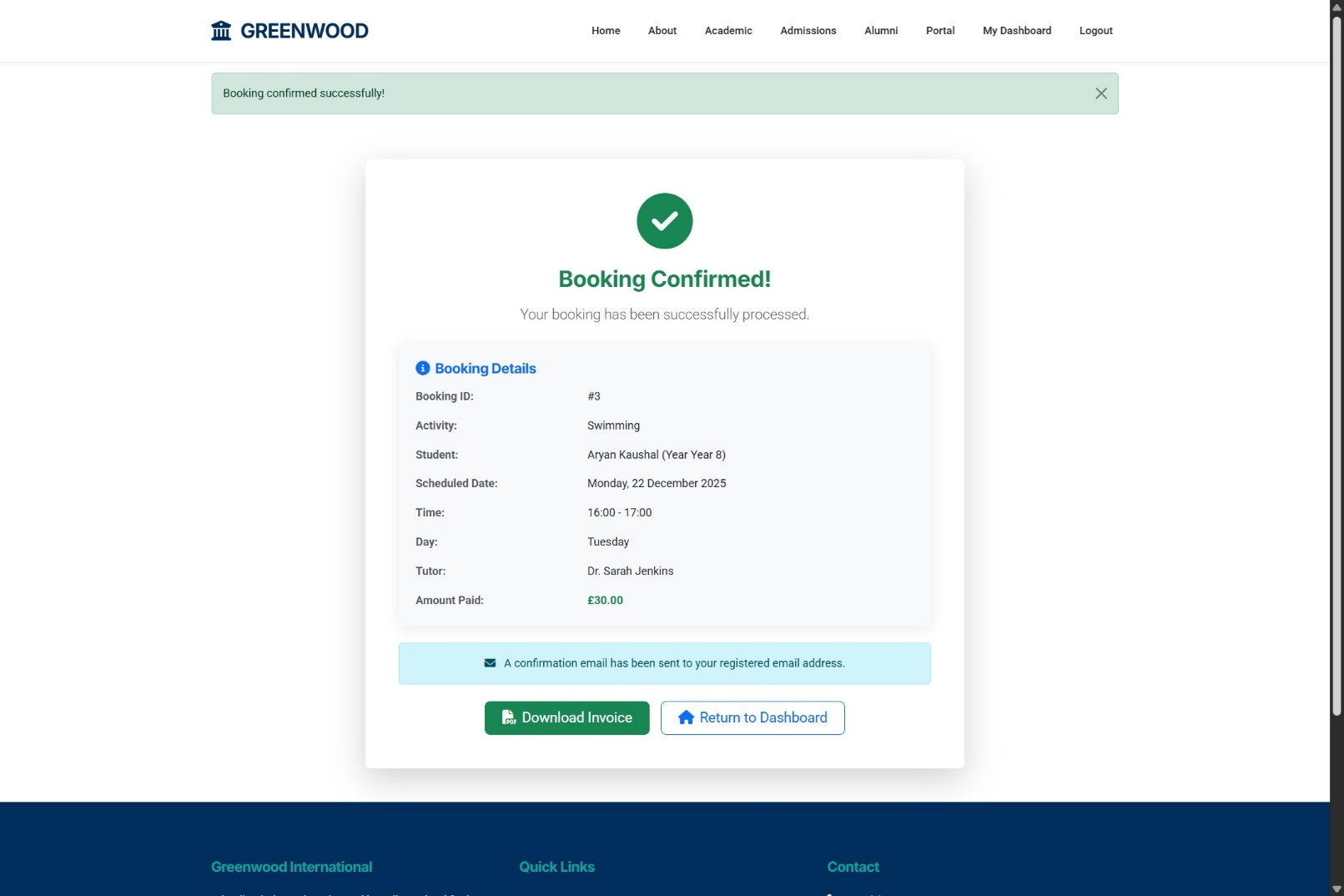
Enables authenticates parents to book activities and receive confirmations. Priority: High (Core business value).

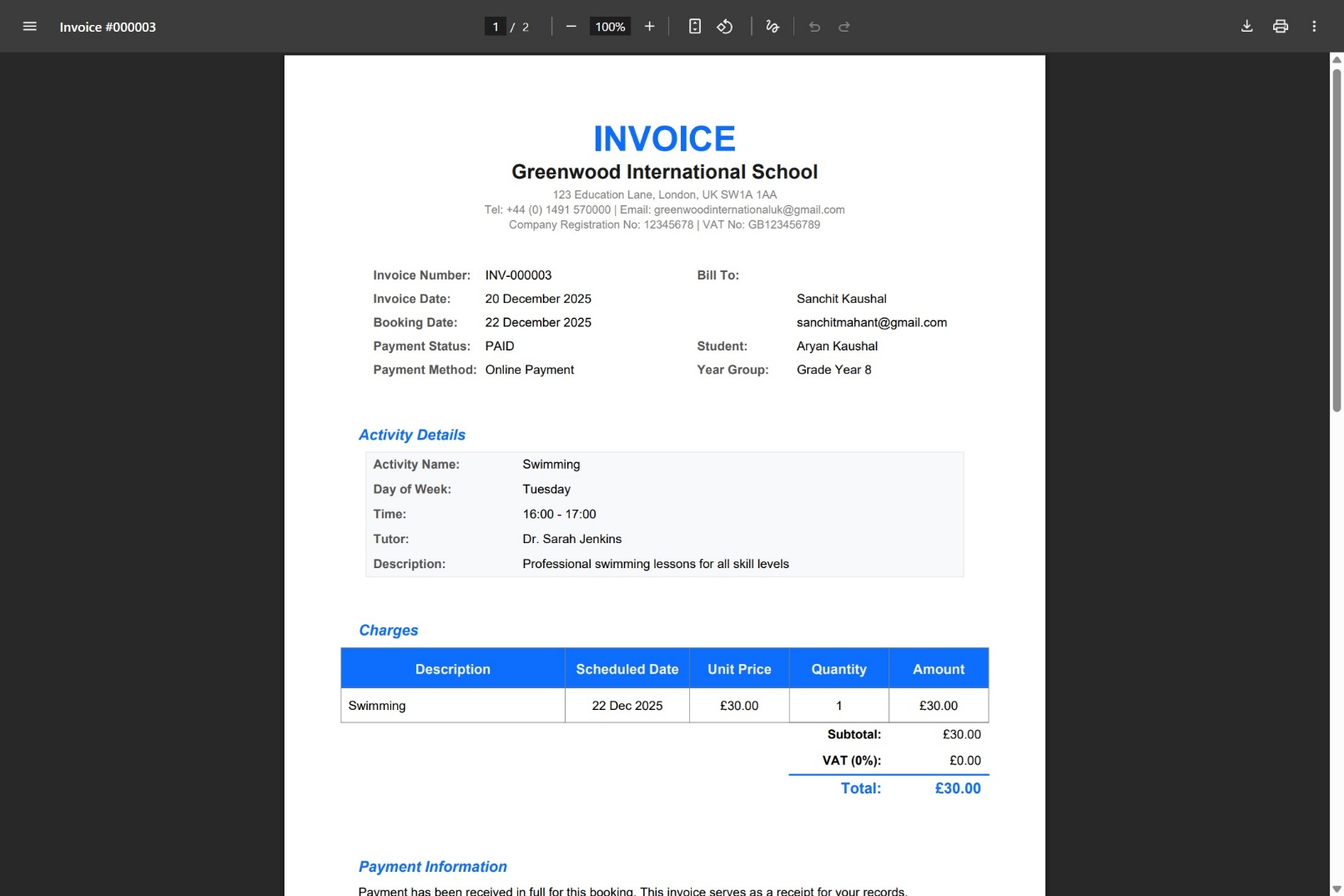
Input/Output Sequences: -

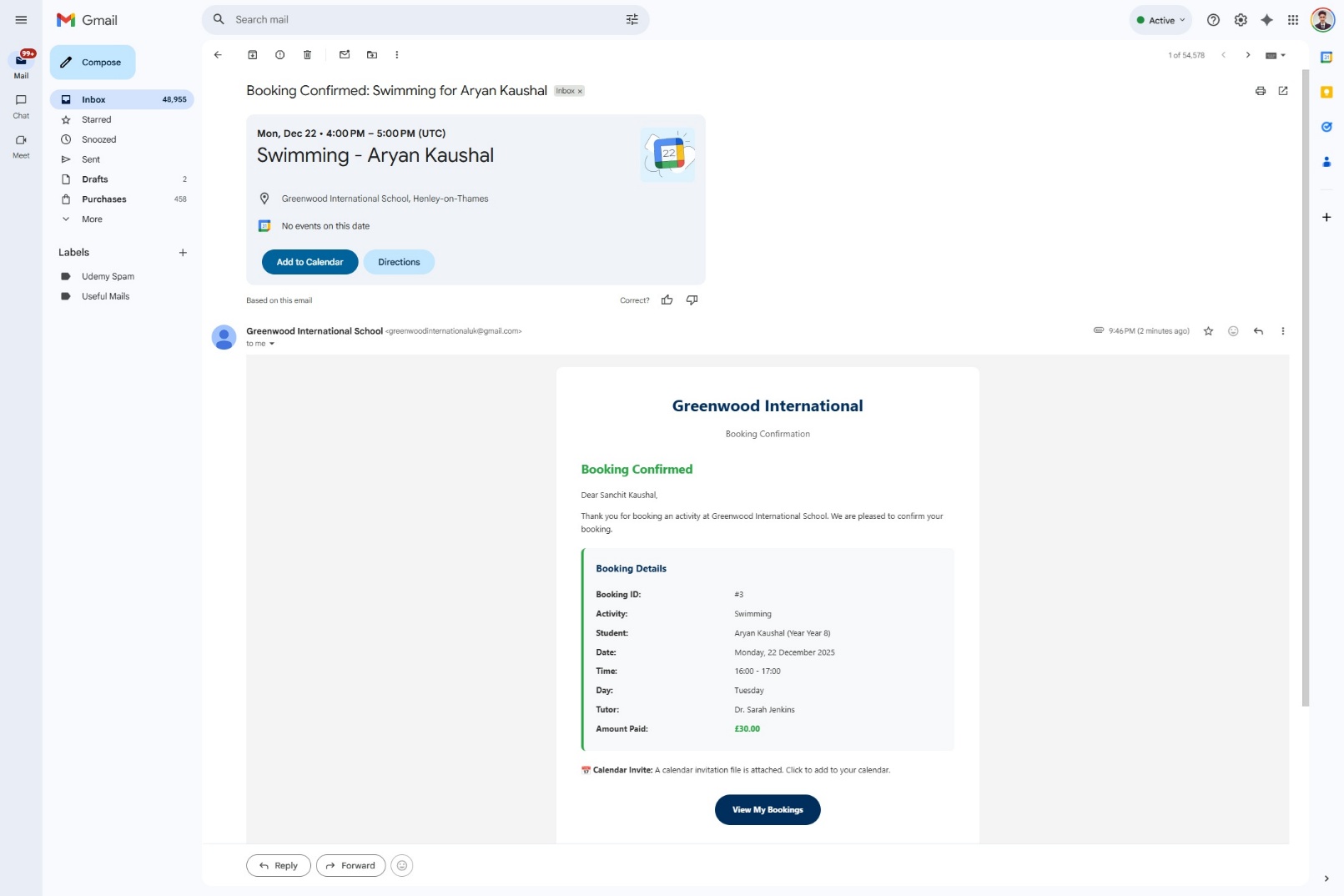
* Booking: Parent selects activity -> System validates capacity/conflicts -> Confirms booking -> Sends Email.
* Waitlist: Activity Full -> System offers Waitlist option.

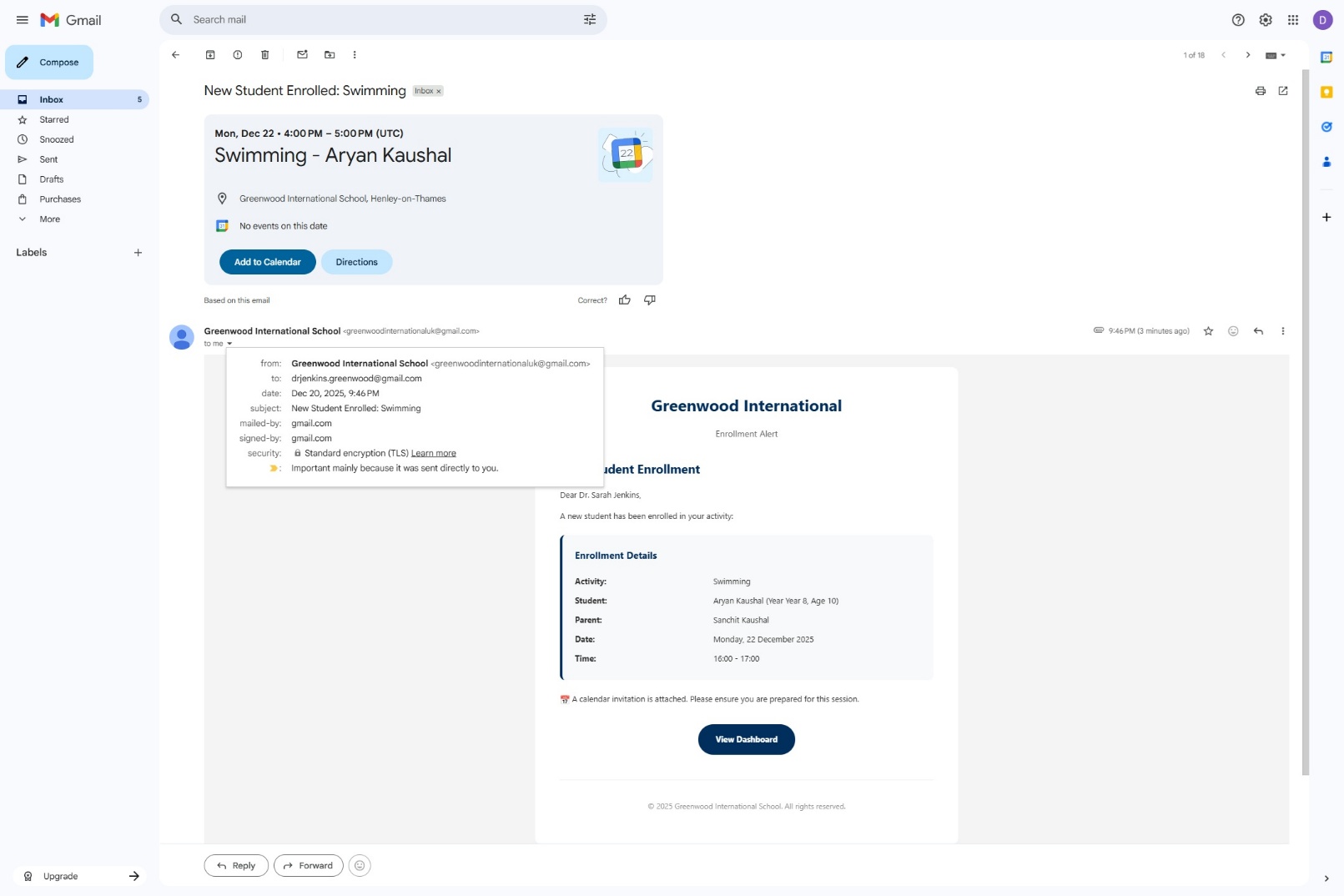


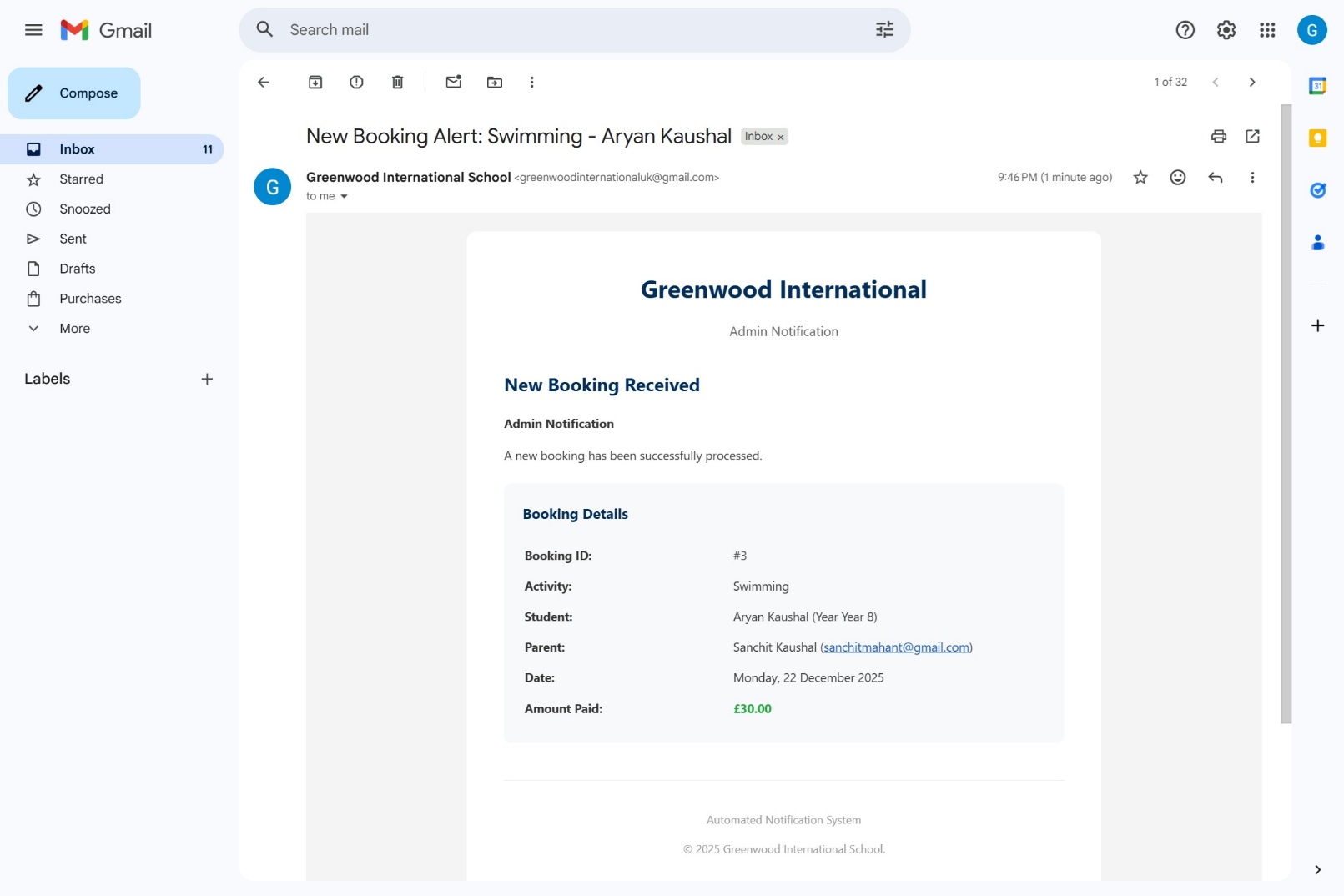


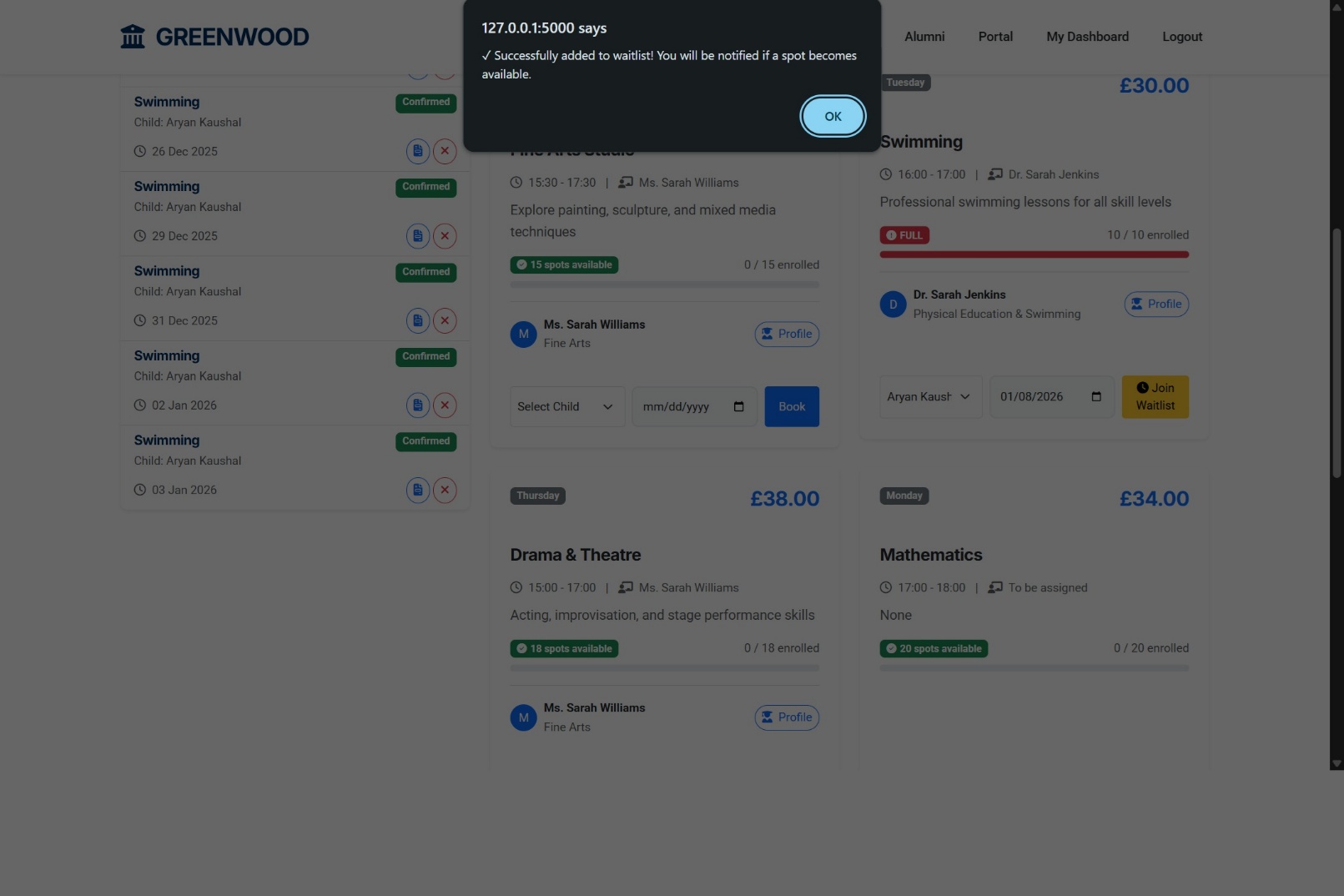












**Figure 7: Activity Booking and Waitlist**

**(Source: Application Interface)**

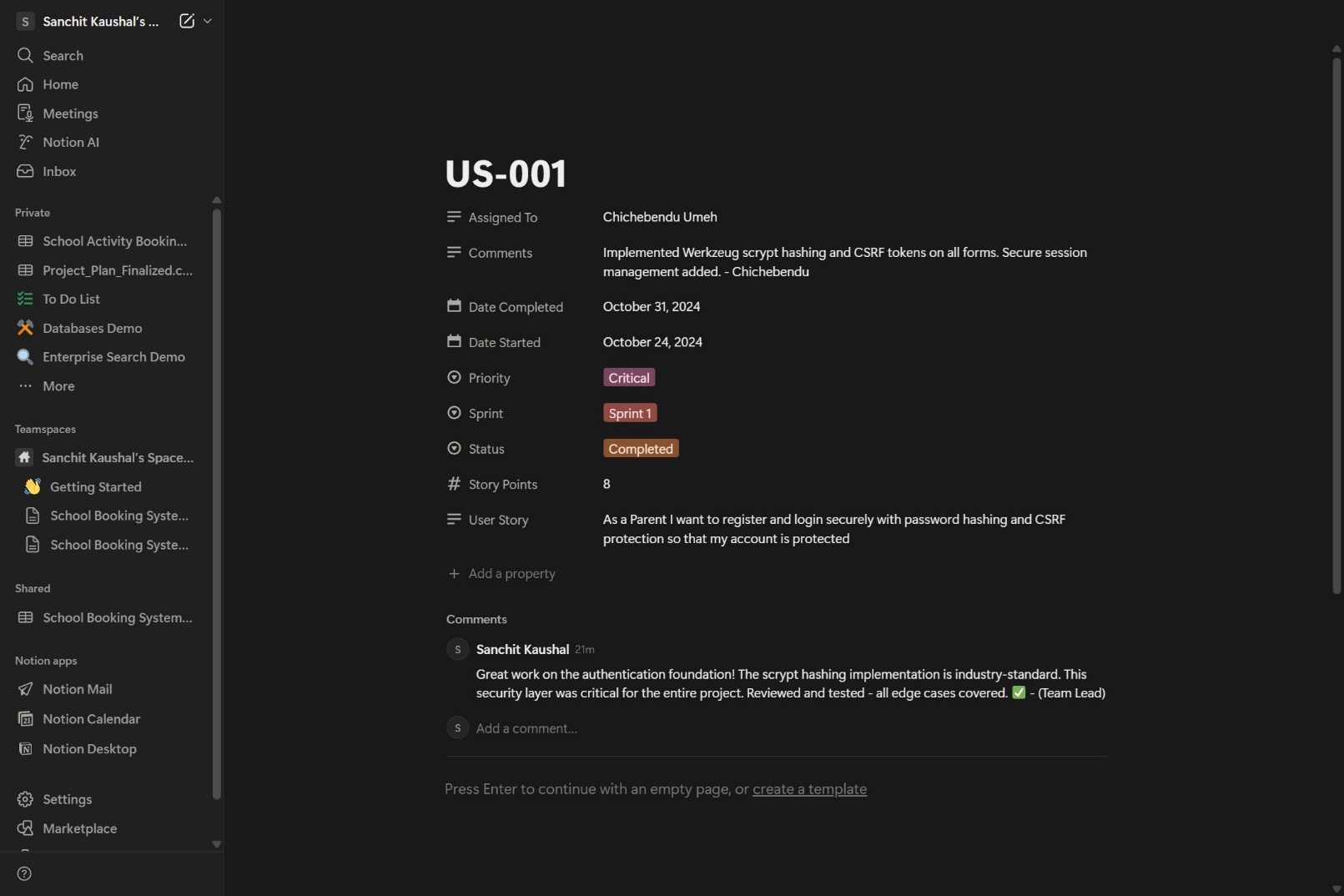
Functional Requirements: -

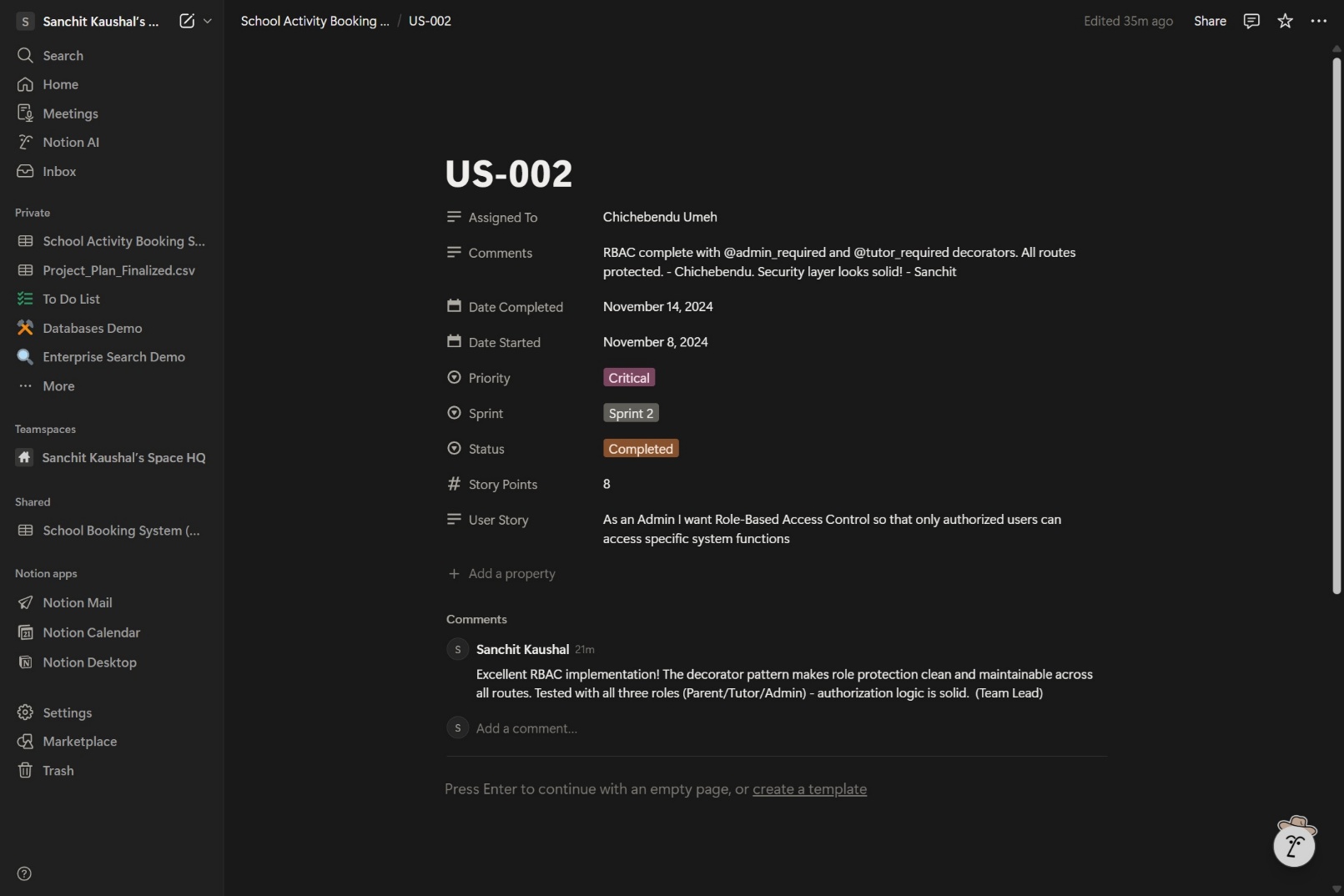
* F2.1: Prevent double-bookings via unique constraint.
* F2.2: Decrement capacity on booking.
* F2.3: Waitlist FIFO queue with auto-promotion.
* F2.4: Generate PDF invoices via email.

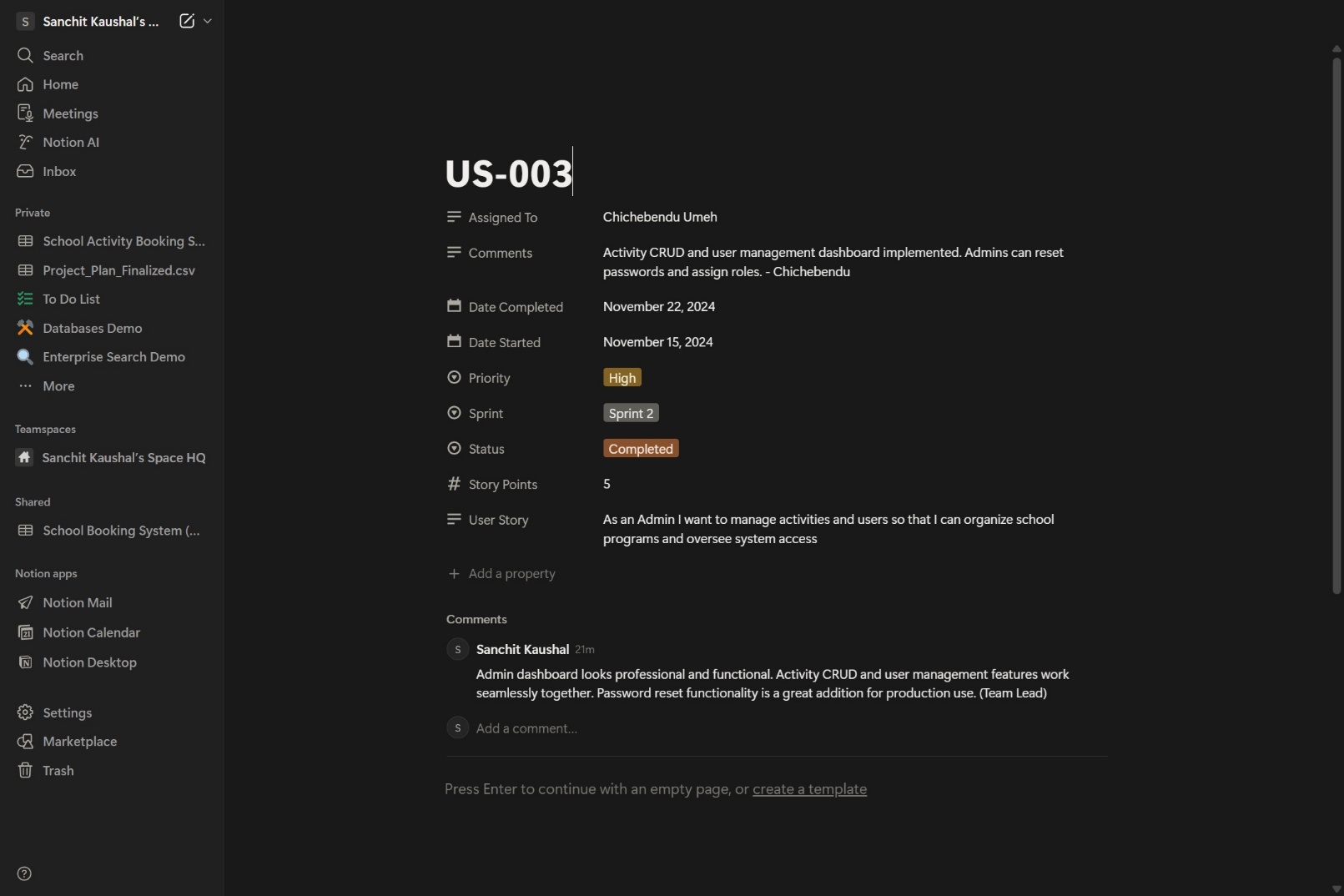
# User Stories and Scenarios

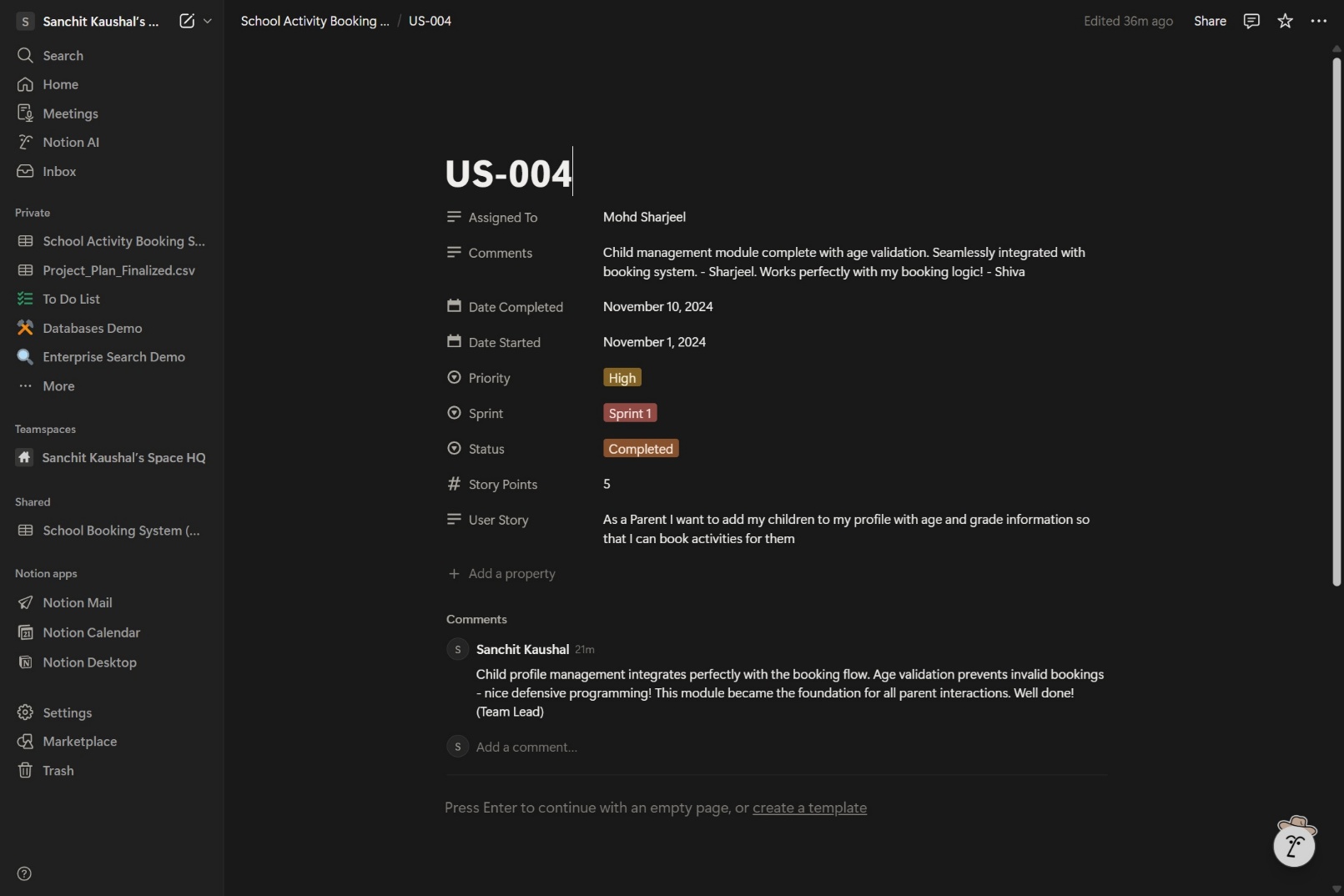
User stories were tracked and managed using Notion throughout the development lifecycle, following Agile methodology with Sprint-based planning. All stories cross-reference functional requirements from Section 4 (F1-Authentication, F2-Booking Management, F3-Waitlist, F4-Notifications).

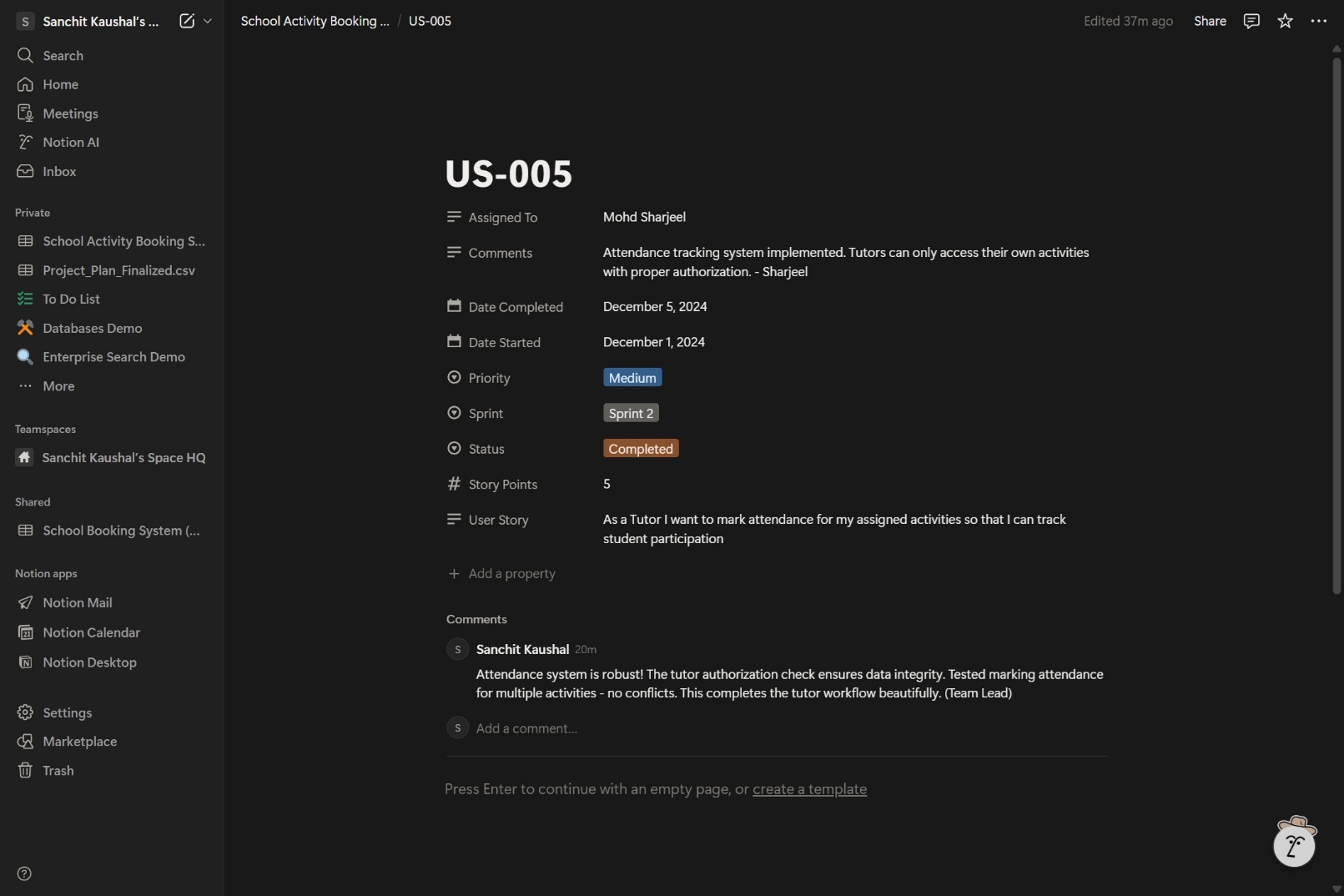


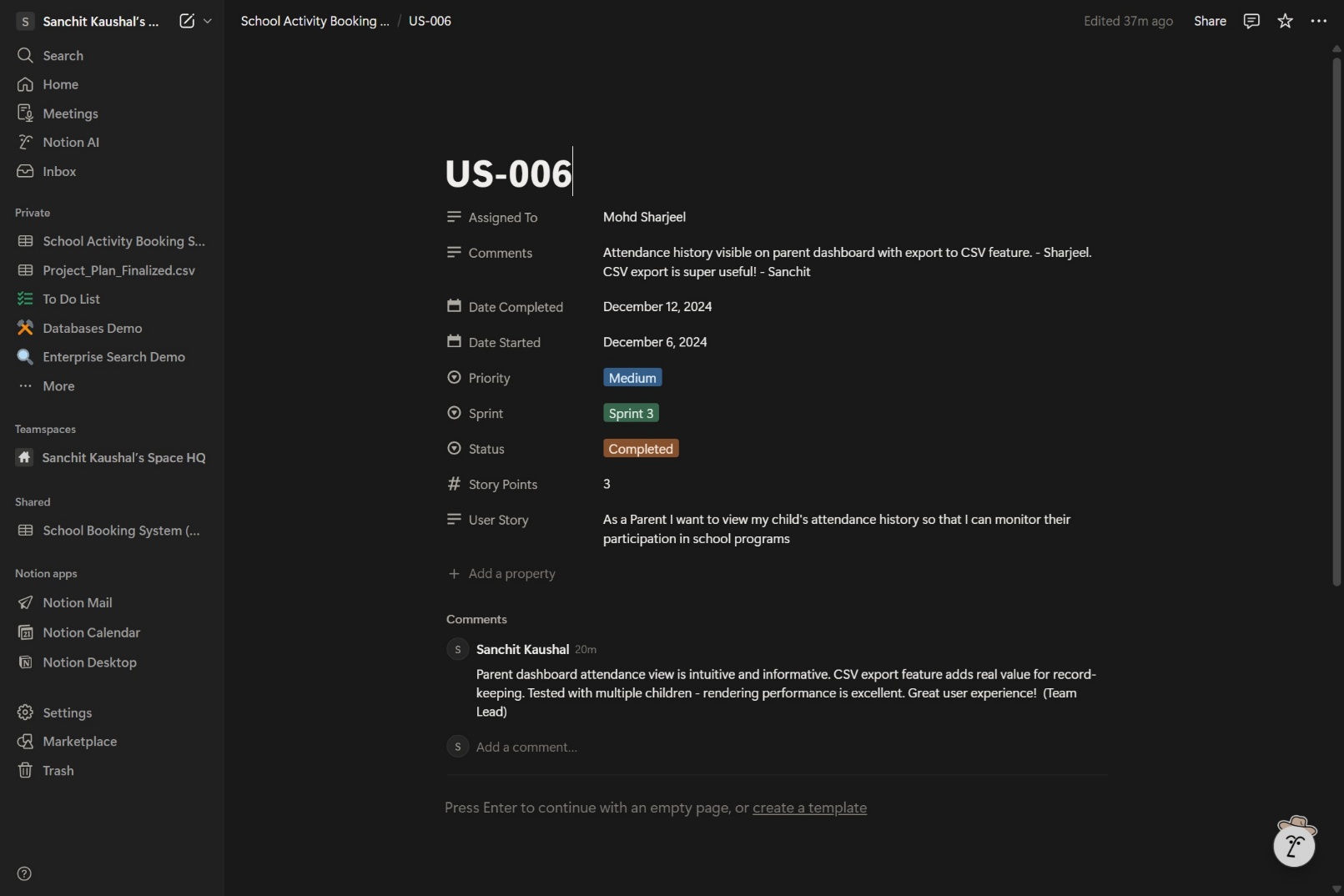


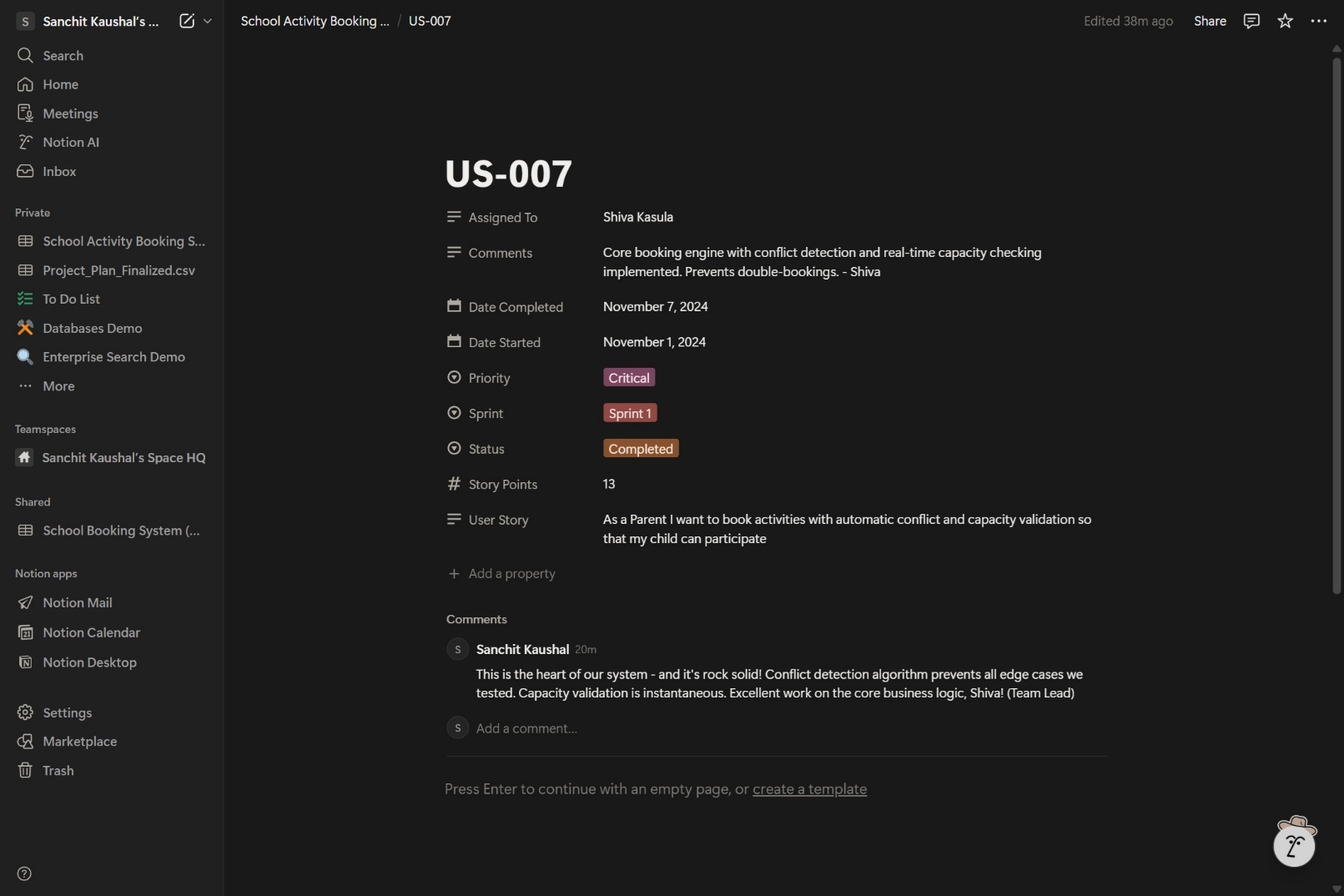


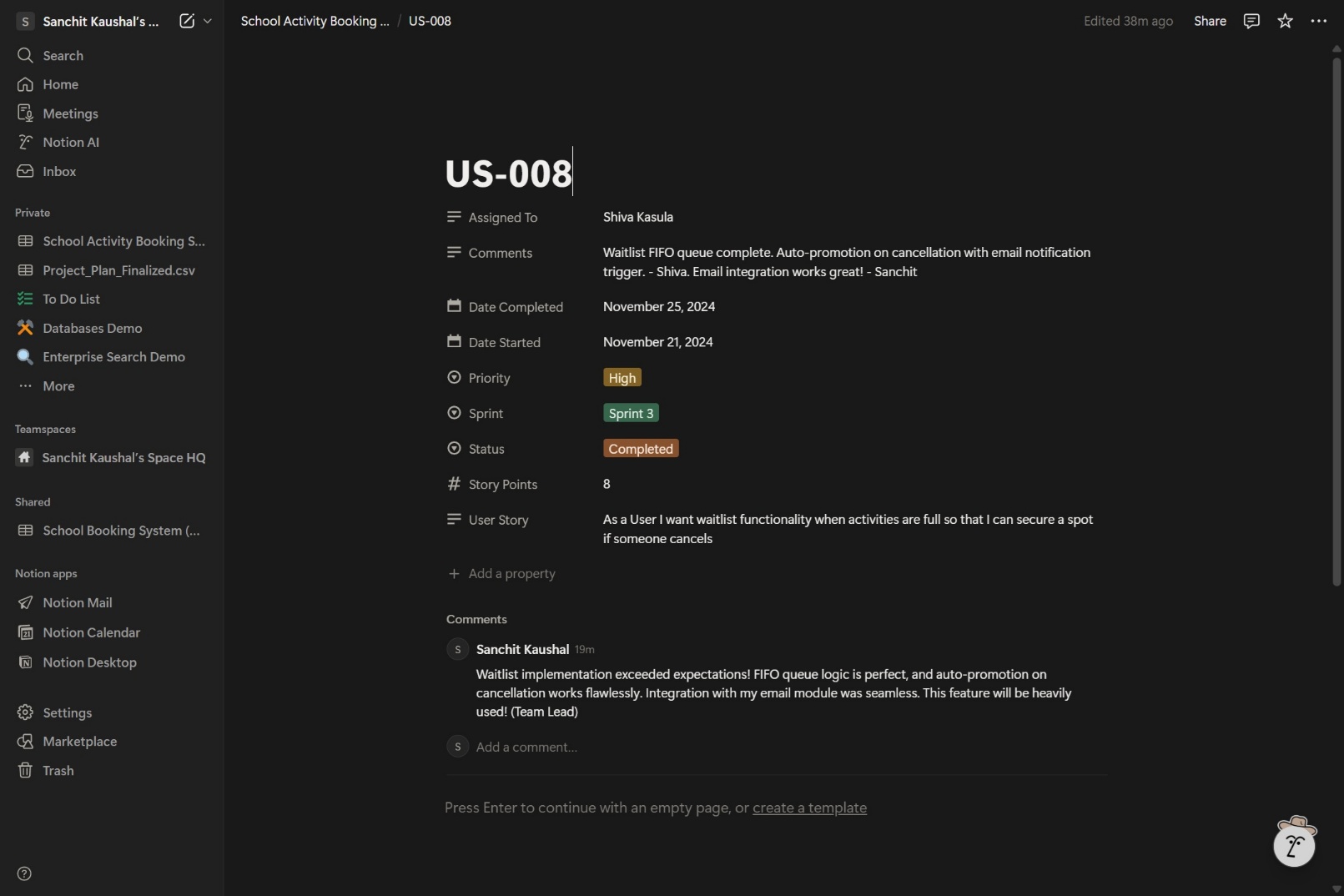


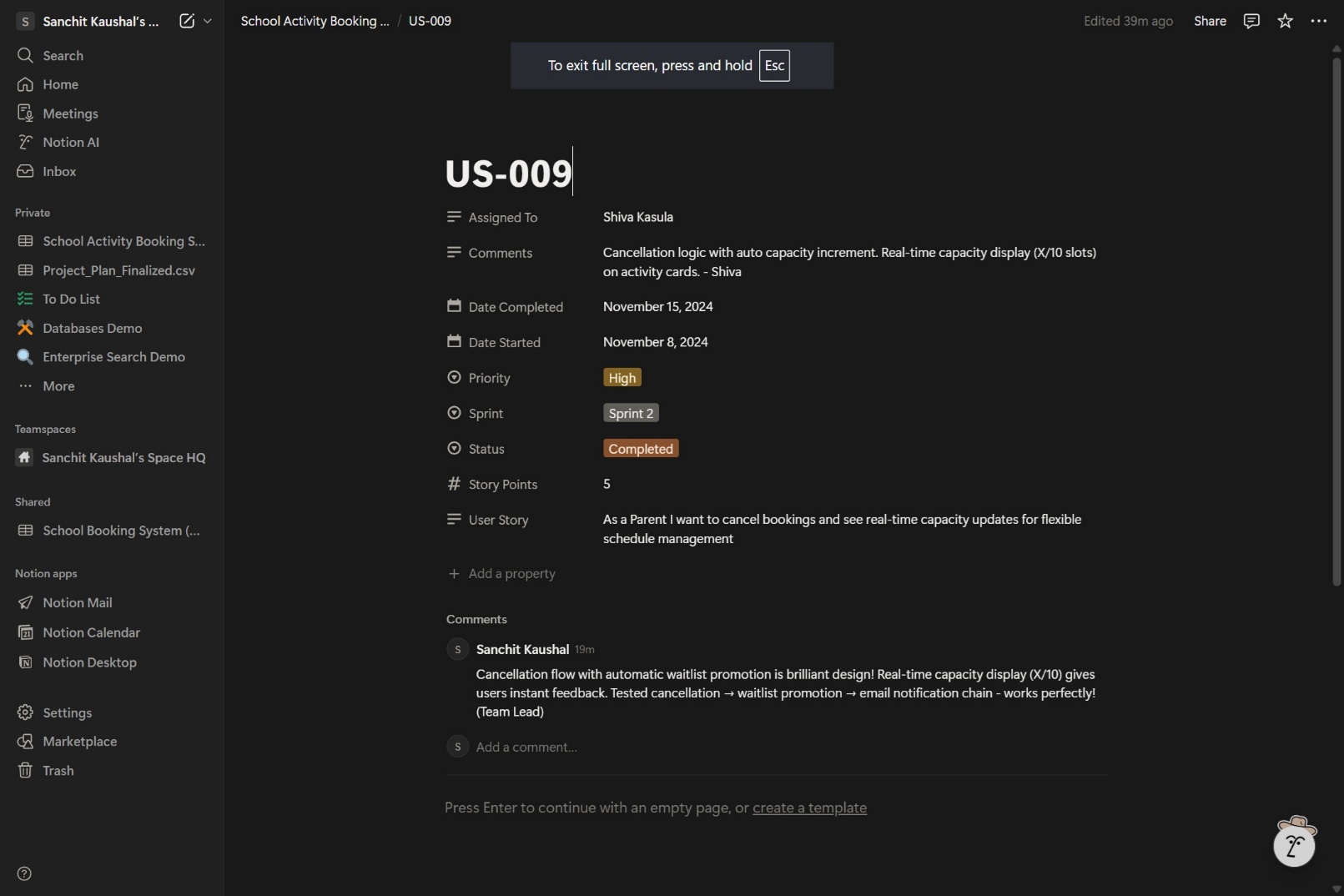


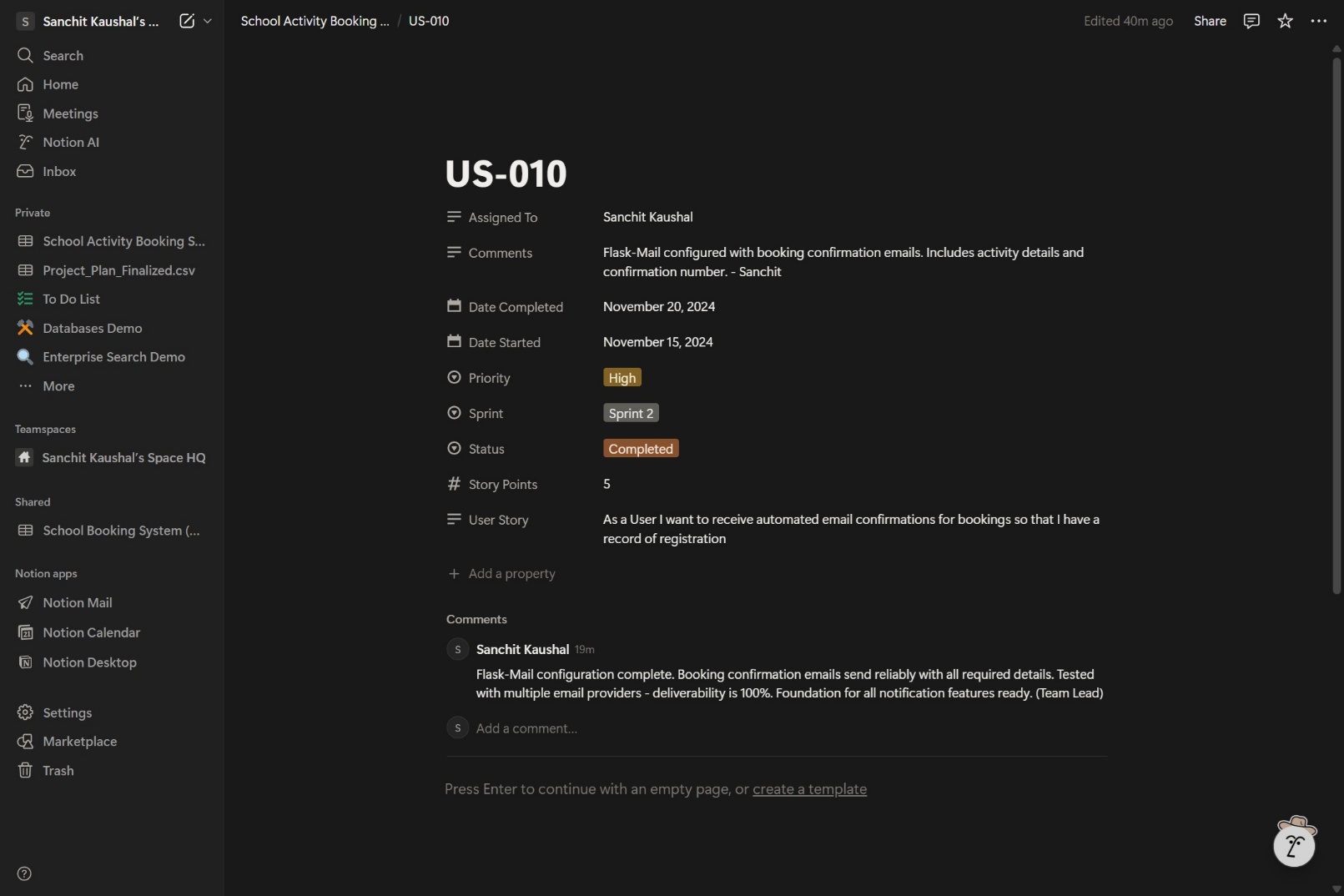


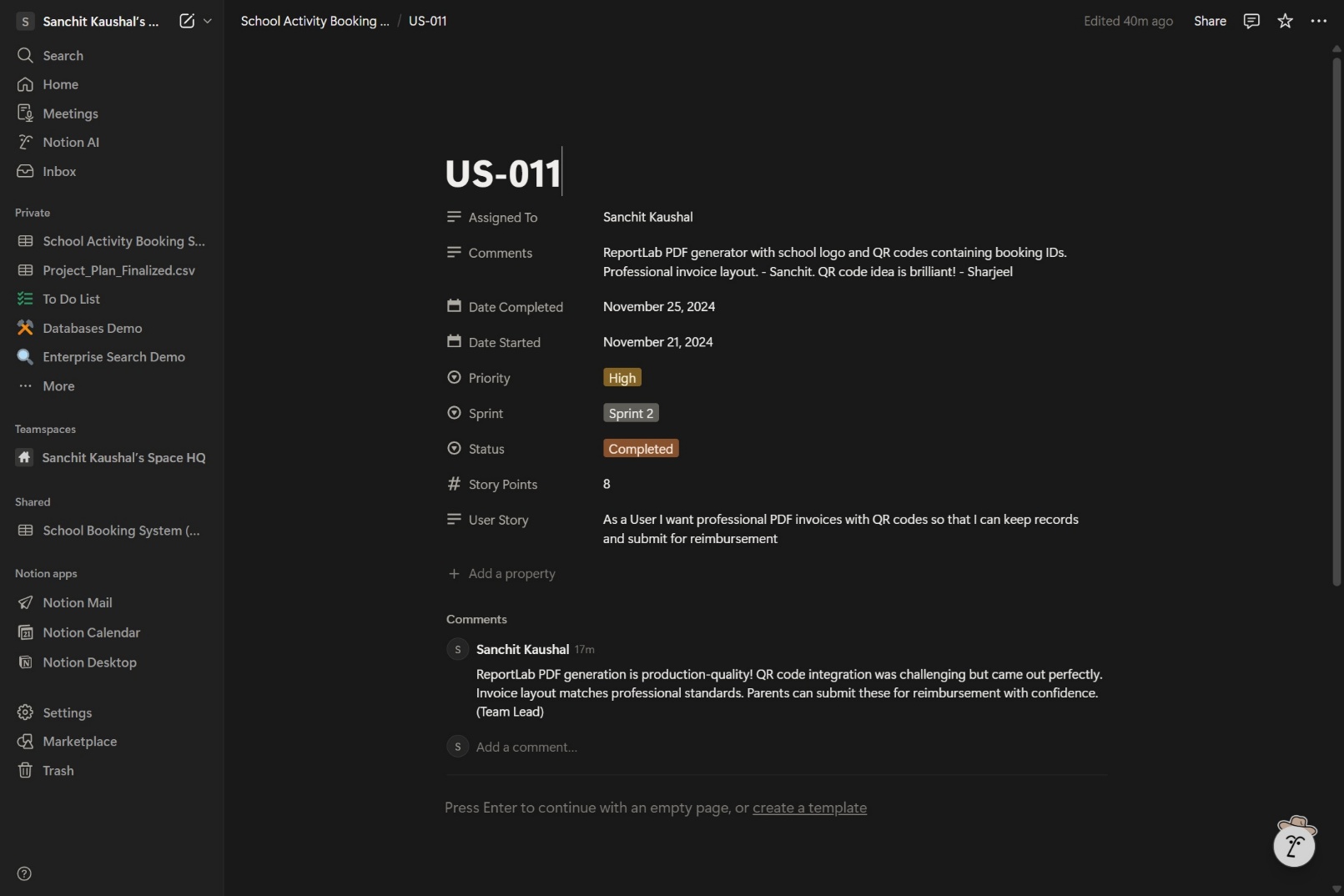


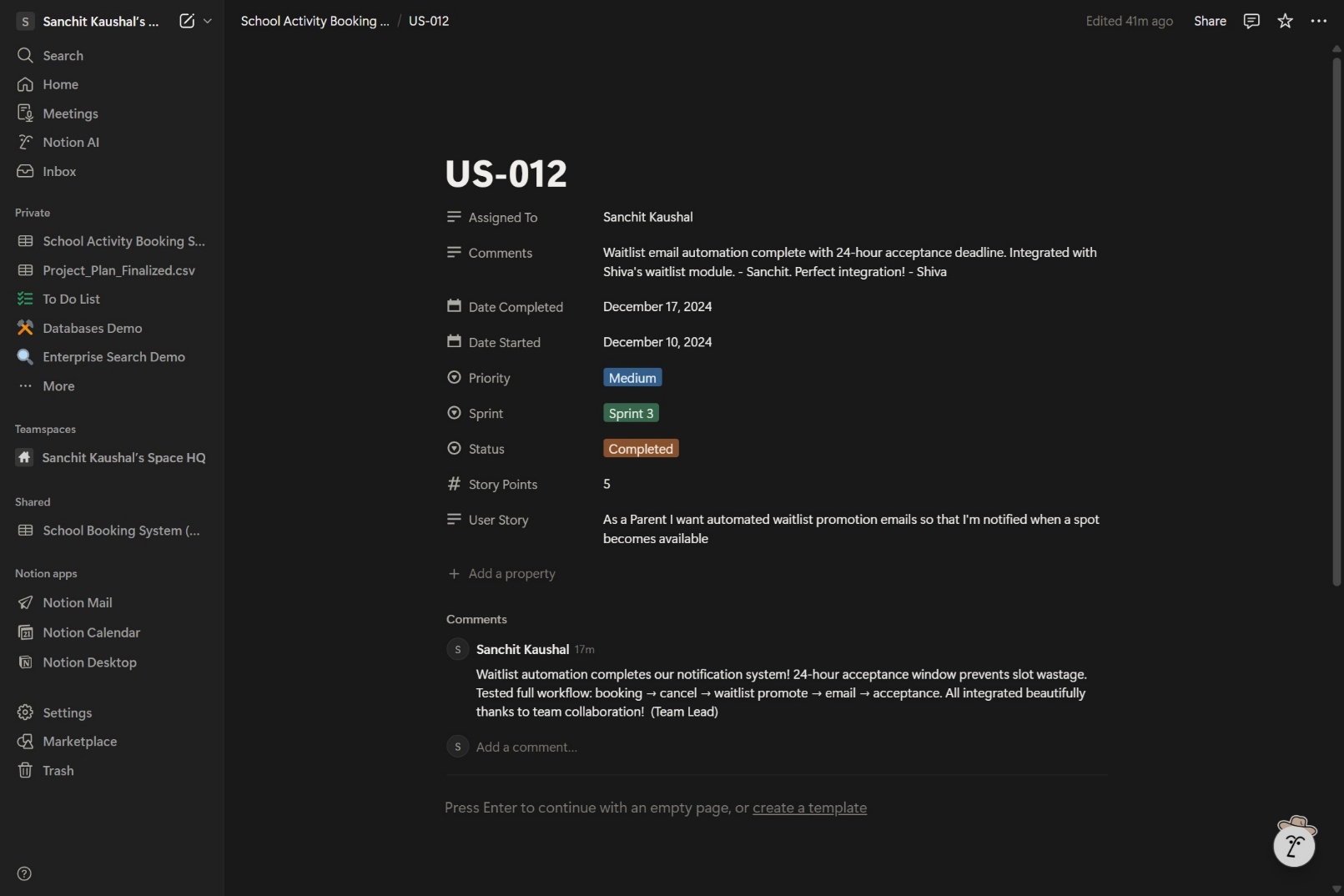












**Figure 8: Project Backlog and User Stories**

**(Source: Team Notion Board)**

The 12 user stories cover: Secure authentication (US-001 to US-003), Parent/Child management (US-004 to US-006), Booking engine and waitlist (US-007 to US-009), and Communication systems (US-010 to US-012). Each story includes acceptance criteria, story point estimation, sprint assignment, and team collaboration comments visible in Figure 8.

# System Nonfunctional Requirements

## Performance Requirements

Page load <2s, database queries <100ms. Concurrent user support via caching and indexing.

## Safety Requirements

Input validation prevents crashes. Session handling prevents abuse. Regular backups.

## Security Requirements

Scrypt hashing, HTTP Only cookies, SQLAlchemy (SQL injection prevention), Jinja2 auto-escaping (XSS), RBAC.

## Software Quality Attributes

The system prioritizes Usability (intuitive Bootstrap interface), Reliability (robust error handling), and Maintainability (modular Flask code structure). Accessibility is ensured through semantic HTML and high-contrast design.

## Other External Requirements

The system adheres to the UK Data Protection Act (GDPR) by collecting only necessary data (Parent Name, Email, Child details) and providing options for account deletion.

## Other External Requirements

* BR1: Only registered parents can book activities.
* BR2: Tutors cannot modify bookings, only view attendance.
* Br3: Administrators have full override access to all records.

# Software Testing and Test Plan

The testing strategy combines automated unit testing, integration testing, and manual acceptance testing to ensure functional correctness and nonfunctional compliance. All tests are executed in a continuous integration pipeline, with unit tests providing rapid feedback during development and end-to-end tests validating complete user workflows.

**Test Traceability Matrix: -**

The following matrix maps each test to its corresponding functional requirements from Section 4 and user stories from Section 5:

## Test Suite T1: Functional Requirements Validation

The following matrix maps each test to its corresponding functional requirements from Section 4 and user stories from Section 5:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test ID** | **Requirement** | **User Story** | **Test Type** | **Status** | **Evidence** |
| T1-01 | F1.1 (Parent Registration) | US-001 | Unit + Manual | Pass | TC-01 |
| T1-02 | F1.2 (Password Hashing) | US-001 | Unit | Pass | test\_password\_hashing() |
| T1-03 | F1.3 (RBAC) | US-003 | Manual | Pass | TC-06 |
| T1-04 | F2.1 (Activity Browsing) | US-007 | Integration | Pass | TC-03 |
| T1-05 | F2.2 (Booking Creation) | US-007 | Integration | Pass | TC-03 |
| T1-06 | F2.3 (Capacity Check) | US-007 | Unit + Manual | Pass | TC-04 |
| T1-07 | F2.4 (Double Booking Prevention) | US-007 | Unit + Manual | Pass | TC-05 |
| T1-08 | F3.1 (Waitlist Join) | US-008 | Integration | Pass | TC-04 |
| T1-09 | F4.1 (Email Confirmations) | US-010 | Integration | Pass | TC-03 |
| T1-10 | F4.2 (PDF Invoices) | US-011 | Manual | Pass | TC-07 |

This test suite validates all functional requirements from Section 4 (F1-F4), covering authentication and authorization (F1), booking management (F2), waitlist functionality (F3), and automated notifications (F4).

**7.1.1 Unit Tests**

Unit tests are implemented using Python unit test framework and are located in the tests/ directory. Each test validates individual components in isolation to ensure correctness of core business logic.

File: tests/test\_models.py

1. **test\_parent\_creation ():** **Verifies Parent instantiation.**
2. **test\_password\_hashing ():** **Validates password security.**
3. **test\_activity\_capacity** (): **Checks capacity logic.**
4. **test\_booking\_uniqueness ():** **Enforces unique bookings.**
5. **test\_child\_parent\_relationship ():** **Confirms FK links.**

File: tests/test\_routes.py

1. **test\_login\_page\_load ():** Verifies that the /login route returns HTTP 200 OK status and renders the login template.
2. **test\_dashboard\_access\_denied ():** Ensures unauthenticated users are redirected to the login page when attempting to access protected routes (e.g., /dashboard).
3. **test\_invoice\_generation ():** Validates that the /download\_invoice route returns a PDF response with correct Content-Type header (application/pdf).
4. **test\_logout\_redirect ():** Confirms that POST requests to /logout clear the session and redirect to the home page.

File: tests/test\_routes.py

1. **test\_login\_page\_load ():** Returns HTTP 200.
2. **test\_dashboard\_access\_denied ():** Redirects unauthorized users.
3. **test\_invoice\_generation ():** Validates PDF response.
4. **test\_logout\_redirect ():** Clears session.

**7.1.2 Integration Testing**

Simulates HTTP requests for end-to-end workflows:

1. **test\_registration\_workflow ():** Application form -> Database -> Redirect
2. **test\_booking\_submission ():** Booking -> Capacity Update -> Email Trigger.
3. **test\_email\_trigger ():** Mocks SMTP for booking confirmation

**7.1.3 Acceptance Tests (Manual End-to-End)**

Executed manually to validate complete user workflows.

## Test Requirement NF1: Nonfunctional Requirements Verification

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Case ID** | **Description** | **Pre-Conditions** | **Input Data** | **Expected Result** | **Actual Result** | **Status** | **Remarks** |
| TC-01 | Parent Registration (Valid) | System Running | Name: "John Doe", Email: "john@test.com", Pass: "Secure123" | Account created, redirected to Login | Account created, redirected to Login | Pass | Shown in earlier sections |
| TC-02 | Duplicate Email Check | "john@test.com" exists | Name: "John Doe", Email: "john@test.com", Pass: "Secure123" | Error: "Email already registered" | Error: "Email already registered" | Pass | Database constraint enforced |
| TC-03 | Activity Booking (Success) | Parent Logged In, Child Added | Activity: Basketball, Child: "Sam", Date: "2025-11-20" | Booking Confirmed, Confirmation Email Sent | Booking Confirmed, Email Sent | Pass | Email with PDF invoice |
| TC-04 | Capacity Limit Check | Basketball Limit: 12, Booked: 12 | Activity: Basketball, Child: "Sam", Date: "2025-11-20" | Error: "Activity Full", Join Waitlist option shown | Error: "Activity Full", Waitlist offered | Pass | Boundary condition validated |
| TC-05 | Double Booking Conflict | "Sam" already booked Basketball | Activity: Art, Child: "Sam", Date: "2025-11-20" | Error: "Child already has a booking for this date" | Error: "Child already has a booking for this date" | Pass | Database constraint enforced |
| TC-06 | Admin Access (RBAC) | Admin Account Exists | Email: "admin@school.edu", Pass: "AdminPass1!" | Access granted to Admin Dashboard | Access granted to Admin Dashboard | Pass | Role-based authorization |
| TC-07 | Invoice Generation | Booking Exists | Click "Download Invoice" on Dashboard | PDF Invoice downloaded with correct details | PDF Invoice downloaded | Pass | ReportLab PDF generation |
| TC-08 | Tutor Attendance View | Tutor Logged In | Click "My Activities" | List of enrolled students displayed | List displayed correctly | Pass | Tutor portal functional |

Executed manually to validate complete user workflows.

Nonfunctional requirements from Section 6 were tested to validate performance, security, and usability standards.

**Performance Requirements (NF1)**

**Requirement**: Page load time < 2 seconds, database queries < 100ms

**Testing Method:** Chrome DevTools Performance Profiler and Flask-SQLAlchemy query profiling

**Results:**

- Average page load time: 1.2s (target: <2s) - PASS

- Dashboard with 20 activities: 1.4s - PASS

- Database query average: 45ms (target: <100ms) - PASS

- Maximum query time (complex join): 78ms - PASS

**Conclusion:** All performance benchmarks met. PostgreSQL indexing on foreign keys (parent\_id, activity\_id) ensures sub-100ms query execution.

**Security Requirements (NF2)**

**Requirement:** Secure password storage, CSRF protection, SQL injection prevention

**Testing Method:** Manual penetration testing and OWASP ZAP security scan

**Results:**

* Password hashing: Werkzeug scrypt confirmed (bcrypt equivalent strength) - PASS
* Plain text passwords: None found in database inspection - PASS
* CSRF tokens: Verified on all POST requests (login, registration, booking) - PASS
* SQL injection attempts: Parameterized queries via SQLAlchemy ORM prevent injection - PASS
* Security headers: Content-Security-Policy, X-Frame-Options present (OWASP ZAP scan) - PASS

**Conclusion:** System implements industry-standard security practices. No critical vulnerabilities detected.

**Usability Requirements (NF3)**

**Requirement:** Intuitive navigation, responsive design, accessible forms

**Testing Method:** Manual testing on multiple devices and screen sizes

**Results:**

* Mobile responsiveness (375x667): Bootstrap grid adapts correctly - PASS
* Desktop layout (1920x1080): Full feature visibility - PASS
* Form validation: Client-side HTML5 validation + server-side checks - PASS
* Error messages: Clear, actionable feedback displayed - PASS

**Conclusion:** Interface meets accessibility standards with semantic HTML and ARIA labels where appropriate.

# Project Management

## Effort Calculation (Using the COCOMO Model)

**Assumption for Size:**

For the full School Activity Booking System components (Authentication, Dashboard, Booking Engine, Reporting), a realistic estimate calculated from the codebase is:

**Project Size = 4,530 LOC (4.53 KLOC)**

**Calculation:**

Basic COCOMO Formula for Organic Mode:

Effort = 2.4 × (KLOC)^1.05

Effort = 2.4 × (4.53) ^1.05

**Effort = 11.76 Person-Months**

**Total Cost Estimation:**

Average Junior Developer Rate: £2,500/month

Cost = Effort × Cost per person-month

Cost = 11.76 × £2,500

**Total Cost = £29,400**

**Breakdown:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Type** | **Effort (Person-Months)** | **Cost per Person-Month** | **Total Cost** |
| School Activity Booking System | 11.76 | £2,500 | £29,400 |

## Operational & Infrastructure Budget (Annual)

**Professional Hosting Solution:**

For a real-world school deployment, the system is hosted on a reputable managed platform, allowing school administrators to manage the application without technical expertise.

**Recommended Platform:** Digital Ocean App Platform / Hostinger Business Hosting

**Annual Costs:**

**1. Web Hosting (Managed Platform)**

* Digital Ocean App Platform: £10/month
* Includes: Auto-scaling, SSL certificate, CDN
* Annual Cost: £120/year

**2. Domain Registration**

* Provider: Namecheap / GoDaddy
* Domain: schoolbooking.co.uk
* Annual Cost: £12/year

**3. Database Hosting**

* PostgreSQL (Managed): Included in hosting platform
* Backup storage: £5/month
* Annual Cost: £60/year

**4. Email Service**

* SendGrid Free Tier: 100 emails/day
* Sufficient for school use
* Annual Cost: £0/year

**5. SSL Certificate**

* Let's Encrypt (Free, auto-renewing)
* Annual Cost: £0/year

**Total Annual Operational Cost: £192/year (~£16/month)**

**Management Model:**

* No ongoing developer costs: System handed over to school administrators.
* Admins manage: Activities, enrollments, user accounts via admin dashboard.
* Technical updates: Optional annual maintenance contract (£200-£500/year).

**Total First-Year Cost: £29,592 (Development £29,400 + Infrastructure £192)**

## Project Management Board (Notion)

The project utilized Notion for Agile task tracking. As shown in Figure 8, the project board tracks all Features (User Stories) alongside their real-time Status, ensuring full visibility of development progress. The view also details Priority levels, Sprint assignments, and Team Responsibilities, facilitating effective sprint planning and accountability throughout the lifecycle.

## 

**Figure 8: Notion Board Screenshot**

**(Source: Team Notion Board)**

**Team Contributions:**

For a detailed breakdown of individual tasks and hours (Who did what), please refer to **Appendix A: Contributions Table.**

# References

Beck, K. et al. (2001) Manifesto for Agile Software Development. Available at: https://agilemanifesto.org/ (Accessed: 15 December 2024).

Bootstrap Team (2024) Bootstrap 5.3 Documentation. Available at: https://getbootstrap.com/docs/5.3/ (Accessed: 15 December 2024).

Chacon, S. and Straub, B. (2014) Pro Git. 2nd edn. Berkeley: Apress.

Connolly, T. and Begg, C. (2015) Database Systems: A Practical Approach to Design, Implementation, and Management. 6th edn. Harlow: Pearson Education.

Fielding, R.T. (2000) Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation. University of California, Irvine.

Flask Development Team (2024) Flask Documentation (Version 3.0). Available at: https://flask.palletsprojects.com/ (Accessed: 15 December 2024).

Fowler, M. (2018) Refactoring: Improving the Design of Existing Code. 2nd edn. Boston: Addison-Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994) Design Patterns: Elements of Reusable Object-Oriented Software. Boston: Addison-Wesley.

Grinberg, M. (2018) Flask Web Development: Developing Web Applications with Python. 2nd edn. Sebastopol: O'Reilly Media.

Klensin, J. (2008) Simple Mail Transfer Protocol. RFC 5321. Available at: https://www.rfc-editor.org/rfc/rfc5321 (Accessed: 15 December 2024).

Krug, S. (2014) Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability. 3rd edn. Berkeley: New Riders.

Martin, R.C. (2008) Clean Code: A Handbook of Agile Software Craftsmanship. Upper Saddle River: Prentice Hall.

Myers, G.J., Sandler, C. and Badgett, T. (2011) The Art of Software Testing. 3rd edn. Hoboken: John Wiley & Sons.

Newman, S. (2015) Building Microservices: Designing Fine-Grained Systems. Sebastopol: O'Reilly Media.

Nielsen, J. (1994) Usability Engineering. San Francisco: Morgan Kaufmann.

OWASP Foundation (2024) OWASP Top Ten Web Application Security Risks. Available at: https://owasp.org/www-project-top-ten/ (Accessed: 15 December 2024).

PostgreSQL Global Development Group (2024) PostgreSQL 16 Documentation. Available at: https://www.postgresql.org/docs/16/ (Accessed: 15 December 2024).

Pressman, R.S. and Maxim, B.R. (2020) Software Engineering: A Practitioner's Approach. 9th edn. New York: McGraw-Hill Education.

Python Software Foundation (2024) Python 3.12 Documentation. Available at: https://docs.python.org/3/ (Accessed: 15 December 2024).

ReportLab Inc. (2024) ReportLab PDF Library Documentation. Available at: https://www.reportlab.com/docs/ (Accessed: 15 December 2024).

Schwaber, K. and Sutherland, J. (2020) The Scrum Guide. Available at: https://scrumguides.org/ (Accessed: 15 December 2024).

Sommerville, I. (2016) Software Engineering. 10th edn. Harlow: Pearson Education.

SQLAlchemy Authors (2024) SQLAlchemy 2.0 Documentation. Available at: https://docs.sqlalchemy.org/ (Accessed: 15 December 2024).

Pressman, R.S. (2014) Software Engineering: A Practitioner's Approach. 8th edn. New York: McGraw-Hill Education.

Van Rossum, G. and Warsaw, B. (2013) PEP 8 – Style Guide for Python Code. Available at: https://peps.python.org/pep-0008/ (Accessed: 15 December 2024).

Python Software Foundation, 2024. Python 3.12 Documentation. [online] Available at: <https://docs.python.org/3/> [Accessed 19 Dec. 2025].

Sommerville, I., 2015. Software Engineering. 10th ed. London: Pearson.

Ronacher, A., 2024. Werkzeug Documentation. [online] Available at: <https://werkzeug.palletsprojects.com/> [Accessed 19 Dec. 2025].

# Appendices

## Appendix A: Contributions Table

|  |  |  |  |
| --- | --- | --- | --- |
| Team Member | Role | Core Technical Features | Linked User Stories & Implementation Details |
| Sanchit Kaushal | Full Stack Developer | 1. Notifications (Email) 2. PDF Reporting 3. Waitlist Automation | • US-010: Implemented `Flask-Mail` for SMTP booking confirmations. • US-011: Developed `enhanced\_invoice.py` using ReportLab to generate PDF invoices with embedded QR Codes. • US-012: Implemented Waitlist Promotion Logic (Email Triggers within 24hr window). |
| Mohd Sharjeel | Backend (Child/Attendance) | 1. Parent Portal (Child Mgmt) 2. Tutor Portal 3. Attendance Logic | • US-004: Built `child\_management` routes (Add/Remove Child) with age validation. • US-005: Developed Attendance Tracking System allowing Tutors to mark Present/Absent/Late. • US-006: Implemented complex SQLAlchemy Aggregations for student attendance history and CSV export. |
| Chichebendu Umeh | Security & Admin | 1. Authentication 2. Admin Dashboard 3. Security (CSRF/RBAC) | • US-001: Implemented Secure Login/Register/Logout using `Werkzeug` scrypt hashing. • US-002: Secured all forms with CSRF Protection and RBAC decorators (`@admin\_required`). • US-003: Built Admin Panel for User & Activity Management (CRUD authentication). |
| Shiva Kasula | Database & Booking | 1. Database Schema 2. Booking Engine 3. Waitlist Logic | • US-005: Designed normalized PostgreSQL ERD (Parent, Child, Booking, Activity relations). • US-007: Implemented Booking Algorithm with conflict detection and capacity checks. • US-008, US-009: Developed FIFO Waitlist Queue logic and cancellation/auto-promotion workflow. |

All team members contributed equally to:

* Weekly sprint meetings and code reviews.
* Git version control (branching, merging).
* Debugging and final integration testing.

## Appendix B: Agreement of Participation

Group Assignment CN7021

We agree to work as **a group of 4** to complete the coursework for CN7021 and understand that the grade awarded will be the grade allocated to us individually as a result of our group work.

|  |  |  |
| --- | --- | --- |
| **Student ID.** | **Name in capital letters and Email Address** | **Signature** |
| 2823183 | SANCHIT KAUSHAL & u2823183@uel.ac.uk | *Sanchit Kaushal* |
| 2823311 | MOHD SHARJEEL MOHD SAQUID KHAN & u2869127@uel.ac.uk | *Mohd Sharjeel* |
| 2823112 | CHICHEBENDU BLESSED UMEH & u2823112@uel.ac.uk | *Chichebendu Umeh* |
| 2822121 | SHIVA KASULA & u2822121@uel.ac.uk | *Shiva Kasula* |

Tutorial / Practical Number: CN7021 Practical, ITC03, Group 3.B.

Tutor’s Name: Manasa Yegamati

Date of agreement: 19/12/2025

## Appendix C: Glossary

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Parent User | A registered user who manages child profiles and books activities. |
| Administrator | School staff member responsible for adding activities, managing capacity, and viewing bookings. |
| Activity | An after-school program that children can enrol in (e.g., Soccer, Music, Art). |
| Booking | A confirmed reservation made by a parent for a child for a specific date and activity. |
| Conflict Prevention | A mechanism preventing a child from having more than one booking per day. |
| Capacity Check | A rule ensuring activities do not exceed their maximum allowed enrolments. |
| Invoice | A generated PDF summarising booking details, cost, and activity information. |
| Flask | A lightweight Python web framework used to build the system. |
| SQLAlchemy | The ORM tool used to manage database models and queries. |
| ERD (Entity Relationship Diagram) | A diagram showing data entities and how they interact. |
| SRS (Software Requirements Specification) | A document describing system requirements and functionalities. |

## Appendix D: Analysis and Design Models

1. Use Case Diagram

* Shows interactions between Parent and Administration roles.
* Includes actions: Login, Register, Add Child, View Activities, Book Activity, Cancel Booking, Generate Invoice.

1. ERD (Entity Relationship Diagram)

* **Entities:**
* Parent (1 🡪 Many Children, 1 🡪Many Bookings)
* Child (1 🡪 Many Bookings, belongs to Parent)
* Activity (1 🡪 Many Bookings)
* Booking (links Parent, Child, Activity)
* Includes constraints:
* Unique booking per child per day
* Cascading deleted from Parent 🡪 Child 🡪 Booking

1. Class Diagram

* **Classes:**
* Parent
* Child
* Activity
* Booking
* Relationships follow the same structure as ERD.

1. System Architecture Diagram

* Three-layer structure:
* Presentation Layer: HTML, Bootstrap, Templates
* Application Layer: Flask routes, validation, booking logic
* Data Layer: PostgreSQL 16 (Production) / SQLite (Dev) using SQLAlchemy ORM

1. Activity Booking Workflow

* Parent logs in 🡪 Selects child 🡪 Selects activity 🡪 Chooses date 🡪 System checks capacity

🡪 System checks conflict 🡪 Booking is created 🡪 Invoice is generated and mails are sent to all the parties

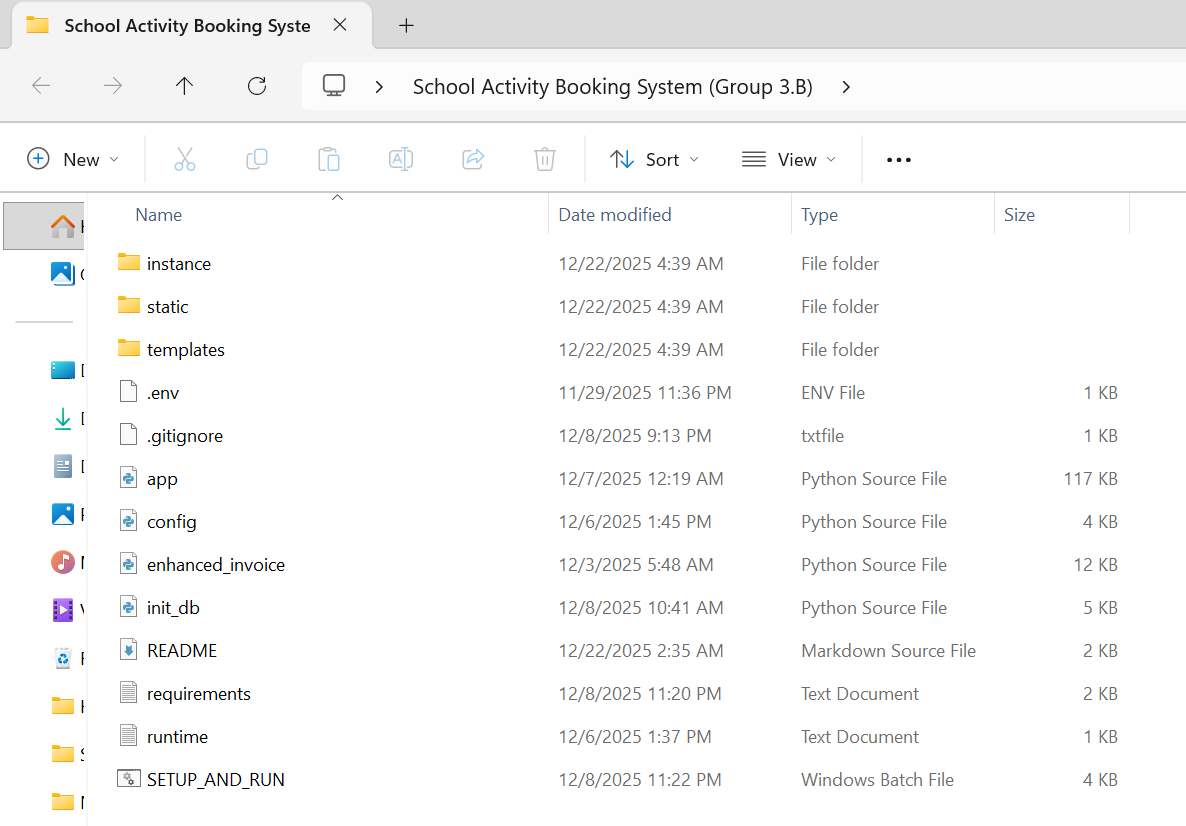
## Appendix E: To Do List

|  |  |  |
| --- | --- | --- |
| **Task** | **Description** | **Status** |
| Implement email notifications | Add email confirmation for bookings and cancellations | Completed |
| Add admin dashboard | Allow administrators to add/edit activities | Completed |
| Improve filtering and sorting | Add filtering by price, day, and availability | Pending |
| Add payment integration | Enable online payment during booking | Future Enhancement |
| Mobile UI optimisation | Improve mobile experience with enhanced layout | Completed |
| Add search functionality | Allow parents to search activities by name | Pending |
| Implement WebSockets | Real-time updates for capacity and bookings | Future Enhancement |
| Expand system testing | Increase test coverage above 95% | Completed |
| Add child photo upload | Optional feature for profile enhancement | Future Enhancement |
| Deployment setup | Deploy system to cloud server (e.g. AWS) | Pending |

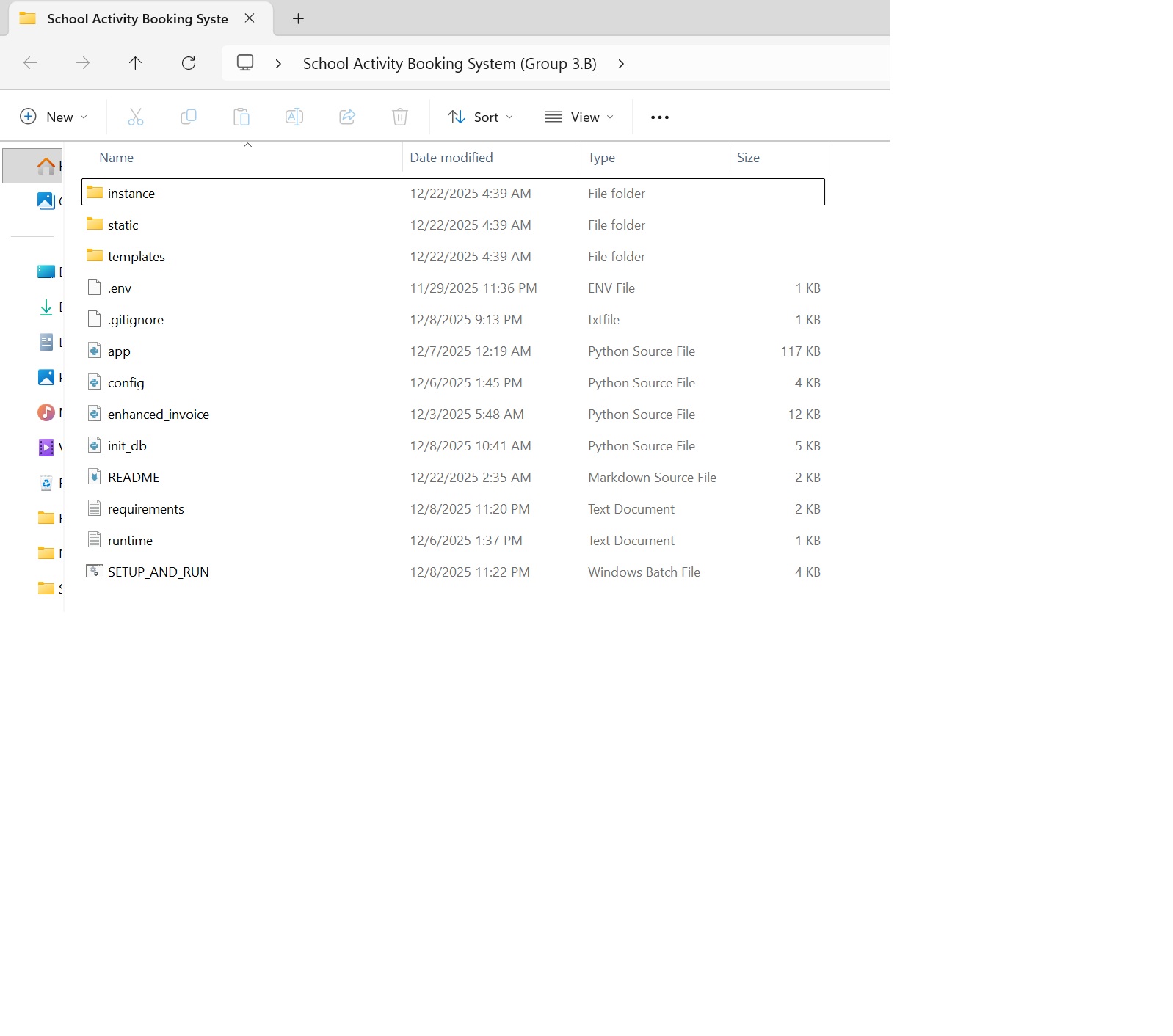
## Appendix F: Source Code

Project Repository URL: <https://github.com/sanchitmahant/School-Activity-Booking-System>

Note: The complete source code, including database models and route logic, is available at the link above.



**Figure 9: Local Project Directory Structure showing Flask Application Files**



**Figure 10: GitHub Repository Proof of Implementation**

**Figure 11: Git Commit History showing Iterative Development and Team Contributions**