Banquet Hall SaaS Platform

Software Requirements Specification

&

Technical Design Document

Version:	1.0
Date:	February 16, 2026
Phase:	Phase 1 - Core Features
Status:	Draft
Technology:	Spring Boot 3 | React 18 | PostgreSQL 16

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this SaaS application is to provide a centralized platform for customers to discover, book, and pay for banquet halls while enabling banquet hall owners and managers to manage registrations, venues, bookings, and payments. The system includes a responsive web-based application accessible on desktop and mobile browsers.

1.2 Scope

The system will support:

- Customer booking journeys (search -> select -> pay -> confirm).
- Banquet hall owner registration, venue management, and payout handling.
- Role-based access for owners, managers, assistants, and SaaS administrators.
- Integration with payment gateways (Stripe) for transactions and refunds.
- Reporting, analytics, and discount management (Phase 2).

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
SaaS Admin	Administrator managing the platform.
OTP	One-Time Password (used for phone authentication).
Venue	A bookable unit within a banquet hall.
Callback	Customer request to be contacted by hall staff.
JWT	JSON Web Token (used for stateless authentication).
RBAC	Role-Based Access Control.
DTO	Data Transfer Object.
JPA	Java Persistence API.

2. Overall Description

2.1 Product Perspective

The application will function as a multi-tenant SaaS platform where multiple banquet halls can self-register and manage their venues. Customers can search for and book available venues through the responsive web application. The platform handles all payment processing, including customer charges, refunds, and payouts to hall owners.

2.2 User Classes and Characteristics

- **Customers:** End-users booking banquet halls. Require simple, guided workflows for searching, booking, and payment.
- **Banquet Hall Owners:** Manage registration, venues, bookings, staff, and payouts. Need comprehensive management dashboards.
- **Managers/Assistants:** Support hall operations with delegated access. Managers have broader permissions than Assistants.
- **SaaS Admins:** Supervise registrations, handle approvals/rejections, and manage platform-wide settings.

2.3 Operating Environment

Web Application: Responsive browser-based application supporting Chrome, Edge, Safari, and Firefox. Optimized for both desktop and mobile viewports.

Server: Cloud-hosted SaaS backend built on Spring Boot 3 with a scalable architecture. PostgreSQL 16 as the primary database. Stripe for payment processing.

3. Functional Requirements

3.1 Authentication & Authorization

FR-1: Customers must be able to log in using phone number (OTP) or username/password.

FR-2: Banquet hall owners, managers, and assistants must authenticate using phone or username/password.

FR-3: Role-based access control must be enforced (Owner, Manager, Assistant, Admin).

3.2 Banquet Hall Registration

FR-4: Owners must be able to register banquet halls by entering hall details, venue details, capacity, availability timeframe, bank account information, and uploading verification documents.

FR-5: Owners must be able to update details and upload new documents until approval.

FR-6: SaaS Admin must be able to approve, reject, or hold applications and request additional info.

FR-7: Only approved banquet halls must be visible in customer searches.

3.3 Venue & Booking Management

FR-8: Owners must define minimum booking slot durations per venue.

FR-9: The system must prevent double bookings for the same time slot.

FR-10: Owners and managers must be able to update pricing by date/time slot.

FR-11: Owners must provide Terms & Conditions (text or document). These must be displayed before payment.

FR-12: Owners and managers must be able to cancel bookings, triggering refunds.

3.4 Payments & Refunds

FR-13: The system must integrate with a payment gateway (Stripe) for customer payments.

FR-14: The system must support payouts to banquet hall bank accounts via Stripe Connect.

FR-15: The system must support full and partial refunds for cancellations.

FR-16: The system must support payment in installments (e.g., two stages: 50% upfront, 50% before event).

3.5 Search & Discovery

FR-17: Customers must be able to search by hall name, date/time availability, capacity, budget, city, zipcode, and search radius.

FR-18: Customers must be able to sort results by price (low to high, high to low) and reviews.

3.6 Customer Interaction (Phase 2)

FR-19: Customers must be able to request a callback from hall staff.

FR-20: Customers must be able to submit amenity requests or questions via chat.

FR-21: Chat must be visible to owner, managers, assistants, and customer.

3.7 Discounts & Promotions (Phase 2)

FR-22: SaaS Admins must be able to create discount coupons (by dates, venues, cities, etc.).

FR-23: Hall owners/managers must be able to create coupons (by dates, multi-date bookings).

FR-24: Customers must be able to apply valid coupons during booking checkout.

3.8 Reporting (Phase 2)

FR-25: Owners must be able to view weekly, monthly, and yearly booking reports.

FR-26: Owners must be able to view booking history for the past 6 months and all future bookings.

4. Non-Functional Requirements

ID	Category	Requirement
NFR-1	Scalability	The system must support 10,000+ concurrent users.
NFR-2	Performance	Search results must return within 2 seconds under normal load.
NFR-3	Security	All sensitive data (passwords, bank accounts) must be encrypted (AES-256 for data at rest, BCrypt for pas
NFR-4	Availability	The system must maintain 99.9% uptime.
NFR-5	Compliance	The system must comply with PCI-DSS for payment security (handled via Stripe).
NFR-6	Cross-Platform	The web application must be responsive and work on desktop, tablet, and mobile browsers.
NFR-7	Data Retention	Booking history must be retained for at least 12 months.

5. System Constraints

C-1: Payment processing depends on the chosen third-party gateway's (Stripe) availability.

C-2: Refund timelines depend on bank/payment processor policies.

C-3: Multi-language support is optional in Phase 1, but architecture must allow future localization.

6. Use Cases

UC-1 Customer Booking Flow

Actors: Customer, System, Payment Gateway

Precondition: Customer has a registered account.

Main Flow:

- 1. Customer logs in with phone number (OTP) or username/password.
- 2. Customer searches for available banquet halls using filters (city, date, capacity, budget).
- 3. System returns list of approved halls matching criteria with sorting options.
- 4. Customer selects a hall and views venue details, pricing, and availability.
- 5. Customer selects a venue, date, and time slot.
- 6. System verifies slot availability (no double booking).
- 7. Customer reviews Terms & Conditions and accepts them.
- 8. Customer chooses payment mode (full or installment).
- 9. Customer enters payment details via Stripe checkout.
- 10. System processes payment and confirms booking.
- 11. System blocks the time slot to prevent further bookings.
- 12. Customer receives booking confirmation.

Postcondition: Booking is confirmed and slot is reserved.

UC-2 Banquet Hall Registration

Actors: Banquet Hall Owner, SaaS Admin

Precondition: Owner has a registered account with OWNER role.

Main Flow:

- 1. Owner logs in to the platform.
- 2. Owner navigates to Hall Registration and enters hall details (name, address, city, zipcode, contact info).
- 3. Owner adds venue details (name, capacity, base pricing, minimum booking duration).
- 4. Owner enters bank account information for payouts.
- 5. Owner uploads verification documents (business license, identity proof, etc.).
- 6. Owner provides Terms & Conditions (text or document upload).
- 7. Owner submits registration for review.
- 8. SaaS Admin receives notification of pending registration.
- 9. SaaS Admin reviews details and documents.
- 10. SaaS Admin approves, rejects (with reason), or places on hold (requesting additional info).
- 11. Upon approval, hall becomes visible in customer searches.

Postcondition: Hall is approved and listed for customers, or rejected/held with feedback.

UC-3 Booking Cancellation & Refund

Actors: Owner/Manager OR Customer, System, Payment Gateway

Precondition: A confirmed booking exists.

Main Flow:

- 1. Owner/Manager or Customer initiates booking cancellation.
- 2. System prompts for cancellation reason.
- 3. System calculates refund amount based on Terms & Conditions and cancellation policy.
- 4. System processes refund via Stripe (full or partial).
- 5. Refund is logged in the payments table.
- 6. Booking status is updated to CANCELLED.
- 7. Time slot is released for new bookings.
- 8. Customer and hall owner are notified.

Postcondition: Booking is cancelled, refund is processed, and slot is available.

UC-4 Discount Creation (Phase 2)

Actors: Owner/Manager OR SaaS Admin

Main Flow:

- 1. Owner/Manager or SaaS Admin defines coupon rules (code, discount %, valid dates, applicable venues/cities).
- 2. Coupon is created and activated in the system.
- 3. Customers can apply the coupon code at checkout.
- 4. System validates coupon and applies discount to the booking total.

7. Technical Architecture

7.1 System Architecture

The system follows a three-tier architecture with clear separation of concerns:

Presentation Layer (Frontend)

React 18 single-page application with TypeScript, served via Vite development server or built as static assets for production. Communicates with the backend exclusively via REST APIs using Axios HTTP client. Tailwind CSS for responsive styling.

Application Layer (Backend)

Spring Boot 3.2 application exposing RESTful APIs. Uses Spring Security with JWT tokens for stateless authentication. Spring Data JPA for database access with Flyway for schema migrations. Stripe Java SDK for payment processing.

Data Layer

PostgreSQL 16 relational database. All entities are managed via JPA with proper indexing and constraints. Sensitive fields (bank account numbers) are encrypted at the application layer using AES-256 via JPA AttributeConverter.

Architecture Flow:

- React App --> Axios HTTP Client --> Spring Boot REST Controllers
- Controllers --> Spring Security (JWT Validation) --> Service Layer
- Service Layer --> Spring Data JPA Repositories --> PostgreSQL Database
- Service Layer --> Stripe SDK --> Stripe Payment Gateway
- Service Layer --> File Storage Service --> Local Filesystem (S3 in production)

7.2 Technology Stack

Component	Technology
Backend Framework	Spring Boot 3.2 with Java 17
Build Tool	Apache Maven
Security	Spring Security 6 + JWT (access + refresh tokens)
ORM	Spring Data JPA (Hibernate)
DB Migrations	Flyway
Payment Gateway	Stripe (Payment Intents API, Refunds API, Connect)
Frontend Framework	React 18 with TypeScript
Build Tool (FE)	Vite 5

CSS Framework	Tailwind CSS 3
Routing	React Router v6
HTTP Client	Axios
Forms	React Hook Form
Icons	Heroicons
Database	PostgreSQL 16
Containerization	Docker Compose (for PostgreSQL)
Password Hashing	BCrypt
Data Encryption	AES-256 (for bank details)

7.3 Project Structure

The project is organized as a monorepo with separate backend and frontend directories:

```
ban/
  backend/                        (Spring Boot 3.2 + Maven)
    pom.xml
    src/main/java/com/banquet/
      BanquetApplication.java     (Main application class)
      config/                     (SecurityConfig, CorsConfig, StripeConfig)
      controller/                 (REST API controllers)
      dto/                        (Request/Response DTOs)
      entity/                     (JPA entities)
      enums/                      (Role, BookingStatus, HallStatus, etc.)
      exception/                  (GlobalExceptionHandler, custom exceptions)
      repository/                 (Spring Data JPA repositories)
      security/                   (JwtTokenProvider, JwtAuthFilter)
      service/                    (Business logic services)
    src/main/resources/
      application.yml             (App configuration)
      db/migration/               (Flyway SQL migration scripts)
  frontend/                       (React 18 + Vite + TypeScript)
    package.json
    src/
      api/                        (Axios instance + API service modules)
      components/                 (Reusable UI components)
      context/                    (AuthContext, global state)
      hooks/                      (Custom React hooks)
      pages/                      (Page-level components by role)
        auth/                     (Login, Register)
        customer/                 (Search, HallDetail, Booking, Payment)
        owner/                    (Dashboard, HallRegistration, Venues, Bookings)
        admin/                    (HallApprovals, Dashboard)
      types/                      (TypeScript interfaces)
      App.tsx                     (Root component with routing)
```

```
docker-compose.yml              (PostgreSQL + pgAdmin)
README.md                       (Setup & run instructions)
```

8. Database Design

8.1 Entity Relationships

The database consists of 8 core entities with the following relationships:

- USERS (1) ---> (many) BANQUET_HALLS: An owner can register multiple halls.
- USERS (1) ---> (many) BOOKINGS: A customer can make multiple bookings.
- USERS (many) <---> (many) BANQUET_HALLS via HALL_STAFF: Staff assignments.
- BANQUET_HALLS (1) ---> (many) VENUES: A hall contains multiple venues.
- BANQUET_HALLS (1) ---> (many) HALL_DOCUMENTS: A hall has verification documents.
- VENUES (1) ---> (many) VENUE_PRICING: Custom pricing per date/slot.
- VENUES (1) ---> (many) BOOKINGS: A venue has multiple bookings.
- BOOKINGS (1) ---> (many) PAYMENTS: A booking can have multiple payments (installments).

8.2 Table Schemas

Table: users

Column	Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Auto-generated unique ID
phone	VARCHAR(20)	UNIQUE, NOT NULL	Phone number
email	VARCHAR(255)	UNIQUE	Email address
password_hash	VARCHAR(255)	NOT NULL	BCrypt hashed password
full_name	VARCHAR(255)	NOT NULL	User's full name
role	VARCHAR(20)	NOT NULL	CUSTOMER/OWNER/MANAGER/ASSISTANT/ADMIN
phone_verified	BOOLEAN	DEFAULT false	OTP verification status
created_at	TIMESTAMP	DEFAULT NOW()	Record creation time
updated_at	TIMESTAMP	DEFAULT NOW()	Last update time

Table: banquet_halls

Column	Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Auto-generated unique ID
owner_id	BIGINT	FK -> users(id)	Hall owner reference
name	VARCHAR(255)	NOT NULL	Hall name
description	TEXT		Hall description
address	VARCHAR(500)	NOT NULL	Street address

city	VARCHAR(100)	NOT NULL	City
state	VARCHAR(100)		State/Province
zipcode	VARCHAR(20)	NOT NULL	Postal/Zip code
latitude	DOUBLE		GPS latitude
longitude	DOUBLE		GPS longitude
phone	VARCHAR(20)		Hall contact phone
email	VARCHAR(255)		Hall contact email
status	VARCHAR(20)	DEFAULT 'PENDING'	PENDING/APPROVED/REJECTED/ON_HOLD
terms_conditions	TEXT		T&C text
bank_account_name	VARCHAR(255)		Account holder name
bank_acct_num_enc	VARCHAR(500)		AES-256 encrypted
bank_routing_enc	VARCHAR(500)		AES-256 encrypted
admin_notes	TEXT		Admin review notes
created_at	TIMESTAMP	DEFAULT NOW()	Record creation time
updated_at	TIMESTAMP	DEFAULT NOW()	Last update time

Table: hall_documents

Column	Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Auto-generated unique ID
hall_id	BIGINT	FK -> banquet_halls	Parent hall reference
document_type	VARCHAR(50)	NOT NULL	e.g. BUSINESS_LICENSE, ID_PROOF
file_path	VARCHAR(500)	NOT NULL	Server file path / URL
uploaded_at	TIMESTAMP	DEFAULT NOW()	Upload timestamp

Table: venues

Column	Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Auto-generated unique ID
hall_id	BIGINT	FK -> banquet_halls	Parent hall reference
name	VARCHAR(255)	NOT NULL	Venue name
description	TEXT		Venue description
capacity	INTEGER	NOT NULL	Max guest capacity
min_booking_hrs	INTEGER	NOT NULL, DEFAULT 2	Min booking duration
base_price_per_hr	DECIMAL(10,2)	NOT NULL	Default hourly rate
image_urls	TEXT		JSON array of image URLs
active	BOOLEAN	DEFAULT true	Is venue bookable

created_at	TIMESTAMP	DEFAULT NOW()	Record creation time
updated_at	TIMESTAMP	DEFAULT NOW()	Last update time

Table: venue_pricing

Column	Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Auto-generated unique ID
venue_id	BIGINT	FK -> venues(id)	Parent venue reference
effective_date	DATE	NOT NULL	Date this pricing applies
slot_start	TIME	NOT NULL	Slot start time
slot_end	TIME	NOT NULL	Slot end time
price	DECIMAL(10,2)	NOT NULL	Price for this slot

Table: bookings

Column	Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Auto-generated unique ID
customer_id	BIGINT	FK -> users(id)	Booking customer
venue_id	BIGINT	FK -> venues(id)	Booked venue
booking_date	DATE	NOT NULL	Event date
start_time	TIME	NOT NULL	Slot start time
end_time	TIME	NOT NULL	Slot end time
total_amount	DECIMAL(10,2)	NOT NULL	Total booking cost
paid_amount	DECIMAL(10,2)	DEFAULT 0	Amount paid so far
status	VARCHAR(20)	DEFAULT 'PENDING'	PENDING/CONFIRMED/CANCELLED/DONE
payment_mode	VARCHAR(20)	NOT NULL	FULL / INSTALLMENT
cancel_reason	TEXT		Cancellation reason
created_at	TIMESTAMP	DEFAULT NOW()	Record creation time
updated_at	TIMESTAMP	DEFAULT NOW()	Last update time

Table: payments

Column	Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Auto-generated unique ID
booking_id	BIGINT	FK -> bookings(id)	Parent booking
amount	DECIMAL(10,2)	NOT NULL	Payment amount
payment_type	VARCHAR(20)	NOT NULL	FULL/INSTALLMENT_1/INSTALLMENT_2

status	VARCHAR(20)	DEFAULT 'PENDING'	PENDING/SUCCESS/FAILED/REFUNDED
stripe_pi_id	VARCHAR(255)		Stripe PaymentIntent ID
created_at	TIMESTAMP	DEFAULT NOW()	Record creation time

Table: hall_staff

Column	Type	Constraints	Description
id	BIGSERIAL	PRIMARY KEY	Auto-generated unique ID
hall_id	BIGINT	FK -> banquet_halls	Hall reference
user_id	BIGINT	FK -> users(id)	Staff user reference
role	VARCHAR(20)	NOT NULL	MANAGER / ASSISTANT
created_at	TIMESTAMP	DEFAULT NOW()	Assignment date

Key Database Indexes

- idx_users_phone ON users(phone) - Fast phone number lookup for login
- idx_users_email ON users(email) - Fast email lookup
- idx_halls_status ON banquet_halls(status) - Filter by approval status
- idx_halls_city ON banquet_halls(city) - Search by city
- idx_halls_zipcode ON banquet_halls(zipcode) - Search by zipcode
- idx_venues_hall ON venues(hall_id) - Venues per hall lookup
- idx_bookings_venue_date ON bookings(venue_id, booking_date, status) - Double-booking prevention
- idx_bookings_customer ON bookings(customer_id) - Customer's bookings lookup
- idx_payments_booking ON payments(booking_id) - Payments per booking lookup

9. API Design

All APIs are RESTful and return JSON. Authentication is via Bearer JWT token in the Authorization header.

9.1 Authentication API (/api/auth)

Method	Endpoint	Description	Access	Payload/Notes
POST	/api/auth/register	Register new user (customer or ow	Public	{ phone, email, password, fullName, role }
POST	/api/auth/login	Login with credentials or OTP	Public	{ phone, password } or { phone, otp }
POST	/api/auth/refresh	Refresh JWT token	Public	{ refreshToken }
GET	/api/auth/me	Get current user profile	Authenticated	Bearer token

9.2 Banquet Halls API (/api/halls)

Method	Endpoint	Description	Access	Payload/Notes
POST	/api/halls	Register new hall	OWNER	Multipart: JSON + files
PUT	/api/halls/{id}	Update hall details	OWNER	JSON body
GET	/api/halls/{id}	Get hall details	Public	-
GET	/api/halls/my	List owner's halls	OWNER	-
GET	/api/halls/admin/pending	List pending halls	ADMIN	Pagination params
PUT	/api/halls/admin/{id}/status	Approve/Reject/Hold	ADMIN	{ status, notes }

9.3 Venues API (/api/halls/{hallId}/venues)

Method	Endpoint	Description	Access	Payload/Notes
POST	/api/halls/{hId}/venues	Create venue	OWNER/MANAGER	JSON body
PUT	/api/halls/{hId}/venues/{vId}	Update venue	OWNER/MANAGER	JSON body
GET	/api/halls/{hId}/venues	List hall venues	Public	-
GET	/api/halls/{hId}/venues/{vId}	Get venue detail	Public	-
PUT	.../{vId}/pricing	Set/update pricing	OWNER/MANAGER	JSON array of pricing

9.4 Bookings API (/api/bookings)

Method	Endpoint	Description	Access	Payload/Notes
POST	/api/bookings	Create booking	CUSTOMER	{ venueId, date, start, end, mode }

GET	/api/bookings/my	Customer's bookings	CUSTOMER	Pagination params
GET	/api/bookings/hall/{hId}	Hall's bookings	OWNER/MANAGER	Pagination + filters
GET	/api/bookings/{id}	Booking details	Authenticated	-
PUT	/api/bookings/{id}/cancel	Cancel booking	OWNER/MGR/CUST	{ reason }

9.5 Payments API (/api/payments)

Method	Endpoint	Description	Access	Payload/Notes
POST	/api/payments/create-intent	Create PaymentIntent	CUSTOMER	{ bookingId, amount, type }
POST	/api/payments/confirm	Confirm payment	CUSTOMER	{ paymentIntentId }
POST	/api/payments/{bId}/refund	Process refund	OWNER/MANAGER	{ amount (optional) }
GET	/api/payments/booking/{bId}	Booking payments	Authenticated	-

9.6 Search API (/api/search)

Method	Endpoint	Description	Access	Payload/Notes
GET	/api/search/halls	Search halls	Public	Query params below

Search Query Parameters:

- name (string) - Hall name partial match
- city (string) - City filter
- zipcode (string) - Zipcode filter
- date (date) - Availability date filter (YYYY-MM-DD)
- startTime / endTime (time) - Time slot filter (HH:mm)
- minCapacity (integer) - Minimum venue capacity
- maxBudget (decimal) - Maximum price per hour
- sortBy (string) - price_asc, price_desc, rating
- page (integer) - Page number (0-indexed)
- size (integer) - Page size (default 20)

10. Key Implementation Details

10.1 Double-Booking Prevention (FR-9)

The system uses pessimistic locking (SELECT ... FOR UPDATE) at the database level to prevent race conditions when two customers attempt to book the same slot simultaneously.

SQL overlap check query:

```
SELECT COUNT(*) FROM bookings
WHERE venue_id = :venueId
  AND booking_date = :date
  AND status IN ('PENDING', 'CONFIRMED')
  AND start_time < :endTime
  AND end_time > :startTime
FOR UPDATE;
```

This is wrapped in a @Transactional service method. If count > 0, a ConflictException is thrown. The FOR UPDATE clause locks matching rows until the transaction completes, serializing concurrent requests.

10.2 Role-Based Access Control (FR-3)

Spring Security with method-level @PreAuthorize annotations enforces access control:

- ROLE_CUSTOMER: Can search halls, create bookings, make payments, view own bookings.
- ROLE_OWNER: Can register halls, manage venues, manage bookings, manage staff, view payouts.
- ROLE_MANAGER: Can manage venues and bookings for halls they are assigned to.
- ROLE_ASSISTANT: Can view bookings and venue details for halls they are assigned to.
- ROLE_ADMIN: Can approve/reject halls, manage platform settings, view all data.

Staff access (MANAGER/ASSISTANT) is verified by checking the hall_staff table to ensure the user is assigned to the hall they are trying to access.

10.3 JWT Authentication Flow

Authentication uses a dual-token approach:

- Access Token: Short-lived (15 minutes), used for API authentication via Authorization header.
- Refresh Token: Long-lived (7 days), used to obtain new access tokens without re-login.
- Tokens are signed using HMAC-SHA256 with a configurable secret key.
- Token payload includes: userId, role, and expiration timestamp.
- JwtAuthFilter intercepts all requests, validates the token, and sets the SecurityContext.

10.4 Installment Payments (FR-16)

Bookings support two payment modes:

FULL: Customer pays the entire amount at booking time. Single PaymentIntent is created. Booking is confirmed upon successful payment.

INSTALLMENT: Two-stage payment. First installment (50%) is paid at booking time. Booking is confirmed after first installment succeeds. Second installment (remaining 50%) must be paid before the event date. System tracks paid_amount vs total_amount.

10.5 Encrypted Bank Details (NFR-3)

Sensitive bank account data is encrypted using AES-256 at the application layer. A custom JPA AttributeConverter handles transparent encryption on write and decryption on read. The encryption key is stored in environment variables, never in source code.

10.6 File Upload for Documents

Hall verification documents are uploaded via multipart form data. In Phase 1, files are stored on the local filesystem under a configurable uploads directory. The file storage service is abstracted behind an interface to allow migration to AWS S3 or similar cloud storage in production. File paths are stored in the hall_documents table.

11. Frontend Design

11.1 Design Principles

- Responsive: All pages work on desktop (1200px+), tablet (768px-1199px), and mobile (320px-767px).
- Accessible: Semantic HTML, proper ARIA labels, keyboard navigation support.
- Modern: Clean, professional UI with Tailwind CSS utility classes.
- Role-Based Navigation: Sidebar/navbar adapts based on user role.
- Protected Routes: React Router guards prevent unauthorized page access.

11.2 Customer Pages

- **Login / Register:** Phone + OTP or email/password forms. Role selection for registration.
- **Home:** Hero section with search bar. Featured/popular halls carousel.
- **Search Results:** List/grid view with filters sidebar (city, date, capacity, budget). Sorting dropdown. Pagination.
- **Hall Detail:** Hall info, photo gallery, list of venues with capacity and pricing. Calendar view for availability.
- **Booking Checkout:** Selected venue + slot summary. T&C acceptance checkbox. Payment mode selection (full/installment). Stripe Elements payment form.
- **My Bookings:** List of bookings with status badges. Cancel action with confirmation dialog. Payment history per booking.

11.3 Owner Pages

- **Dashboard:** Overview cards: total bookings, revenue, pending bookings. Recent bookings table.
- **Hall Registration:** Multi-step form: Step 1 - Hall details, Step 2 - Venue details, Step 3 - Bank info, Step 4 - Document upload, Step 5 - T&C, Step 6 - Review & Submit.
- **Venue Management:** CRUD table for venues. Inline pricing editor with date picker.
- **Booking Management:** Filterable table of all bookings across venues. Cancel with refund action.
- **Staff Management:** Add/remove managers and assistants. Role assignment per hall.

11.4 Admin Pages

- **Registration Approvals:** List of pending hall registrations. Detail view with documents. Approve/Reject/Hold buttons with notes field.
- **Dashboard:** Platform statistics: total halls, total bookings, total revenue, active users.

12. Phase 1 Scope & Exclusions

12.1 Phase 1 Inclusions

- Authentication: JWT-based login with simulated OTP and username/password.
- Authorization: Full RBAC with Owner, Manager, Assistant, Customer, Admin roles.
- Hall Registration: Multi-step registration with document upload and admin approval workflow.
- Venue Management: CRUD operations with capacity, pricing, and availability management.
- Booking Management: Search, book, confirm, cancel with double-booking prevention.
- Payment Processing: Stripe integration for full and installment payments.
- Refund Processing: Full and partial refunds via Stripe.
- Search & Discovery: Filter by city, date, capacity, budget with sorting and pagination.
- Responsive Web App: Works on desktop, tablet, and mobile browsers.

12.2 Phase 2 (Deferred Features)

- Chat/Messaging (FR-19, FR-20, FR-21): Real-time chat between customers and hall staff.
- Discount Coupons (FR-22, FR-23, FR-24): Coupon creation and application at checkout.
- Reporting & Analytics (FR-25, FR-26): Weekly/monthly/yearly booking and revenue reports.
- Callback Requests (FR-19): Customer callback request feature.
- Real SMS OTP: Integration with Twilio or similar SMS provider (simulated in Phase 1).
- Multi-language Support: Internationalization/localization framework.
- Native Mobile Apps: React Native iOS and Android applications.
- Email Notifications: Transactional emails for booking confirmations, cancellations, etc.
- Review/Rating System: Customer reviews and ratings for halls.

12.3 Architecture Preparedness for Phase 2

While Phase 2 features are deferred, the Phase 1 architecture is designed to accommodate them:

- Service layer abstraction allows adding new business logic modules without controller changes.
- Database schema supports extension with new tables (coupons, messages, reviews) via Flyway migrations.
- File storage interface enables seamless migration from local storage to S3.
- OTP service interface allows swapping simulated OTP with real SMS provider.
- Internationalization-ready: All user-facing strings can be externalized using React i18n libraries.
- WebSocket support can be added to Spring Boot for real-time chat features.