Software Requirements Specification Template

Software Engineering

The following annotated template shall be used to complete the Software Requirements Specification (SRS) assignment.

Template Usage:

Text contained within angle brackets ('<', '>') shall be replaced by your project-specific information and/or details. For example, <Project Name> will be replaced with either 'Smart Home' or 'Sensor Network'.

Italicized text is included to briefly annotate the purpose of each section within this template. This text should not appear in the final version of your submitted SRS.

This cover page is not a part of the final template and should be removed before your SRS is submitted.

FlickFast

Software Requirements Specification

4.0.0

10.23.25

Group 10

Duffy Adam
Soham Kulkarni
Brandon Helm

Prepared for
CS 250- Introduction to Software Systems

Instructor: Gus Hanna, Ph.D.
Fall 2025

Revision History

Date	Description	Author	Comments
10-08-25	<Version 1.00>	Duffy Adams	Imported UML + Diagrams
10-09-25	<Version 1.01>	Brandon Helm	Sections 3.7 - 5.4 Updated
10-20-25	<Version 3.00>	Duffy Adams	Section 7 + Updated TOC
10-23-25	<Version 4.00>	Brandon Helm	Section 7.0 - 7.9 updated/reformatted

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date
	Duffy Adam	Software Eng.	
	Soham Kulkarni	Software Eng.	
	Brandon Helm	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

Table of Contents

1. Introduction

The purpose of this section is to provide an overview of the FlickFast software system, its purpose, scope, terminology, references, and the structure of this Software Requirements Specification (SRS). This document contains all the information required by software engineers to design, develop, test, and deploy the FlickFast ticketing system.

1.1 Purpose

The purpose of this SRS is to define the requirements for **FlickFast**, a **single-theatre ticketing system**. The intended audience includes:

- **Software engineers and developers**, who will use this document to implement the system.
- **Testers and quality assurance teams**, who will validate compliance with requirements.
- **Project managers and stakeholders**, who need to confirm scope and goals.
- **Theatre management and staff**, who will interact with the administrative and validation components of the system.

The SRS ensures a clear, shared understanding of what FlickFast will and will not provide, minimizing ambiguity during implementation.

1.2 Scope

1.2.1 Product Identification

The software product is named **FlickFast Ticketing System**.

1.2.2 Product Capabilities

FlickFast will:

- Allow customers to browse movies and showtimes for a **single theatre**.
- Provide **real-time seat selection** using interactive seat maps.
- Support secure **checkout** with loyalty point redemption and promotional codes.
- Deliver tickets **only as QR codes**, accessible through email and in-app display.
- Enable customers to request **refunds and exchanges** within theatre policy, with staff override options.
  Provide staff tools to **validate tickets** quickly at entry and manage showtimes.

FlickFast will not:

- Support multiple theatres or locations.
- Sell concessions (food or drinks).
  Deliver tickets via Apple Wallet, Google Wallet, or paper printing.
- Provide advanced analytics, marketing features, or detailed accessibility accommodations in the first release.

1.2.3 Application of the Software

FlickFast will be deployed as a **web and mobile application** for customers and staff of a single theatre. The system's core application is to streamline ticket purchase and entry validation by prioritizing speed and simplicity.

Benefits, objectives, and goals:

- Customers can complete a purchase in **under two minutes** from showtime selection to confirmation.
- Seat maps render in **less than 400 milliseconds**, enabling quick seat selection.
- QR ticket validation processes in **under 250 milliseconds**, ensuring fast entry at the theatre doors.
- A **loyalty program** incentivizes repeat business by awarding and redeeming points for ticket purchases.
- A **reliable refund and exchange mechanism** improves customer satisfaction and reduces staff friction.
- Strict enforcement of **no seat overselling** ensures fairness and trust in the system.

The scope of FlickFast is limited to **core ticketing functionality with speed as its primary feature**, while deliberately excluding advanced or multi-theatre features that would increase complexity and reduce performance focus.

1.3 Definitions, Acronyms, and Abbreviations

To ensure clarity and consistency, this subsection defines the key terms, acronyms, and abbreviations used throughout the FlickFast SRS.

- **FlickFast** – The name of the single-theatre ticketing system being developed.
- **QR Code (Quick Response Code)** – A two-dimensional barcode used to deliver and validate digital tickets.
- **Loyalty Points** – Reward credits earned by customers for ticket purchases, redeemable for discounts on future transactions.
- **Checkout** – The process by which a customer finalizes ticket selection, applies promotions or loyalty points, and submits payment.
- **Seat Map** – The interactive visual layout of an auditorium showing available, held, and reserved seats.
- **Hold** – A temporary reservation of seats (e.g., five minutes) while a customer completes checkout.
- **Redemption** – The act of scanning and validating a QR code ticket at theatre entry.
- **Staff App** – The administrative interface used by theatre employees to validate tickets, manage showtimes, and process refunds or exchanges.
- **Refund** – A return of payment to the customer when tickets are cancelled within theatre policy.
- **Exchange** – The replacement of one set of tickets with another, possibly adjusting price or loyalty points.
- **PCI DSS (Payment Card Industry Data Security Standard)** – A compliance standard governing the secure handling of payment information.

- **OTP (One-Time Password)** – A temporary security code used to verify identity during login or sensitive actions.
- **Throughput** – The measure of how many transactions or scans the system can process within a set time, directly linked to speed performance goals.

1.4 References

The following documents and standards are referenced in this SRS and provide supporting information for the design and implementation of the FlickFast Ticketing System:

1. **PCI DSS v4.0** – Payment Card Industry Data Security Standard, 2022. Published by the PCI Security Standards Council. Provides mandatory requirements for secure handling of payment card transactions.
2. **Stripe Payments API Documentation** – Current release, available from https://stripe.com/docs. Used as the primary payment gateway integration reference.
3. **IEEE Standard 830-1998** – IEEE Recommended Practice for Software Requirements Specifications. Provides guidelines for the structure and content of SRS documents.
4. **ISO/IEC 27001:2022** – Information Security Management Standard. Provides principles for managing sensitive data and ensuring system security.
5. **W3C Web Performance Working Group Recommendations** – Latest publications from the World Wide Web Consortium on responsive web applications, available at https://www.w3.org/. Supports FlickFast's performance requirements.

1.5 Overview

The remainder of this Software Requirements Specification (SRS) provides a complete description of the FlickFast Ticketing System.

- **Section 2 – General Description** introduces the system context, its main functions, user characteristics, operating environment, and constraints.
- **Section 3 – Specific Requirements** provides detailed functional and non-functional requirements, external interface descriptions, and use case definitions.
- **Section 4 – Analysis Models** contains diagrams and models, including use case scenarios, sequence flows, and state transitions that illustrate system behavior.
  Section 5 – Change Management Process defines how modifications to the requirements will be proposed, reviewed, approved, and tracked throughout the system's lifecycle.

This organization ensures that readers can move from a high-level understanding of FlickFast's purpose and scope (Section 1), into a deeper exploration of its context and functions (Section 2), followed by explicit requirements (Section 3), supporting models (Section 4), and maintenance processes (Section 5).

2. General Description

This section describes the general factors that influence the FlickFast Ticketing System. It provides context, user characteristics, and constraints that help clarify the requirements described in later sections.

2.1 Product Perspective

FlickFast is a standalone, single-theatre ticketing system. It will operate as both a web application and a mobile application, providing customers with a seamless way to browse showtimes, select seats, purchase tickets, and access QR codes for entry.

- The system interacts with external services for:
- Payment processing via Stripe (or a similar PCI-compliant provider).
- Email delivery for order confirmations and QR code tickets.
- It also includes a staff-facing interface for ticket validation and showtime management. Unlike multi-theatre solutions, FlickFast does not require synchronization across different venues, which simplifies its architecture and allows for greater speed optimization.

2.2 Product Functions

At a high level, FlickFast provides the following core functions:

- **Browse Movies and Showtimes** – Customers can view available movies, schedules, and auditorium details.
- **Seat Selection** – Customers can interact with a real-time seat map to select and hold seats during checkout.
- **Checkout** – Customers finalize ticket purchases by entering payment details, applying promo codes, and redeeming loyalty points.
- **Ticket Delivery** – Tickets are delivered **only as QR codes** via email and accessible within the app.
- **Ticket Validation** – Staff validate QR codes at entry, ensuring authenticity and preventing re-use.
- **Refunds and Exchanges** – Customers can request refunds or exchanges within policy limits; staff may override or assist as needed.
- **Loyalty Program** – Customers can earn and redeem points for discounts on ticket purchases.
- **Theatre Management Tools** – Staff and managers can configure showtimes, pricing, and approve refund/exchange requests.

2.3 User Characteristics

The system is designed for the following user groups:

- **Customers (Guests)** – Occasional users who may purchase without creating an account. They expect a fast and simple checkout.
- **Customers (Members)** – Registered users who participate in the loyalty program, saving preferences and payment information for faster checkout.
- **Staff (Ushers)** – Use the staff interface to scan and validate QR code tickets quickly at entry.
- **Staff (Managers/Admins)** – Configure showtimes, manage pricing, and approve refund/exchange exceptions.

All users are assumed to have basic familiarity with mobile apps or web browsers. No specialized training is required.

2.4 General Constraints

- **Single Theatre Limitation** – FlickFast supports one theatre location only.
- **Performance First** – System must prioritize speed (sub-second responses for seat maps and ticket validation).
- **PCI Compliance** – Payment processing must follow PCI DSS requirements.
- **Ticket Format Restriction** – Only QR codes are supported; no printed tickets or wallet passes.
- **No Accessibility Extensions** – Accessibility is limited to standard web/mobile compliance; advanced ADA features are out of scope for this release.
- **Online Connectivity** – The system assumes reliable internet access for customers and staff, though limited offline scanning options may be added later.

2.5 Assumptions and Dependencies

- Payment providers (e.g., Stripe) are available and functioning.
- Email delivery services (SMTP or third-party providers) maintain reliable uptime.
- Theatre staff maintain accurate showtime schedules and seat maps.
- Customers have devices capable of displaying QR codes (smartphones or printed emails if needed).
- Future expansions, such as multi-theatre support or concessions, may be built on top of FlickFast but are not part of the initial release.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- **Customer Web Application**: Responsive layout for browsing, seat maps, checkout, and receipts with QR codes. Seat maps will be keyboard navigable and include clear legends for the screen, aisles, taken seats, available seats, and accessible seats.
- **Customer Mobile Application**: Identical flows to the web application. Includes a persistent wallet page that displays active QR codes offline for up to 24 hours after the last sync.
- **Staff Validator Application**: Camera-based QR scanner interface with a large Pass/Fail result screen, ticket details preview (seat, showtime, auditorium), and redemption status.
- **Manager Console**: Interface for showtime scheduling, pricing rules, refund and exchange requests, loyalty adjustments, and audit log viewing.

3.1.2 Hardware Interfaces

Mobile Device Cameras: Used for QR scanning by both customers (ticket display) and staff (validation).

Optional USB or Bluetooth Barcode Scanners: Supported as input devices in the staff validator application.

Thermal Receipt Printer: Optional for box office use. Prints a receipt containing order ID and ticket link (not a scannable ticket).

3.1.3 Software Interfaces

Payment Gateway: Stripe-compatible API for charges, refunds, partial refunds, and idempotent requests.

Email Delivery Service: SMTP or API-based provider used for sending order confirmations and QR ticket links, with retry and backoff mechanisms.

Optional SMS Provider: Sends ticket confirmation links and reminders; configurable per theatre.

Analytics Export: Daily CSV export via secure download endpoint, including orders, tickets, refunds, and loyalty transactions.

Authentication: JWT session tokens for customers and staff. Email-based OTP used for sensitive actions such as refunds or exchanges.

Time and Currency Handling: Server maintains UTC; local theatre timezone is applied in views. Currency formatting will follow the theatre's locale.

3.1.4 Communications Interfaces

Protocols: HTTPS for all API calls (HTTP/2 preferred). WebSockets for real-time seat map updates and validator responses.

Authentication: Bearer JWT included in Authorization headers. CSRF protection enforced for browser-based forms using cookies.

Rate Limiting: Per-IP and per-account rate limits applied to checkout, refund actions, and validator scan attempts.

Error Format: JSON-based responses with standardized problem details (code, message, remediation hint).

Performance Targets:

- Seat map data fetch: 95th percentile under 400 ms.
- Validator response: 95th percentile under 250 ms (measured at theatre network edge).

3.2 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

3.2.1 Browse and Select Showtime

3.2.1.1 Introduction

Customers can browse available movies and showtimes for the theatre. Upon selecting a showtime, the system displays an interactive seat map.

3.2.1.2 Inputs

- Date and movie filter
- Selected showtime ID

3.2.1.3 Processing

- Retrieve showtime details including schedule, pricing, and available seats
- Render an interactive seat map with real-time availability

3.2.1.4 Outputs

- List of available showtimes with prices
- Seat map for the selected showtime

3.2.1.5 Error Handling

- If showtime is unavailable, display a message and suggest alternatives

3.2.2 Seat Reservation Hold

3.2.2.1 Introduction

Seats are temporarily held during checkout to prevent overselling.

3.2.2.2 Inputs

- Showtime ID
- Seat IDs
- User session ID

3.2.2.3 Processing

- Validate seat availability
- Place a temporary hold for five minutes
- Display a countdown timer

3.2.2.4 Outputs

- Hold confirmation with expiry time

3.2.2.5 Error Handling

- If seats are no longer available, update seat map and notify customer

3.2.3 Seat Swap Between Users

3.2.3.1 Introduction

Customers can exchange their reserved or purchased seats with other users for the same showtime. This provides flexibility if groups want to sit together or if individuals prefer different seat positions.

3.2.3.2 Inputs

- Original Order ID and Seat ID(s)
- Target User ID and Seat ID(s)
- Swap request confirmation from both users

3.2.3.3 Processing

- Validate that both seats belong to confirmed, valid tickets for the same showtime and auditorium
- Ensure neither ticket has already been redeemed

- Update ticket records to reflect new seat assignments
- Log the swap in the system audit trail

3.2.3.4 Outputs

- Updated QR codes for both users with swapped seat assignments
- Confirmation messages to both users (in-app and email)

3.2.3.5 Error Handling

- If either ticket is already redeemed, deny the swap with error "Ticket Already Used"
- If seats belong to different showtimes or auditoriums, deny swap with error "Swap Not Allowed"
- If the second user does not confirm within two minutes, cancel the swap and restore original assignments

3.3 Use Cases

3.3.1 Use Case #1: Purchase Tickets

Purpose: Describe a typical customer purchase flow from browsing showtimes to receiving QR tickets.

Actors: Customer (Guest or Member)
Description: Customer searches for a showtime, selects seats, completes checkout, and receives QR code tickets.
Preconditions: Showtime exists with available seats.
Basic Flow:

1. The customer selects a movie and showtime.
2. The customer selects seats from the seat map.
3. The system holds seats.
4. Customer proceeds to checkout, applies loyalty points or promo codes if applicable, and provides payment information.
5. The system processes payment, commits seats, and generates QR tickets.
6. Customers receive confirmation and tickets via email and app. Alternate Flows:

- Payment fails → system releases seats and prompts retry.
- Customer applies insufficient loyalty points → system denies redemption.

Postconditions: Seats are reserved and tickets are issued.

3.3.2 Use Case #2: Validate Entry

Purpose: Describe staff validation of QR tickets during customer entry.

Actors: Staff (Usher)
Description: Staff validates customer QR code tickets at theatre entry.

Preconditions: Ticket has been purchased and not redeemed.
Basic Flow:

1. Staff scans QR code using a validator app.
2. The system verifies authenticity and redemption status.
3. System marks ticket as redeemed.
4. Staff sees "Pass" confirmation.

Alternate Flows:

- Duplicate scan → system displays "Already Redeemed."
- Invalid code → system displays "Invalid Ticket."
- Postconditions: Valid tickets are redeemed, invalid attempts are rejected.

3.3.3 Use Case #3: Refund or Exchange Tickets

Purpose: Customer requests refund or exchanges tickets, with policy checks and optional manager override.

Actors: Customer, Staff (Manager override if required)
Description: Customer requests a refund or exchange.
Preconditions: Tickets exist and meet policy requirements.
Basic Flow:

1. Customer requests refund/exchange in-app.
2. System checks policy eligibility.
3. If valid, system processes refund or assigns new seats.
4. Loyalty points are adjusted accordingly.

Alternate Flows:

- Request outside policy → system denies unless manager override is used.

Postconditions: Order status updated, confirmation sent to customer.

3.3.4 Use Case #4: Seat Swap Between Users

Purpose: Two customers swap seat assignments for the same showtime, with confirmation and time limits.

Actors: Two Customers
Description: Two customers swap seats for the same showtime.
Preconditions: Both tickets are valid, unredeemed, and belong to the same showtime.
Basic Flow:

1. Customer A requests a swap with Customer B.
2. Customer B receives notification and confirms.
3. The system validates eligibility and swaps seat assignments.
4. New QR codes are issued to both users.

Alternate Flows:

- Customer B does not confirm within time limit → swap cancelled.
- One or both tickets already redeemed → swap denied.

Postconditions: Seats reassigned, tickets updated, both customers notified.

3.4 Classes / Objects

3.4.1 Class: Seat

3.4.1.1 Attributes

- seat_id
- row
- number
- type (standard, accessible, premium)
- status (available, held, sold, redeemed)

3.4.1.2 Functions

- hold()
- release()
- assignToOrder()
- getStatus()

References: FR2 (Seat Reservation Hold), FR3 (Checkout), FR5 (Ticket Validation), FR3.2.3 (Seat Swap).

3.4.2 Class: Ticket

3.4.2.1 Attributes

- ticket_id
- order_id
- showtime_id
- seat_id
- qr_code
- status (delivered, redeemed, refunded, exchanged)

3.4.2.2 Functions

- generateQRCode()
- deliver()
- redeem()
- revoke()

- swapSeat()

References: FR3 (Checkout and Payment), FR4 (Ticket Delivery), FR5 (Ticket Validation), FR6 (Refund/Exchange), FR3.2.3 (Seat Swap).

3.4.3 Class: LoyaltyAccount

3.4.3.1 Attributes

- account_id
- user_id
- points_balance
- tier

3.4.3.2 Functions

- earnPoints(order_total)
- redeemPoints(amount)
- adjustBalance(change, reason)

References: FR3 (Checkout), FR6 (Refund/Exchange), FR7 (Loyalty Program).

3.4.4 Class: Order

3.4.4.1 Attributes

- order_id
- user_id
- showtime_id
- total_amount
- payment_status
- created_at

3.4.4.2 Functions

- createOrder()
- processPayment()
- updateStatus()
- refund()
- exchange()

References: FR3 (Checkout), FR6 (Refund/Exchange).

3.5 Non-Functional Requirements

3.5.1 Performance

- 95% of seat map requests shall load in under 400 ms.

- 95% of QR validation scans shall complete in under 250 ms.
- 95% of checkouts shall complete in under 120 seconds.

3.5.2 Reliability

- The system shall maintain a mean time between failures (MTBF) of at least 30 days.
- Failed email deliveries shall retry up to 3 times with exponential backoff.

3.5.3 Availability

- System uptime shall be at least 99.9% per month, excluding planned maintenance.

3.5.4 Security

- All payments shall comply with PCI DSS v4.0 standards.
- All data in transit shall be encrypted using TLS 1.2 or higher.
- Customer accounts shall use secure password storage and optional multi-factor authentication.

3.5.5 Maintainability

- APIs shall be documented using OpenAPI specifications.
- System logs shall include structured error messages with correlation IDs.
- Code modules shall follow separation of concerns to ease updates.

3.5.6 Portability

- Web applications shall support current and previous major releases of Chrome, Firefox, Safari, and Edge.
- Mobile applications shall support the current and previous two major versions of iOS and Android.

3.6 Inverse Requirements

FlickFast shall not:

- Support multiple theatres or locations.
- Sell concessions (food or drinks).
- Deliver tickets via Apple Wallet, Google Wallet, or paper printing.
- Provide advanced analytics, marketing features, or detailed accessibility accommodations in the first release.

3.7 Design Constraints

- **Compliance:** The system must meet PCI DSS v4.0 requirements for secure payment handling.
- **Hardware:** Mobile devices used for ticket validation rely on built-in cameras; no external scanners will be required.

- **Performance:** The system must operate efficiently on standard consumer internet connections and theatre Wi-Fi.
- **Technology Stack:** Only approved open-source frameworks and cloud services will be used to ensure maintainability and cost control.
- **Security:** All data transmission must occur over HTTPS, and all passwords must be stored using secure hashing methods.

3.8 Logical Database Requirements

FlickFast will use a relational database to store users, showtimes, seats, orders, and ticket information.

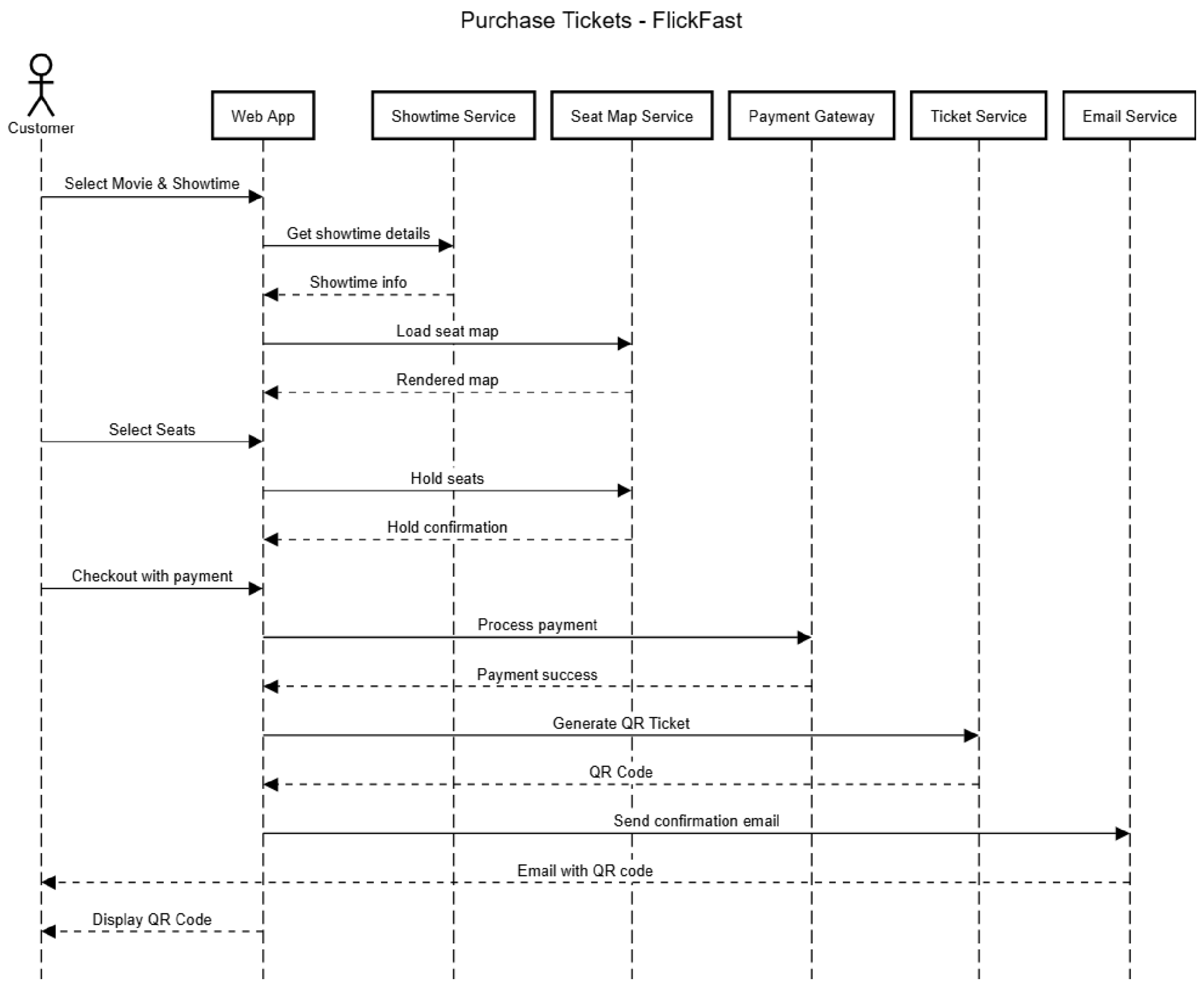
- Each order links to one or more tickets and one user account.
- Referential integrity and foreign key relationships will prevent data loss or duplication.
- Seat and ticket data must support real-time updates to avoid overselling.
- All timestamps will be stored in UTC, and monetary values will use fixed-precision decimal types.
- Backups will occur daily, and historical data older than one year will be archived.

3.9 Other Requirements

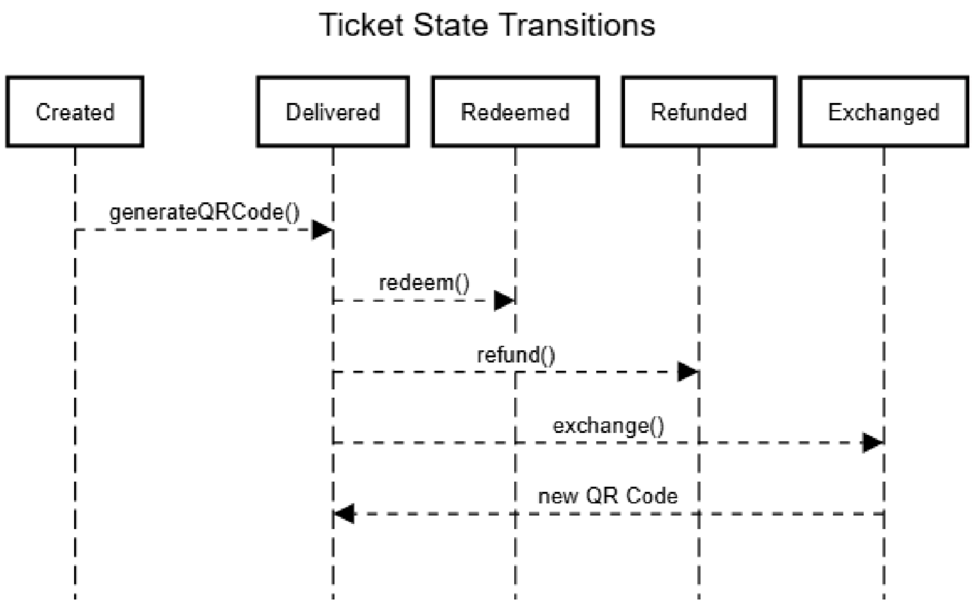
- The system must log all payment, refund, and ticket validation events for audit purposes.
- FlickFast must operate in English and use the local theatre's time zone for display.
- All errors shall be logged and reported to administrators automatically.

4. Analysis Models

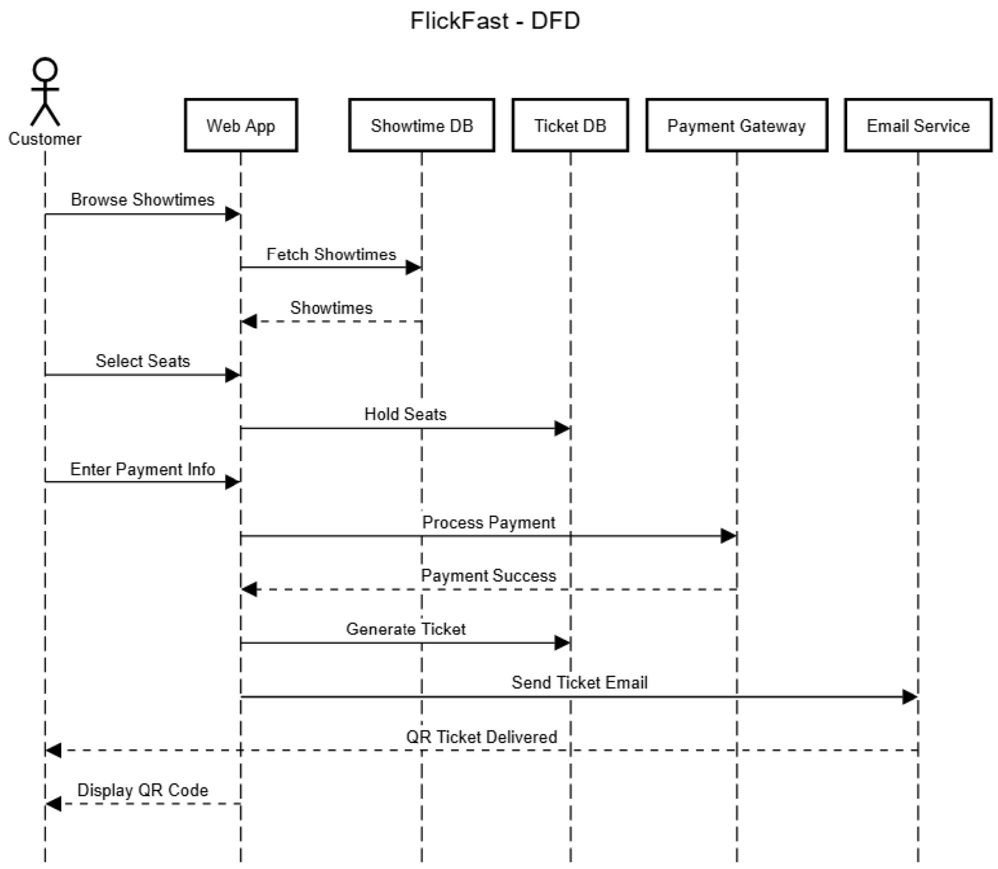
4.1 Sequence Diagram (Ticket Purchasing)



4.2 State-Transition Diagram (STD)



4.3 Data Flow Diagram (DFD)



5. Change Management Process

5.1 Change Request Initiation

- **Who may submit:** Any team member (developer, tester, manager) or the client instructor.
- **Submission method:** Formal *Change Request Form* submitted via project repository issue tracker or internal form.
- **Contents:** Description, rationale, affected sections, proposed priority, and estimated impact.

5.2 Review and Impact Analysis

- Change requests are logged in the Change Register.
- The Lead Software Engineer and Project Manager perform impact analysis on scope, cost, and timeline.
- Each change is categorized as Minor, Major, or Critical.

5.3 Approval and Implementation

- Approved changes are entered into the SRS revision history table.
- The responsible developer updates affected sections and commits revisions to the version-controlled repository (e.g., GitHub).
- All changes must undergo peer review and regression testing.

5.4 Summary

The change management process ensures that all modifications to the FlickFast SRS are systematic, documented, and approved prior to implementation, preserving alignment between the software product and stakeholder expectations.

6. Software Design Specification

6.1 System Description

The FlickFast Ticketing System is designed as a fast, lightweight, single-theatre ticketing solution accessible through web and mobile applications. It supports customers in browsing movies, selecting seats, completing purchases, and accessing digital QR code tickets. Staff members use dedicated tools to validate entries and manage showtimes.

The overall design emphasizes speed, security, and simplicity. Each major component is modular and communicates via a well-defined RESTful API. The frontend (customer and staff interfaces) handles presentation and user interaction, while the backend manages business logic, database operations, and third-party integrations such as Stripe for payment processing and an email service for ticket delivery.

6.2 Software Architecture Overview

The FlickFast system follows a **three-tier architecture**:

1. **Presentation Layer (Client Applications)**

    - **Customer Web App:** Allows users to browse movies, select seats, and complete purchases.
    - **Customer Mobile App:** Provides the same functionality as the web app, with offline QR storage for validation.
    - **Staff Validator App:** Used by ushers to scan QR codes and check ticket validity.
    - **Manager Console:** Administrative web interface for managing showtimes, pricing, and refunds.

2. **Application Layer (Backend Server / API)**

    - Implements all business logic, request validation, and session handling.
    - Exposes REST and WebSocket endpoints to the frontend.
    - Handles secure communication with third-party APIs such as Stripe and SMTP.

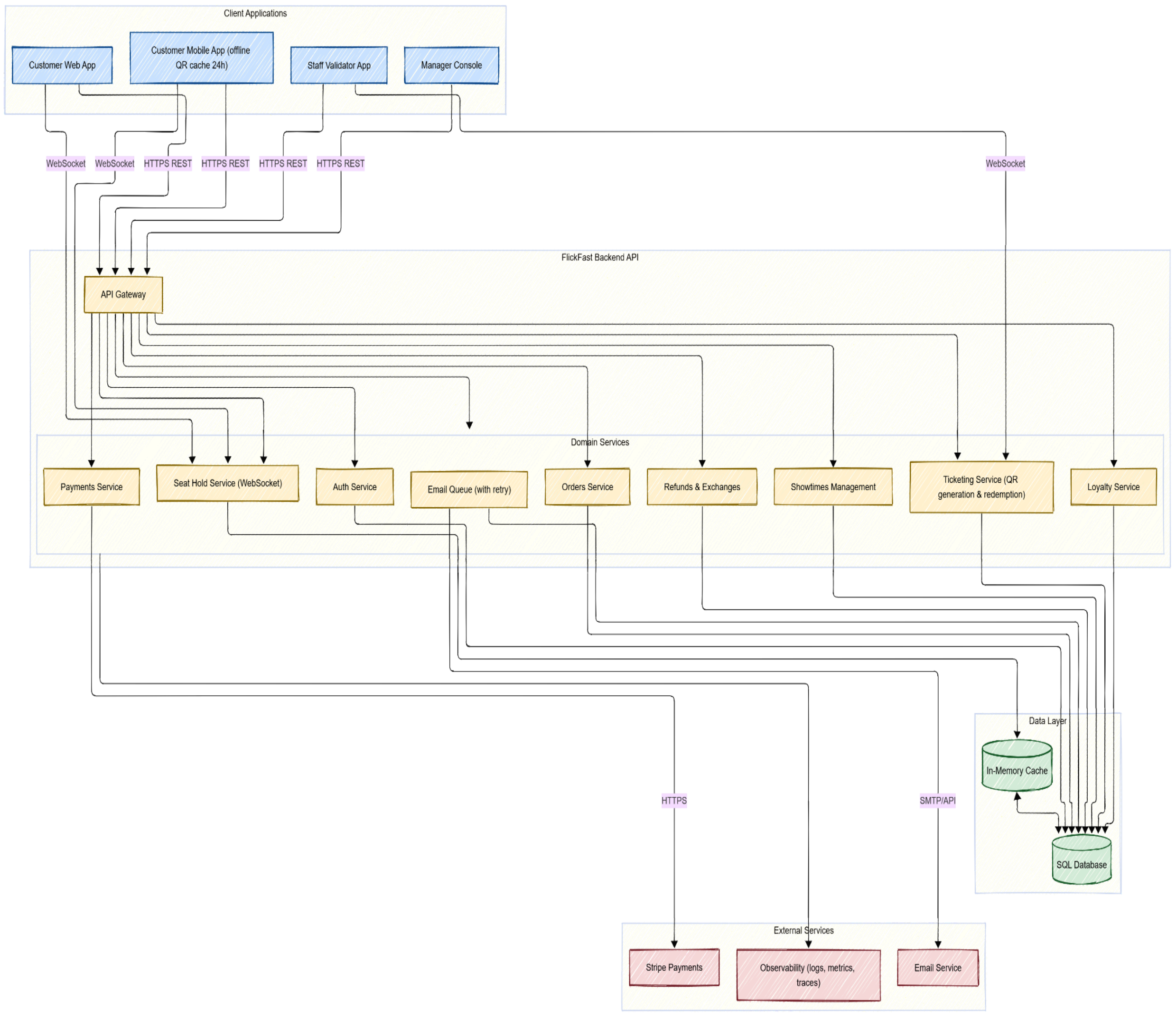
3. **Data Layer (Database and External Services)**

    - Relational database for users, orders, tickets, seats, and showtimes.
    - External services for payment (Stripe) and email delivery.
    - Data persistence ensures referential integrity and prevents seat overselling.

6.2.1 Software Architecture Diagram

The FlickFast architecture connects client apps to a central backend that handles data, logic, and the external integrations.

Figure 1: FlickFast Software Architecture Overview



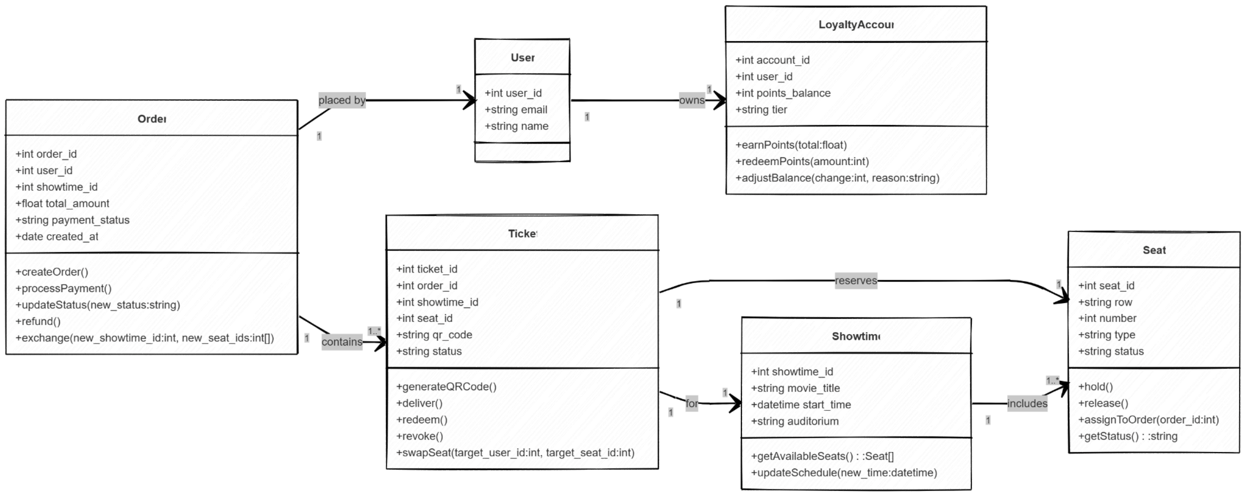
Architecture Explanation:

- All client applications connect securely via HTTPS to the FlickFast backend.
- The backend manages sessions, processes payments via Stripe, and updates the SQL database.
- Email services are used to send QR codes and confirmations.
- The WebSocket service keeps seat maps synchronized in real-time between customers.

6.3 UML Class Diagram

The FlickFast UML diagram shows how Users place Orders for movie Showtimes, which include reserved Seats and generated Tickets, while each User also owns a LoyaltyAccount that tracks and redeems rewards.

Figure 2: FlickFast UML Class Diagram



6.3.1 UML Class Descriptions

- **Seat:** Represents a specific location in an auditorium. Maintains state (available, held, sold, redeemed) and handles seat holds and assignments.
- **Ticket:** Links a seat and order. Manages QR generation, delivery, redemption, and swapping.
- **Order:** Represents a customer purchase. Handles creation, payment, refund, and exchange processes.
- **LoyaltyAccount:** Tracks points earned and redeemed by customers. Supports balance adjustment and tier progression.
- **Showtime:** Stores movie schedule information and provides access to available seats.

Each class connection uses unique IDs to keep data linked correctly in the database.

6.4 Development Plan and Timeline

6.4.1 Task Partitioning

Task	Description	Responsible Member
UI Design	Develop mockups and implement responsive layouts for web and mobile.	Duffy Adams
Backend API	Implement endpoints for checkout, seat holds, and ticket delivery.	Soham Kulkarni
Database Schema	Design and configure relational databases with constraints.	Brandon Helm

Payment Integration	Integrate Stripe API for secure payments and refunds.	Soham Kulkarni
QR Validation System	Implement ticket scanning and validation logic.	Brandon Helm
Testing & QA	Unit and integration testing across modules.	All Members
Deployment Setup	Configure hosting, environment variables, and monitoring.	Duffy Adams

6.4.2 Development Timeline

Week	Milestone	Deliverables
Week 1	Project setup and repository creation	GitHub repo, base project structure
Week 2	Database and backend API scaffolding	Schema, REST endpoints
Week 3	Web and mobile UI prototypes	Functional mockups
Week 4	Payment gateway integration	Stripe test environment
Week 5	QR validation and loyalty module	Scanning system, points tracking
Week 6	Testing and bug fixes	QA report
Week 7	Final integration and presentation	Working demo, documentation

6.5 Summary

This document explains how the FlickFast Ticketing System is built, how its parts work together, and how it will be developed. The system is designed to be easy to update, run smoothly, and handle growth. It also follows payment security rules and aims to give customers and theater staff a fast, simple ticketing experience.

7. Test Plan

7.1 Objectives

The goal of this test plan is to confirm that FlickFast works as expected. This includes checking:

- **Functional features**: ticket purchase, seat holds, seat swaps, ticket validation, refunds/exchanges, and loyalty points.
- **Performance targets**:
    - Seat map loads in ≤400 ms (95th percentile)
    - Ticket validator response in ≤250 ms (95th percentile)
    - Checkout process finishes in ≤120 seconds

Tests will be done at different levels:

- **Unit testing** – small pieces of code
- **Integration testing** – how components work together
- **System testing** – the full system under realistic conditions

All tests are traceable to requirements in **SRS Sections 3–6** and diagrams in **Section 4** (Sequence Diagrams, State Transitions, and Data Flow).

7.2 In Scope

We will test:

Customer flows:

- Browsing movies
- Selecting seats
- Completing checkout
- Receiving and using QR code tickets

Staff flows:

- Scanning tickets at the door
- Handling refunds, exchanges, and seat swaps
- Using manager overrides when needed

Loyalty program logic:

- Earning and redeeming points

Performance targets from the non-functional requirements in **SRS §3.5**

7.3 Out of Scope

We will *not* test the following features, as they are outside the initial release scope:

- Support for multiple theatres
- Selling concessions (food, drinks)
- Wallet passes (e.g., Apple/Google Wallet)

These exclusions align with the constraints and inverse requirements listed in the SRS.

7.4 Test Items (Features and Components)

Feature / Function	SRS Reference	Component(s) Tested
Seat Holds	§3.2.2	`Seat` class (`hold()`, `release()`)
Checkout & Payment	§3.2.1	`Order`, `Ticket`, Stripe sandbox
Ticket Delivery & Validation	§3.2.1, §3.2.2	`Ticket.redeem()`
Refunds and Exchanges	§3.3.3	`Order.refund()`, `Order.exchange()`
Seat Swap	§3.2.3	`Ticket.swapSeat()`
Loyalty Points	§3.2.1, §3.3.3	`LoyaltyAccount` (earn/redeem)

7.5 Test Strategy and Levels

7.5.1 Unit Testing

Goal: Test individual functions in isolation with mock data.

Tools: Use your language's standard testing framework (e.g., `pytest`, `JUnit`). Mocks/stubs will replace:

- External APIs (Stripe, email)
- Databases
- WebSockets

Key Checks:

- Seat hold and release logic
- Prevent duplicate ticket redemptions
- Correct calculation of loyalty points

Example Unit Tests:

- `Seat.hold()` and `Seat.release()` — Confirm 5-minute timeout, correct status changes, and prevent double booking.
- `Ticket.redeem()` — Ticket can only be used once. Further attempts should raise an error or be ignored.
- `LoyaltyAccount.redeemPoints()` — Prevent negative balances. Ensure adjustments on refunds/exchanges.

7.5.2 Functional (Integration) Testing

Goal: Make sure major features work end-to-end, including connections between modules and external services.

Setup:

- Stripe test keys
- Fake email server
- Test database with preset data
- Simulated real-time updates (e.g., WebSockets)

Example Functional Tests:

- **Happy Path Checkout** — Select seats → hold → pay → generate QR → send confirmation email
- **Failed Payment** — Payment fails → seats are released → UI reflects correct state
- **Seat Swap** — Two users swap seats within 2 minutes → both receive new QR codes → audit entry is logged

---

7.5.3 System Testing

Goal: Test the full system under realistic, high-load conditions.

Tools:

- Load testing tools: `k6`, `JMeter`
- Synthetic QR scanner rig
- Monitoring via application performance tools (APM/logs)

Example System Tests:

- **End-to-End SLA** — From browse to QR scan at entry ≤120 seconds; validator p95 ≤250 ms
- **Validator Throughput** — Handle ~8 scans/second for 60 seconds, including 10% duplicate tickets; p95 latency ≤250 ms
- **Seat Map Performance** — With 200 virtual users, seat map loads with p95 latency ≤400 ms and <0.1% error rate

---

7.6 Test Environments

Environment	Description
Test	Isolated database schema, Stripe test keys, fake SMTP server, feature flags enabled
Staging	Near-production data, real seat maps, theatre timezone configuration, limited real email delivery to test inboxes

---

7.7 Test Data

Seed the database with:

- One theatre
- Multiple movies and showtimes
- Auditorium layout with standard, accessible, and premium seats
- Users with various loyalty point balances
- Orders in different states (delivered, redeemed, refunded)

7.8 Entry & Exit Criteria

Entry Criteria:

- Features merged into the main branch
- Unit test coverage ≥85% on domain modules
- No open P1 (critical) bugs

Exit Criteria:

- All P1 and P2 bugs are resolved or waived
- System performance targets are met
- Full regression suite passes

7.9 Defect Management

- All defects will be tracked in **GitHub Issues**
- Severity levels: P1 (Critical) to P4 (Low)
- Each issue must:
  - Be linked to failed test(s)
  - Reference the related SRS requirement

7.10 Traceability (sample)

Requirement	Artifact	Test IDs
FR: Seat Reservation Hold (§3.2.2)	Seat.hold(), Seat Map	U-001, F-003, S-004
FR: Validate Entry (Use Case #2)	Validator App, Ticket.redeem()	U-002, S-001, S-002
FR: Seat Swap (§3.2.3)	Ticket.swapSeat(), Audit Log	F-002
NFR: Performance (p95 targets)	Seat Map, Validator	S-001, S-002, S-004

7.11 Risks & Mitigations

- Third-party outages → sandbox retries/fallbacks; inject fault scenarios.
- Race conditions during holds → simulate concurrent purchases in CI.
- Email deliverability → assert on outbound queue + test inbox.

7.12 Deliverables

- **Test Plan (this section)**
  Excel test cases (10) - see link below
  Test scripts (unit/integration), load test scripts, CI results summary
- **GitHub link** to the /testing folder and this updated SRS (see §7.13)

7.13 GitHub Link

https://github.com/DuffyAdams/FlickFast-Movie-Ticketing-System.git

A. Appendices

Appendices may be used to provide additional (and hopefully helpful) information. If present, the SRS should explicitly state whether the information contained within an appendix is to be considered as a part of the SRS's overall set of requirements.

Example Appendices could include (initial) conceptual documents for the software project, marketing materials, minutes of meetings with the customer(s), etc.

A.1 Appendix 1

A.2 Appendix 2