<CineGet>

Software Design Specification

<Version 0.1>

<10/09/2025>

<Group 12>

<Mateo Boccalato >
<Rayaan Mohammed>

Prepared for
CS 250- Introduction to Software Systems
Instructor: Gus Hanna, Ph.D.

Revision History

Date	Description	Author	Comments
<9/25/25>	<Version 0.1>	<Mateo Boccalato>	<First Revision>
<9/25/25>	<Version 0.1>	<Rayaan <Mohammed>	<First Revision>

Document Approval

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

Signature	Printed Name	Title	Date
	<Mateo Boccalato>	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	
	<Mohammed Rayaan>	Software Eng.	

Table of Contents

REVISION HISTORY
..... II
DOCUMENT APPROVAL
..... II
1. INTRODUCTION..... 1
1.1 PURPOSE..... ..... 1 1.2 SCOPE .... 1 1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS ..... 1 1.4 REFERENCES 2
1.5 OVERVIEW 2
2. GENERAL DESCRIPTION
..... 2

2.1 PRODUCT PERSPECTIVE ..... 2 2.2 PRODUCT FUNCTIONS ..... 2 2.3 USER CHARACTERISTICS ..... 3 2.4 GENERAL CONSTRAINTS..... 3
2.5 ASSUMPTIONS AND DEPENDENCIES 3

3. SPECIFIC REQUIREMENTS..... 3

3.1 EXTERNAL INTERFACE REQUIREMENTS 4
3.1.1 User Interfaces *3*
3.1.2 Hardware Interfaces..... *3*
3.1.3 Software Interfaces *3*
3.1.4 Communications Interfaces *3*
3.2 FUNCTIONAL REQUIREMENTS..... 5
3.2.1 <Functional Requirement or Feature #1> *3*
3.2.2 <Functional Requirement or Feature #2> *3*
3.3 USE CASES 6
3.3.1 Use Case #1..... *3*
3.3.2 Use Case #2..... *3*
3.4 CLASSES / OBJECTS 6
3.4.1 <Class / Object #1> *6*
3.4.2 <Class / Object #2> *6*
3.5 NON-FUNCTIONAL REQUIREMENTS 7
3.5.1 Performance..... *7*
3.5.2 Reliability *7*
3.5.3 Availability *7*
3.5.4 Security *7*

3.5.5 Maintainability *7*
3.5.6 Portability *7*
3.6 INVERSE REQUIREMENTS ..... 7 3.7 DESIGN CONSTRAINTS ..... 7 3.8 LOGICAL DATABASE REQUIREMENTS ..... 8 3.9 OTHER REQUIREMENTS 8
4. ANALYSIS MODELS 8
4.1 SEQUENCE DIAGRAMS ..... 5 4.3 DATA FLOW DIAGRAMS (DFD) ..... 8 4.2 STATE-TRANSITION DIAGRAMS (STD) 8
5. CHANGE MANAGEMENT PROCESS 8
A. APPENDICES 8
A.1 APPENDIX 1 ..... 8 A.2 APPENDIX 2 8

1. Introduction

The introduction to the Software Requirement Specification (SRS) document should provide an overview of the complete SRS document. While writing this document please remember that this document should contain all of the information needed by a software engineer to adequately design and implement the software product described by the requirements listed in this document. (Note: the following subsection annotates are largely taken from the IEEE Guide to SRS).

1.1 Purpose

The purpose of this SRS document is to specify the requirements for the **Theater Ticketing System**, an online platform to browse movies, purchase tickets, and manage theater operations. The document provides a detailed description of the system's functionality, external interfaces, constraints, and requirements as elicited from client interviews and stakeholder input. It is intended for stakeholders, developers, testers, and instructors overseeing the project.

1.2 Scope

The Theater Ticketing System is a web-based application that enables customers to purchase tickets with assigned seating. Each showtime will have an auditorium layout displayed, allowing customers to choose their preferred seats. Customers will be able to purchase tickets up to two weeks before a showtime and until ten minutes after the movie begins, with a maximum of twenty tickets per transaction. The system will apply discounts for students, military personnel, and seniors, and it will also provide options for customer feedback at the end of each purchase. Administrators will have access to a management mode that allows them to override transactions, manage showtimes, and handle operational issues. The system will also integrate critic reviews and ratings from online platforms. Out of scope for this version are kiosk integrations, refund processing online, and multilingual support.

1.3 Definitions, Acronyms, and Abbreviations

Showtimes: The scheduled screening time for a given movie.

- **Assigned Ticket**: A ticket linked to a specific seat in an auditorium.
- **Hold**: A temporary five-minute reservation on seats while a customer completes a purchase.
- **Admin Mode**: Special interface for theater managers to override issues and manage showtimes.
- **UI**: User Interface.

QR: Quick response code used for ticket identification.

1.4 References

This subsection should:

(1) Provide a complete list of all documents referenced elsewhere in the SRS, or in a separate, specified document.
(2) Identify each document by title, report number - if applicable - date, and publishing organization.
(3) Specify the sources from which the references can be obtained.
This information may be provided by reference to an appendix or to another document.

- PCI DSS (referenced for best practices; full compliance not required for class mock payments).
- IEEE 830-1998 Recommended Practice for Software Requirements Specifications
- How to Write a Software Requirements Specification (SRS) Document (Gerhard Krüger)

1.5 Overview

The following sections describe the system in detail. Section 2 provides the overall description, including product perspective, user characteristics, and constraints. Section 3 outlines specific requirements such as functional, non-functional, and interface requirements. Section 4 provides supporting information including use cases, appendices, and traceability.

2. General Description

The proposed Theater Ticketing System will resolve the limitations seen by the current system, by supporting at least one thousand concurrent users, providing real-time updates across online and in-person sales, and offering an intuitive and user-friendly interface. The system will also provide scalability across twenty theaters in San Diego, operating within the Pacific Time Zone.

The potential clients identified during client interviews expressed a need for consistent assigned seating across all theaters, effective seat management, and the inclusion of critic reviews to improve customer decision-making. They also required discount mechanisms, feedback options, and a robust administrative portal. The system will run exclusively in web browsers, ensuring accessibility from both desktop and mobile devices without requiring separate applications.

2.1 Product Perspective

CineSnap is a standalone web/mobile app (client + backend + small admin portal). It integrates with a mock payment gateway and sends email receipts/QR tickets. No third-party seat maps are assumed.

2.2 Product Functions

At a high level, the system enables customers to browse available movies, view showtimes, select specific seats in auditoriums, apply discounts, and complete purchases using a secure checkout process. After successful payment, customers receive a confirmation email with QR codes for entry. Administrators use the system to create and edit showtimes, manage auditoriums, enforce seating availability, and resolve ticketing conflicts. Staff at theater entrances validate tickets

using QR scanners, ensuring that each ticket can only be used once. The system also provides sales summaries, critic review displays, and customer feedback collection.

2.3 User Characteristics

The system is designed for three primary user groups. Customers are members of the general public with varying levels of computer literacy, but all are expected to be familiar with using a web browser. Staff members are theater employees responsible for validating tickets quickly and efficiently; their interface must be simple and optimized for speed. Administrators are managers with responsibilities for scheduling shows, resolving conflicts, and generating reports; they require a more feature-rich interface with greater control over system functions. All user groups require minimal training, and the system should be intuitive enough to support first-time users.

2.4 General Constraints

The system must support at least one thousand concurrent users to handle peak demand during popular movie releases. It must be delivered exclusively through web browsers and cannot rely on mobile app installations. The database must be centralized but partitioned logically to support twenty theaters in the San Diego area, all operating in the Pacific Time Zone. Purchases are limited to twenty tickets per transaction, and the sales window is restricted to two weeks before and ten minutes after a showtime. Refunds are not handled online and must be managed by staff in person. These constraints are derived directly from client interviews and operational requirements.

2.5 Assumptions and Dependencies

3. Specific Requirements

The system assumes reliable internet connectivity for both customers and staff devices. It depends on third-party services for payment authorization, email delivery, and critic review integration. It further assumes that seat layouts for auditoriums are known and fixed before the showtime configuration. Time synchronization across servers is assumed to be consistent enough to manage five five-minute seat holds with a skew of less than five seconds -minute seat holds with less s tha relies on the availability of APIs from providers such as Rotten Tomatoes or IMDb n a five-second skew. The success of critic review integration assumes the availability of APIs from providers such as Rotten Tomatoes or IMDB. If these assumptions or dependencies change, requirements for the system will need to be updated accordingly.

Attention should be paid to carefully organizing the requirements presented in this section so that they may easily accessed and understood. Furthermore, this SRS is not the software design document, therefore one should avoid the tendency to over-constrain (and therefore design) the software project within this SRS.

3.1 External Interface Requirements

3.1.1 Browse and Search Movies

The system shall allow customers to view all available movies and showtimes. Customers shall be able to search for movies by title, date, or theater location and filter results by showtime and rating.

3.1.2 Select and Hold Seats

The system shall display a graphical seat map for each auditorium. Customers shall be able to select one or more seats, which the system will hold for five minutes while the purchase is completed. When the hold expires, the seats shall be released back to inventory.

3.1.3 Purchase Tickets

The system shall allow customers to purchase tickets online through a secure checkout process. Customers shall be able to buy up to twenty tickets per transaction. The purchase window shall open two weeks before the showtime and remain open until ten minutes after the scheduled start.

3.1.4 Discounts

The system shall provide discounted ticket pricing for students, seniors, and military personnel. Customers selecting a discount shall be required to confirm their eligibility.

3.1.5 Payment Processing

The system shall integrate with an external payment gateway to process transactions securely. If payment authorization succeeds, the system shall confirm the purchase and commit the selected seats. If authorization fails, the system shall allow retry attempts while the hold remains.

3.1.6 Ticket Confirmation and QR Codes

The system shall generate a unique QR code for each purchased seat. QR codes shall be displayed on the confirmation page and emailed to the customer. Each QR code shall be scannable for validation at entry.

3.1.7 Validate Tickets

The system shall allow staff to scan QR codes using mobile devices. Tickets shall be marked as valid if unused and for the correct showtime, or invalid if previously used, expired, or for another showtime.

3.1.8 Administrative Management

The system shall provide administrators with tools to create and manage showtimes, adjust auditorium configurations, override transactions, and close or reopen sales. Administrative actions shall be logged for traceability.

3.1.9 Reviews and Feedback

The system shall integrate critic ratings and review excerpts from approved sources and shall request post-purchase customer feedback through a rating interface. Customer feedback shall be stored and accessible to administrators.

3.2 Functional Requirements

3.2.1 Graphical User Interface

3.2.1.1 The system shall provide a uniform look and feel across all pages with consistent navigation and controls.

3.2.1.2 The system shall render correctly and be usable on both desktop and mobile browsers.

3.2.1.3 The system shall display the seat map clearly, showing available, held, and sold seats with a visible countdown timer for holds.

3.2.1.4 The system shall present transparent pricing, including discounts and fees, before finalizing payment.

3.2.1.5 The system shall provide contextual help and display error messages close to the relevant input fields.

3.2.2 Accessibility

3.2.2.1 The system shall comply with WCAG 2.1 AA standards for accessibility.

3.2.2.2 The system shall support keyboard-only navigation for all key features including the seat map.

3.2.2.3 The system shall allow screen readers to announce seat states, error messages, and other important information.

3.2.2.4 The system shall not rely on color alone to convey status and shall maintain proper contrast ratios.

3.2.2.5 The system shall ensure touch targets are large enough for mobile devices, and forms shall include proper labels and associations.

3.2.3 Feedback and Error Handling

3.2.3.1 The system shall prompt customers for feedback after purchases using a simple interface such as rating icons.

3.2.3.2 The system shall handle errors such as declined payments, expired holds, or unavailable seats with clear and actionable messages.

3.2.3.3 The system shall display loading indicators during long operations and preserve progress during a page refresh within the active hold window.

3.2.4 Usability Benchmark

3.2.4.1 A first-time customer shall be able to complete browsing, seat selection, and payment in five steps or fewer under normal conditions.

3.2.4.2 Staff members shall be able to validate tickets quickly using minimal training.

3.2.4.3 Administrators shall be able to manage showtimes and overrides with minimal instruction, ensuring overall usability across all user groups.

3.3 Use Cases

3.3.1 Use Case 1: Purchase Tickets

Actors: Customer, Payment Gateway, Email Service.

Flow: Customer browses listings, selects seats, enters details, submits payment. System authorizes payment, commits seats, and emails QR codes. Exceptions: Payment decline, seat conflict, expired hold.

3.3.2 Use Case 2: Validate Tickets

Actors: Staff, Customer.

Flow: Staff scans QR code, system checks validity, updates ticket status, and grants/denies entry. Exceptions: Already used, wrong showtime, invalid code.

3.3.3 Use Case 3: Manage Showtimes

Actors: Administrator.

Flow: Admin logs in, enters showtime details, system validates and saves. Exceptions: Auditorium conflicts or invalid input.

3.4 Classes / Objects

3.4.1 Tickets <Class / Object #1>

3.4.1.1 Attributes: Ticket ID, Seat Number, Showtime ID, Price, Discount Type, Status.
3.4.1.2 Functions: Generate QR code, Update status, Link to order.
Referenced in Functional Requirements 3.2.2, 3.2.3, and Use Case 1.

3.4.2 Showtime<Class / Object #2>

3.4.2.1 Attributes: Showtime ID, Movie Title, Date/Time, Auditorium, Seat Layout.
3.4.2.2 Functions: Display listings, Validate auditorium availability, Link to tickets.
Referenced in Functional Requirements 3.2.1, 3.2.5, and Use Case 3.

3.4.3 User <Class / Object #3>

3.4.3.1 Attributes: User ID, Role, Name, Email.
3.4.3.2 Functions: Browse, Purchase, Validate, Manage depending on role. Referenced in multiple Functional Requirements and Use Cases.

3.4.4 Screen Room <Class / Object #4>

3.4.4.1 Attributes: Room ID, Theater Location, Seat Layout (rows, columns, accessibility seats), Capacity.
3.4.4.2 Functions: Provide seat map to showtime, Track unavailable seats, Return list of available seats, Validate seat identifiers.
Referenced in Functional Requirements 3.2.2, 3.2.3, and Use Case 1.

3.5 Non-Functional Requirements

Non-functional requirements may exist for the following attributes. Often these requirements must be achieved at a system-wide level rather than at a unit level. State the requirements in the following sections in measurable terms (e.g., 95% of transaction shall be processed in less than a second, system downtime may not exceed 1 minute per day, > 30 day MTBF value, etc).

3.5.1 Performance

The system shall support at least one thousand concurrent users. 95% of seat-map loads shall complete within two seconds.

3.5.2 Reliability

Seat holds shall expire automatically within five minutes, ensuring no overselling.

3.5.3 Availability

The system shall maintain 99% uptime during theater hours.

3.5.4 Security

All communication shall occur over HTTPS; no card data shall be stored.

3.5.5 Maintainability

Code shall be version-controlled, with configurable parameters accessible without code changes.

3.5.6 Portability

The system shall run on current browsers across Windows, macOS, iOS, and Android.

3.6 Inverse Requirements

The system shall not provide kiosk hardware support in version 1. It shall not process online refunds. It shall not store raw payment card data.

3.7 Design Constraints

3.7.1The system must be browser-only and accessible via modern browsers.
3.7.2 It must comply with WCAG 2.1 AA standards.
3.7.3 The centralized database shall support partitioning by theater.
3.7.4 The system must be able to handle atleast 1000 ticket sales at once.

3.8 Logical Database Requirements

Entities: Movie, Showtime, Auditorium, Seat, Ticket, Order, User. Relationships: Showtime–Seat (one-to-many), Order–Ticket (one-to-many), User–Order (one-to-many). Seat state transitions: available → held → sold → used.

3.9 Other Requirements

The system shall log administrative overrides for auditing. It shall request customer feedback after purchases. It shall integrate with review APIs for critic ratings.

4. Analysis Models

To be completed in later phases. Includes sequence diagrams for ticket purchase, validation, and showtime management; state-transition diagrams for ticket lifecycle; and data flow diagrams for core processes.

4.3 Data Flow Diagrams (DFD)

4.2 State-Transition Diagrams (STD)

5. Change Management Process

Identify and describe the process that will be used to update the SRS, as needed, when project scope or requirements change. Who can submit changes and by what means, and how will these changes be approved.

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

6. Software Design Specification

6.1 Software Architecture Overview

CineGet is implemented using a three-tier architecture composed of the Presentation, Application, and Data layers. Each layer is logically separated to simplify maintenance and allow independent scaling.

6.1.1 Presentation Layer

This layer provides all user-facing features. It includes the responsive web interface used by customers, administrators, and theater staff. HTML5, CSS, and JavaScript frameworks are used to ensure consistent layout across devices. All user actions in this layer are converted into HTTP requests that are passed to the backend API.

6.1.2 Application Layer

This layer contains the core business logic that enforces CineGet's rules. It validates user inputs, manages seat availability, handles payment requests, and produces QR-coded tickets. The backend is structured into modules—authentication, movie management, showtime scheduling, ticket processing, and feedback collection—each accessible through REST endpoints secured by HTTPS and JSON Web Tokens (JWTs).

6.1.3 Data Layer

The data layer consists of a centralized SQL database. It stores persistent entities such as users, movies, auditoriums, seats, tickets, and orders. Referential integrity is enforced through primary and foreign keys. Indexing on frequently queried fields (movieID, showtimeID, seatID) improves performance during high traffic periods.

6.1.4 Scalability and Communication

All layers communicate through RESTful APIs using JSON payloads. Load balancing can be applied to the Application Layer, while replication ensures database reliability.

6.2 UML Class Diagram

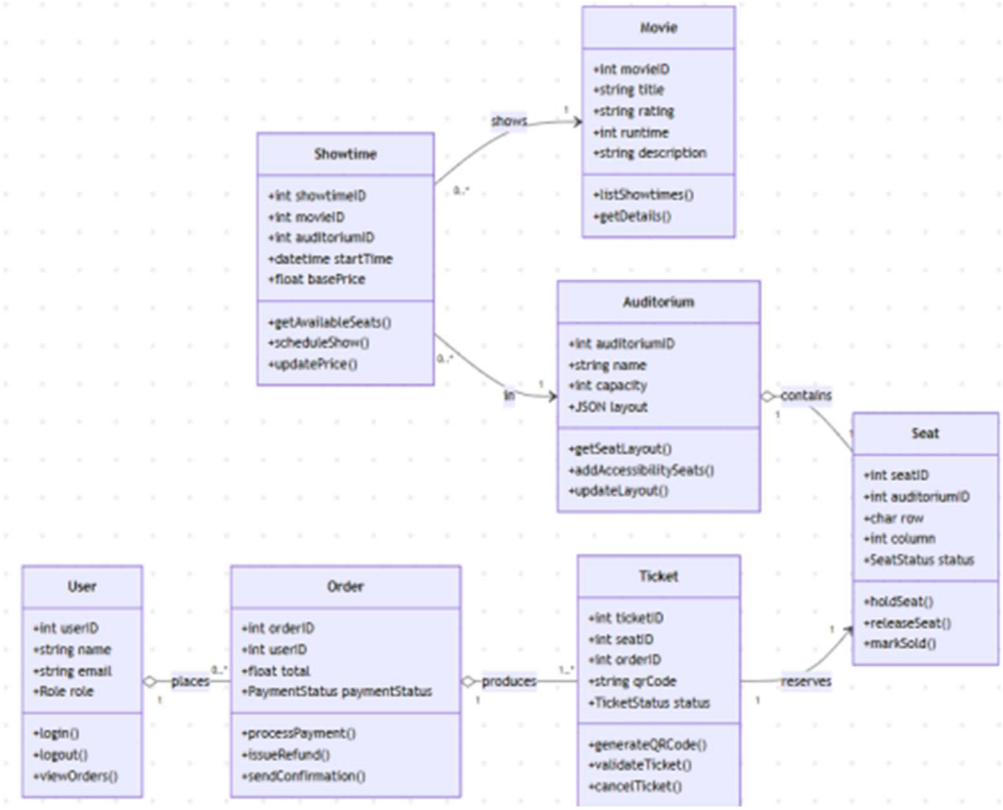
The UML Class Diagram illustrates the object-oriented structure of CineGet. It identifies major classes, their attributes, operations, and associations. The diagram highlights how data flows between movies, showtimes, and tickets.

6.2.1 Relationships

> *6.2.1.1 User ↔ Order ↔ Ticket*: A user can have many orders; each order produces one or more tickets.

6.2.1.2 Movie ↔ Showtime ↔ Seat: A movie can have multiple showtimes; each showtime manages a unique set of seats.

6.2.1.3 Auditorium ↔ Seat: Each auditorium defines its seat layout and accessibility options.



6.2.2 Design Patterns

CineGet employs a Model-View-Controller (MVC) pattern at the application level, separating presentation logic from data manipulation.

Class	Attributes	Operations
User	userID:int, name:string, email:string, role:enum	login(), logout(), viewOrders()
Movie	movieID:int, title:string, rating:string, runtime:int, description:string	listShowtimes(), getDetails()
Showtime	showtimeID:int, movieID:int, auditoriumID:int, startTime:datetime, basePrice:float	getAvailableSeats(), scheduleShow(), updatePrice()

Class	Attributes	Operations
Seat	seatID:int, auditoriumID:int, row:char, column:int, status:enum	holdSeat(), releaseSeat(), markSold()
Ticket	ticketID:int, seatID:int, orderID:int, qrCode:string, status:enum	generateQRCode(), validateTicket(), cancelTicket()
Order	orderID:int, userID:int, total:float, paymentStatus:enum	processPayment(), issueRefund(), sendConfirmation()
Auditorium	auditoriumID:int, name:string, capacity:int, layout:JSON	getSeatLayout(), addAccessibilitySeats(), updateLayout()

6.3 Software Architecture (SWA) Description

CineGet is divided into micro-modules that communicate via secure REST calls. Each service performs a specific responsibility to promote reusability.

6.3.1 Customer Service – manages browsing, seat selection, and purchases.

6.3.2 Staff Service – validates QR tickets and records entry time.

6.3.3 Admin Service – handles showtime creation, pricing, and report generation.

6.3.4 Database Service – provides centralized access to persistent data with transaction rollback on errors.

6.3.5 Notification Service – sends emails with receipts and QR codes.

6.3.6 Integration Service – connects with external APIs such as payment gateways and critic-review sources.

6.3.7 Security - All endpoints require authentication tokens. Sensitive data such as passwords and payment tokens are hashed or stored through third-party providers that comply with PCI standards. Audit logs capture administrative overrides and login events to support accountability.

6.3.8 Error Handling - Every REST response follows a standard format containing a status code, message, and data object. This design enables predictable client behavior and reduces ambiguity.

6.4 Development Plan and Timeline

The development plan organizes CineGet into incremental deliverables so progress can be tracked easily. Agile iterations allow for continuous feedback from stakeholders.

Task	Assigned Member	Description	Deadline
Frontend UI & Navigation	Rayaan	Design responsive pages for customer, staff, and admin	Week 3
Backend API & Logic	Mateo	Implement REST endpoints and seat-hold/payment logic	Week 4
Database Design	Rayaan	Create relational schema, constraints, and indexes	Week 4
Ticket Validation Module	Mateo	Build QR scan and validation workflow	Week 5
Testing & Debugging	Both	Write unit, integration, and acceptance tests	Week 6
Deployment & Docs	Both	Publish final system to GitHub and prepare documentation	Week 7

6.4.1 Milestones

6.4.1.1 Weeks 1–2: Confirm requirements and finalize architectural plan.
6.4.1.2 Weeks 3–4: Complete core UI and backend integration.
6.4.1.3 Weeks 5–6: Add advanced features, test, and refine.
6.4.1.4 Week 7: Final deployment, demonstration, and submission.

6.4.2 Tools and Environment

6.4.2.1 Version Control: GitHub private repository.
6.4.2.2 Database: MySQL or PostgreSQL.
6.4.2.3 Languages: JavaScript (frontend and Node.js backend).
6.4.2.4 Frameworks: Express.js, React, Bootstrap.
6.4.2.5 IDE: VS Code.

6.5 Summary

This Software Design Specification provides the technical roadmap for CineGet. The document translates user requirements into implementable structures and clearly defines the interaction between components.
By adhering to the modular architecture and secure development principles described above, the CineGet team will deliver a scalable, maintainable, and accessible web application that meets all functional and non-functional requirements set forth in the original SRS.

7. Test Plan

7.1 Purpose

The purpose of this Test Plan is to define the testing scope, approach, and objectives for verifying and validating the CineGet Theater Ticketing System. Testing ensures that all requirements identified in the Software Requirements and Design Specification are implemented correctly and that CineGet performs reliably, securely, and efficiently under expected operating conditions.

7.2 Items to Be Tested

Testing covers the following components and features:

- Domain Models and Classes: User, Movie, Showtime, Auditorium, Seat, Order, Ticket.
- Core Features: movie browse and search, seat map and five-minute hold/release, purchase limit (≤ 20 tickets), discount mechanisms, payment authorization, QR ticket generation and validation, administrative management of showtimes, critic review integration, and customer feedback.

- Non-Functional Features: performance (≤ 2 seconds for 95 percent of seat-map loads), reliability (automatic hold expiry), availability (≥ 99 percent during theater hours), security (HTTPS, no card storage), and accessibility (WCAG 2.1 AA).

7.3 Out of Scope

Testing excludes kiosk hardware, online refunds, multilingual user interfaces, and any storage of raw payment card data.

7.4 Test Strategy and Methods

7.4.1 Unit Testing (Granularity 1)

Unit tests verify individual methods and utilities without external services.

- Techniques: xUnit style tests with mocks and stubs; boundary analysis; property-based tests for seat coordinates.
- Coverage Goals: ≥ 80 percent line coverage in Seat, Showtime, Ticket, and Order modules.
- Risks Covered: overselling due to race conditions, invalid seat identifiers, incorrect discount calculations, and QR format errors.

7.4.2 Functional and Integration Testing (Granularity 2)

Functional tests validate end-to-end API and service interactions such as authentication, orders, payments, notifications, administrative showtime management, and staff ticket validation.

- Techniques: API contract testing, positive and negative scenarios, fault injection (payment decline, email failure), and concurrency simulation for simultaneous purchases.
- Mocks: Payment gateway sandbox, email capture server, and static review API fixtures.

7.4.3 System and End-to-End Testing (Granularity 3)

System tests confirm that all modules work together under production-like conditions.

- Techniques: browser automation tests for desktop and mobile layouts, load testing for 1 000 concurrent users, accessibility audits (keyboard navigation and screen reader), and health-check monitoring for uptime.

7.5 Test Environment

Backend: Node.js v18 with Express.js and SQL database (MySQL or PostgreSQL).
Frontend: latest Chrome, Firefox, and Safari browsers.
Operating Systems: Windows 11, macOS Monterey, Ubuntu 22.04.
Tools: Jest, Mocha, Postman, Playwright, k6 for load, axe-core for accessibility, and GitHub Actions for continuous integration.

7.6 Entry and Exit Criteria

Entry Criteria: Core endpoints implemented, test data seeded, and mock services configured.
Exit Criteria: All unit tests ≥ 80 percent coverage, no critical failures, all functional and system tests pass, accessibility checks clean, and load targets met.

7.7 Test Data

Seeded sample data includes ten movies, three auditoriums (each 150 seats), five user accounts (Customer, Staff, Admin), five daily showtimes, and one hundred sample transactions for load tests.

7.8 Test Cases Summary

Test Case ID	Objective	Inputs	Expected Result	Test Level
TC-01	Validate login authentication	Valid credentials	User redirected to dashboard	Unit
TC-02	Verify seat hold expiration	Select seat, wait 5 minutes	Seat released to available	Unit
TC-03	Confirm payment approval	Valid card details	Order confirmed and ticket issued	Functional
TC-04	Generate unique QR	Completed order	One unique QR per seat	Functional
TC-05	Validate QR at entry	Scan valid QR	Ticket marked as used	Functional

Test Case ID	Objective	Inputs	Expected Result	Test Level
TC-06	Reject duplicate QR	Rescan used QR	“Ticket Already Used” error displayed	Functional
TC-07	Measure load performance	1 000 users	95 percent requests < 2 s	System
TC-08	Database recovery test	Simulated crash	No data loss after restart	System
TC-09	Email notification test	Successful payment	Receipt and QR sent to user	System
TC-10	Accessibility check	Screen reader mode	All elements readable and navigable	System

7.9 Traceability

Each test maps to functional and non-functional requirements from Section 3. Examples: TC-01 → FR 3.1.1, TC-02 → FR 3.1.2, TC-03 to TC-06 → FR 3.1.5 – 3.1.7, and TC-07 to TC-10 → NFR 3.5.1 – 3.5.6. A complete traceability matrix is included in the Excel file **testCasesSample.xlsx**.

7.10 Summary

This Test Plan establishes a comprehensive approach for testing CineGet across unit, functional, and system levels. It ensures that every requirement defined in the SRS is verified, preventing defects and confirming system stability, security, and performance before deployment.

8. Architecture Update and Data Management Strategy

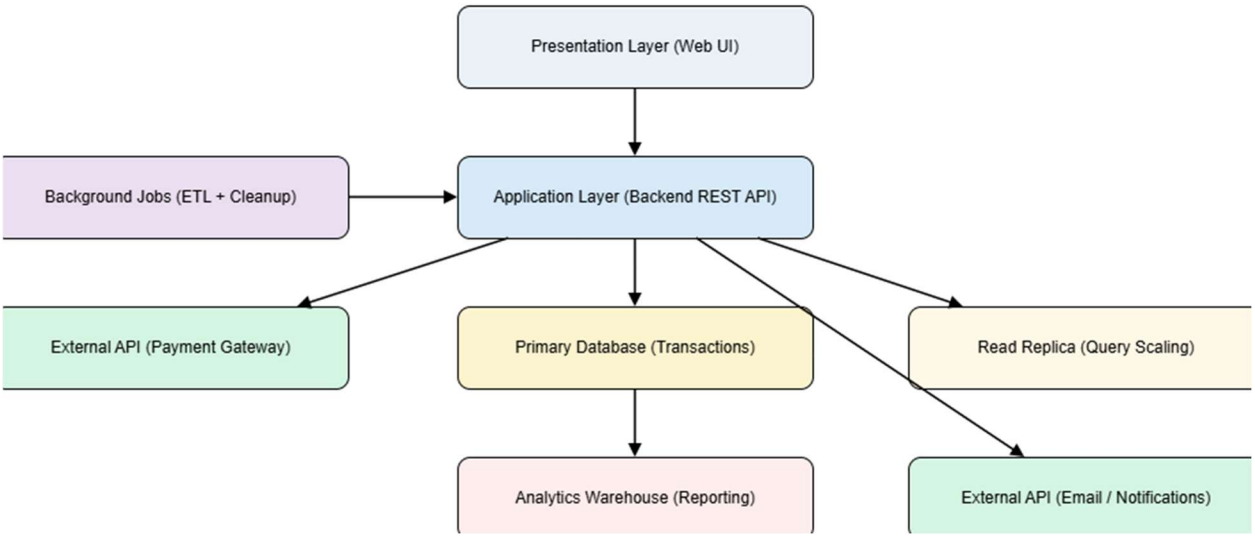
8.1 Purpose

This section describes updates to the software architecture and details the data management strategy for CineGet. It defines how databases are structured, secured, and scaled to support reliable operation and long-term maintenance.

8.2 Updated Architecture Overview

The three-tier design from Section 6 remains unchanged but now details the data layer implementation. The architecture explicitly includes one primary transactional database, one read replica for query scaling, and one analytics warehouse for reporting. A background service within the application layer handles data migration to the warehouse and expired seat cleanup.

Figure 8.1 – Updated Software Architecture Diagram (CineGet)
(Insert updated diagram here showing Primary DB, Replica, Warehouse, and Background Jobs Service.)



8.3 Data Model and Schema

The data model follows a relational structure with entities User, Movie, Auditorium, Seat, Showtime, Order, and Ticket. Primary and foreign keys enforce referential integrity. Seat states transition from available → held → sold → used. Composite indexes on showtime and theater identifiers optimize seat map queries.

8.4 Database Topology

CineGet uses a clustered SQL setup with one write node and one or more read replicas. Replication is continuous and asynchronous. An external data warehouse receives nightly ETL batches for long-term analytics and reporting. Backups run nightly with seven-day retention and encrypted storage.

8.5 Data Partitioning and Retention

Operational tables use a theater identifier to partition data for fast queries during peak load. Records older than six months move from the primary database to the warehouse to maintain performance. Orders and tickets are retained for three years, feedback for one year, and logs for ninety days. Personal data is deleted on user request in compliance with privacy policies.

8.6 Security and Access Control

All database connections use transport-layer encryption. Sensitive information is hashed or stored through third-party PCI-compliant providers. Roles define permissions for Customer, Staff, and Administrator. Administrative actions and logins are logged and audited. Backups are encrypted with separate keys and stored in restricted object storage.

8.7 Backup and Recovery

Nightly full backups and incremental log shipping enable point-in-time recovery with an RPO of 15 minutes and an RTO of one hour. Quarterly restore tests validate the recovery process.

8.8 Alternatives and Tradeoffs

A document database was considered for seat maps and tickets due to its schema-flexibility and scalable reads. However, relational databases were chosen for ACID compliance and simpler reporting. This choice ensures transactional consistency for ticket sales and reduces complexity for joins and aggregations. Caching of seat maps via an in-memory store was rejected to avoid stale data and overselling.

8.9 Summary

The updated architecture and data management plan for CineGet strengthen performance, consistency, and security. By adding explicit database roles, partitioning strategies, and backup policies, the system can scale reliably and maintain integrity while supporting future growth and analytics needs.