

# 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.

# **Theater Ticketing System**

## **Software Requirements Specification**

**<Version>**

**7-8-2025**

**Group 10**

**Anh Vo, Luis Do, Michael Pham**

Prepared for  
CS 250- Introduction to Software Systems  
Instructor: Gus Hanna, Ph.D.  
Summer 2025

Revision History

Date	Description	Author	Comments
<date>	<Version 1>	<Your Name>	<First Revision>

Document Approval

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

Signature Printed Name Title Date			
	<Your Name>	Software Eng.	
	Dr. Gus Hanna	Instructor, CS 250	

# Table of Contents

<b>REVISION HISTORY .....</b>	<b>II</b>
<b>DOCUMENT APPROVAL .....</b>	<b>II</b>
<b>1. INTRODUCTION .....</b>	<b>1</b>
1.1 PURPOSE.....	1
1.2 SCOPE .....	1
1.3 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS .....	1
1.4 REFERENCES.....	1
1.5 OVERVIEW.....	1
<b>2. GENERAL DESCRIPTION.....</b>	<b>2</b>
2.1 PRODUCT PERSPECTIVE .....	2
2.2 PRODUCT FUNCTIONS .....	2
2.3 USER CHARACTERISTICS .....	2
2.4 GENERAL CONSTRAINTS.....	2
2.5 ASSUMPTIONS AND DEPENDENCIES .....	2
<b>3. SPECIFIC REQUIREMENTS.....</b>	<b>2</b>
3.1 EXTERNAL INTERFACE REQUIREMENTS.....	3
3.1.1 <i>User Interfaces</i> .....	3
3.1.2 <i>Hardware Interfaces</i> .....	3
3.1.3 <i>Software Interfaces</i> .....	3
3.1.4 <i>Communications Interfaces</i> .....	3
3.2 FUNCTIONAL REQUIREMENTS.....	3
3.2.1 <i>&lt;Functional Requirement or Feature #1&gt;</i> .....	3
3.2.2 <i>&lt;Functional Requirement or Feature #2&gt;</i> .....	3
3.3 USE CASES.....	3
3.3.1 <i>Use Case #1</i> .....	3
3.3.2 <i>Use Case #2</i> .....	3
3.4 CLASSES / OBJECTS.....	3
3.4.1 <i>&lt;Class / Object #1&gt;</i> .....	3
3.4.2 <i>&lt;Class / Object #2&gt;</i> .....	3
3.5 NON-FUNCTIONAL REQUIREMENTS .....	4
3.5.1 <i>Performance</i> .....	4
3.5.2 <i>Reliability</i> .....	4
3.5.3 <i>Availability</i> .....	4
3.5.4 <i>Security</i> .....	4
3.5.5 <i>Maintainability</i> .....	4
3.5.6 <i>Portability</i> .....	4
3.6 INVERSE REQUIREMENTS.....	4
3.7 DESIGN CONSTRAINTS .....	4
3.8 LOGICAL DATABASE REQUIREMENTS.....	4
3.9 OTHER REQUIREMENTS .....	4
<b>4. ANALYSIS MODELS.....</b>	<b>4</b>
4.1 SEQUENCE DIAGRAMS .....	5
4.3 DATA FLOW DIAGRAMS (DFD).....	5
4.2 STATE-TRANSITION DIAGRAMS (STD).....	5
<b>5. CHANGE MANAGEMENT PROCESS .....</b>	<b>5</b>
<b>A. APPENDICES.....</b>	<b>5</b>

A.1 APPENDIX 1.....5

A.2 APPENDIX 2.....5

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to outline the requirements and specifications for software engineers to develop a functioning ticketing system for theaters to use.

## 1.2 Scope

1. Movie Theater Ticketing System
2. The ticketing system will handle the seating and ticket sales for 20 theaters in the San Diego area. The system will be browser-based and accessible both inside and outside the theater. It will be an interface that allows consumers to purchase ticket(s) and pick seating, along with supporting various factors such as discounts and implementing various constraints to the purchasing of tickets.
3. The system will be implemented in devices inside theater lobby areas, allowing customers to easily purchase tickets and choose seating for upcoming showtimes. The goal of the system is to provide a smooth and streamlined experience to choosing a movie, picking seating, and paying for the ticket.

## 1.3 Definitions, Acronyms, and Abbreviations

- SRS – Software Requirement Specification
- UI – User Interface
- DB – Database

## 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.*

Reference ID	Document Title	Version/report	Date	Publisher/source	URL
A	IEEE Std 830-1998: Recommended Practice for Software Requirements Specification	IEEE Std 830-1998	1998	IEEE Computer Society	<a href="https://sdsu.instructure.com/courses/178325/files/17281459?module_item_id=4921119">https://sdsu.instructure.com/courses/178325/files/17281459?module_item_id=4921119</a>
B	Theater	n/a	Summer 2025	Dr. Gus	<a href="https://sdsu.in">https://sdsu.in</a>

	Ticketing Requirements Examples			Hanna, SDSU CS 250 (Canvas module)	<a href="https://structure.com/courses/178325/files/17281417?module_item_id=4921121">structure.com/courses/178325/files/17281417?module_item_id=4921121</a>
C	Stripe API Referenced	API v2025-03-15	March 15 2025	Stripe, Inc.	<a href="https://docs.stripe.com/api">https://docs.stripe.com/api</a>
D	RFC 7519: JSON Web Token (JWT)	RFC 7519	May 2015	Internet Engineering Task Force (IETF)	<a href="https://datatracker.ietf.org/doc/html/rfc7519">https://datatracker.ietf.org/doc/html/rfc7519</a>
E	PostgreSQL 13 Documentation	v13	2020	PostgreSQL Global Development Group	<a href="https://www.postgresql.org/docs/13/">https://www.postgresql.org/docs/13/</a>

## 1.5 Overview

1. The rest of this document will contain design constraints, expectations, requirements, and functionality expected from the system.
2. The document is organized into sections, and those sections into subsections. Every section is labeled with enlarged and bold headers. Every subsection will cover a specific topic that branches out from the main section.

## 2. General Description

### 2.1 Product Perspective

Similar ticketing systems have been implemented in many other theaters and adjacent industries. This ticketing system will be catered to our client's needs as opposed to other systems on the market.

### 2.2 Product Functions

- Handle at least 1000 users at once
- Be able to run in a browser and be accessible both inside and outside theaters
- Handle security of purchases, and block bot attacks that are looking to mass buy tickets

- Interface with existing databases to display available showtimes, and tickets
- Add constraints to purchases: 20 ticket limit per user, availability starting from 2 weeks before showtime, availability ending 10 minutes after showtime
- Support factors in ticket price, such as discounts
- Support an administrator mode for theater staff to handle issues
- Be able to scrape info from review sites online to display movie ratings, reviews, and critic quotes

### 2.3 User Characteristics

Users will vary highly due to the demographics of theaters spanning almost all ages. Therefore the UI should be highly accessible, and streamlined. The UI should be easy to maneuver for users in both ends of the age range, and for users not tech savvy.

### 2.4 General Constraints

The ticketing system should be fully functional with touch screen and keyboard input. Additionally, as most ticketing system devices in theaters do not come with a keyboard, a built in onscreen keyboard will be supported.

### 2.5 Assumptions and Dependencies

It is assumed that devices inside theaters running the ticketing system will have a strong and stable internet connection, and be able to load the application smoothly and quickly. It is also assumed that theater devices will have touch screen support. The devices are also assumed to be able to run Chromium.

## 3. Specific Requirements

*This will be the largest and most important section of the SRS. The customer requirements will be embodied within Section 2, but this section will give the D-requirements that are used to guide the project's software design, implementation, and testing.*

*Each requirement in this section should be:*

- *Correct*
- *Traceable (both forward and backward to prior/future artifacts)*
- *Unambiguous*
- *Verifiable (i.e., testable)*
- *Prioritized (with respect to importance and/or stability)*
- *Complete*
- *Consistent*
- *Uniquely identifiable (usually via numbering like 3.4.5.6)*

*Attention should be paid to the carefully organize 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 User Interfaces



- Optional login screen
  - Inputs: email, password, date of birth
  - Outputs: “Welcome <Name>” banner, or error message.
  - Constraints: Must display on 1920x1080 computer screens and follow respective aspect ratios.
- Movie Listing pages
  - Displays: Movie poster thumbnails, title, showtimes, average ratings
  - Controls: “Select seats” options under each showtime.
- Seat Selection Functionalities
  - Available seats are green, taken seats are red, user-selected seats are gray.
  - Display the row number and position of all seats on hover.
  - Constraints: To prevent multiple users selecting and buying the same seats concurrently, implement first come first serve (e.g. who ever clicked on seats first gets put in “not available” state for 5 minutes and show gray).

### 3.1.2 Hardware Interfaces

- Kiosk Touchscreen
  - 10 inch screens that handles capacitive touch (touchscreen). No physical keyboard or mouse.
  - **Driver: TBA**
- Barcode scanner using a mobile camera for scanning printed or digital tickets

### 3.1.3 Software Interfaces

- Database
  - Type: PostgreSQL 13
- Payment Gateway
  - Stripe REST API v2025-06-30.basil

### 3.1.4 Communications Interfaces

- Web
  - Protocol: HTTPS

## 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 <User Authentication>

#### 3.2.1.1 Introduction

- Allows users to register, log in, and log out securely

#### 3.2.1.2 Inputs

- Email (String)
- Password (String)
- Date of birth (Integer)

#### 3.2.1.3 Processing

1. Validate ‘email’ that matches the standard email regex  
(`^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$`)

2. For registration: hash 'password' to prevent security breaches using bcrypt and store '{email, hash}' in 'Users' table in DB.
3. For logging in, retrieve the stored hash for 'email' and compare it with bcrypt to ensure authentication.
4. Once logged in, prompt the user to "remember login". If the user declines, generate a JWT token that expires in 1 hour to terminate the user session.

### 3.2.1.4 Outputs

- On successful login, output HTTP 200 + JSON
- Json
  - {"userId": "123",
  - "name": "John Doe",
  - "token": "<JWT>"}
- On unsuccessful login, output HTTP 401 + JSON
- json
  - {"error": "Invalid email or password, please try again."}

### 3.2.1.5 Error Handling

- If email or password is missing, output HTTP 400 + JSON  
json:
  - {"error": "Email and password required"}
- If server error output HTTP 500 + JSON  
json: {"error": "Server error, please try again later."}
- 

## 3.2.2 <Browsing Shows and Times>

### 3.2.2.1 Introduction

- After a user successfully logs in, the system will fetch and display current movies and their available showtimes.

### 3.2.2.2 Inputs

- Session token: JWT token is sent in the HTTP header.
- Date – shows only showtimes on a specific day
- theatreID – show only one theatre's movie listings.
- searchTerm – text search to look for movie titles.

### 3.2.2.3 Processing

1. Validate session token and check if user is authenticated.
2. If user changes filters for search, apply them to narrow movie list query
3. Query the movie tables for all movies marked "active" from DB.
4. For each movie, also query Showtimes table for upcoming showtimes alongside current times.
5. Sort by movie name, then showtime chronologically.
6. Format combined data into a response object.

### 3.2.2.4 Outputs

- On successful completion, outputs HTTP 200.
- Rendered UI: grid of movie listings/titles and clickable showtime buttons.

### 3.2.2.5 Error Handling

- If session token is missing or invalid, output HTTP 401 unauthorized + json {"error":

“Authentication required”}

- If a filter is malformed (messed up date format), output HTTP 400 bad request + json {"error": "Invalid date format"}
- If server or database error, output HTTP 500 internal server error + json {"error": "Unable to fetch show data; please try again later."}

## 3.3 Use Cases

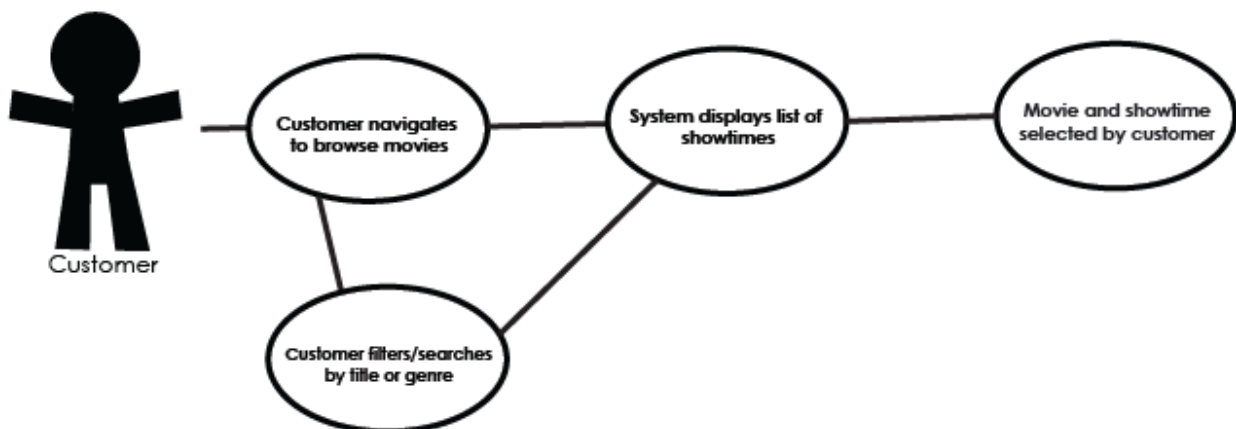
### 3.3.1 Browse and Select Showtime

- Actors: Customer.
- Precondition: Customer opens website.
- Post-con: customer selects movie and showtime

MAIN FLOW:

- Customer navigates to browse movies
- System displays list of movies with showtimes and ratings
- Customer filters by theatre and/or time slot
  - Or by title
- System dynamically updates / displays matching results

#### BROWSE AND SELECT SHOW TIME



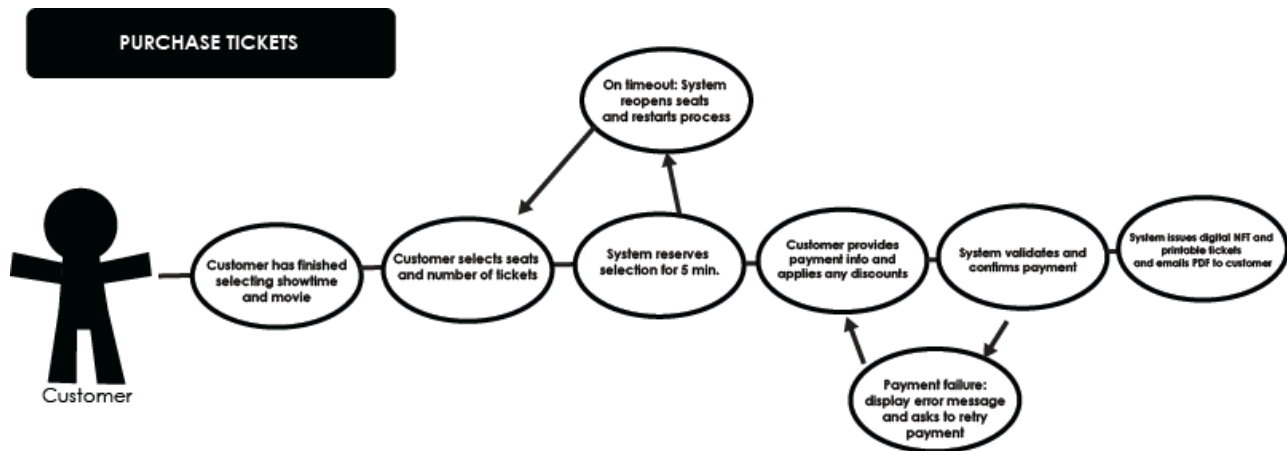
### 3.3.2 Purchase Tickets

- Actors: customer, payment gateway
- Precon: customer select showtime
- Postcon: tickets confirmed and delivered

MAIN FLOW:

- Customer selects seats and number of tickets
- System reserves selection for 5 min

- On timeout: System reopens seats and restarts process
- Customer provides payment (and discount if any) info
- System validates payment (using wtv software/security... idk this)
  - On payment failure: system displays error message and asks to retry payment
- System issues digital NFT and printable tickets and emails PDF to customer

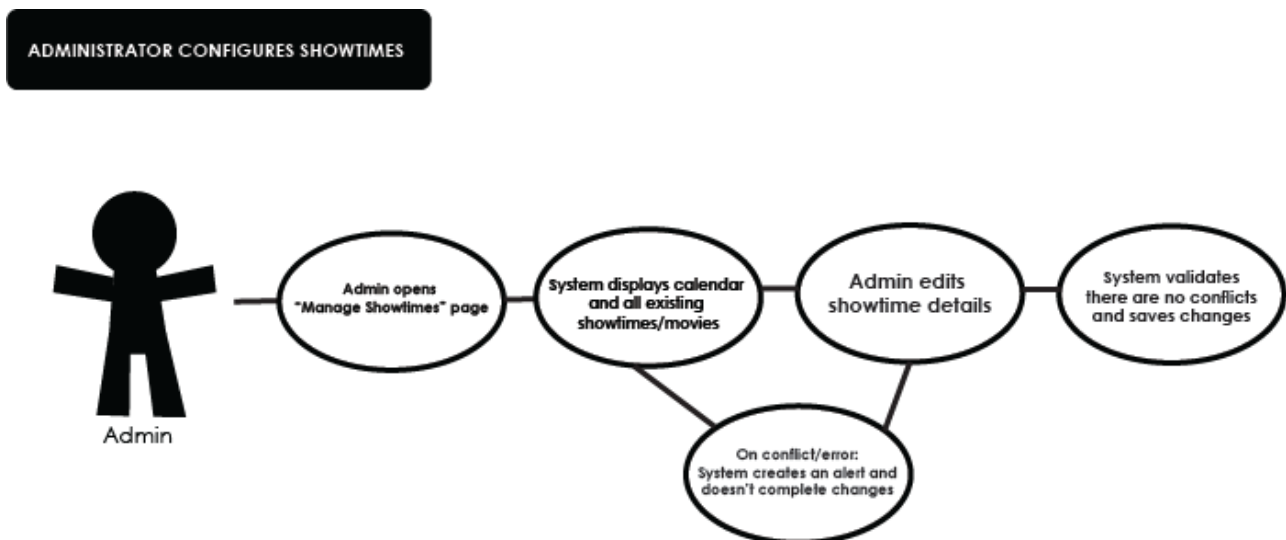


### 3.3.3 Administrator Configures Showtimes

- Actors: admin
- Precon: admin verified into system
- Postcon: new showtime active

#### MAIN FLOW:

- Admin opens “Manage Showtimes” page
- System displays calendar and all existing showtimes/movies
- Admin edits showtime details
  - If conflict, system alerts and requires resolving
- System validates if there are 0 conflicts (check first) and saves changes



## 3.4 Classes / Objects

### 3.4.1 <Movie>

#### 3.4.1.1 Attributes

- title, rating, duration, reviewScores

### 3.4.1.2 Functions

- [getShowtimes\(\)](#), [getReviews\(\)](#)

<Reference to functional requirements and/or use cases>

### 3.4.2 <Theater>

#### 3.4.2.1 Attributes

- id, name, location, seatLayout, availableSeats

#### 3.4.2.2 Functions

- [getAvailableSeats\(showtimeId\)](#)

### 3.4.3 <Showtime>

#### 3.4.3.1 Attributes

- id, movieId, theaterId, startTime

#### 3.4.3.2 Functions

- [reserveSeats\(\)](#), [releaseSeats\(\)](#)

### 3.4.4 <Ticket>

#### 3.4.4.1 Attributes

- id, showtimeId, seatNumber, ticketType, price, nftToken

#### 3.4.4.2 Functions

- [generateDigitalTicket\(\)](#), [printPDF\(\)](#)

## 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 process of picking a movie, choosing seating, and purchasing a ticket should be able to be completed within two minutes by users.

### 3.5.2 Reliability

The system should prioritize reliability and stability because an error in the ticketing system can quickly cause a build up and cause frustration for numerous customers. If all ticketing kiosks were to stop functioning, the theater's performance would severely decline due to their reliance on the ticketing system. The mean time between failures should be around 30 days, and recovery from downtime should be around 10 minutes with assistance from staff.

### 3.5.3 Availability

Due to the ticketing system running through a browser, it should be available during all times with internet connection.

### 3.5.4 Security

The main security concern revolves around payments. Stripe API which is being used for handling payments, has built in security features and data encryption for payments. The system should utilize the API properly in order to handle security properly.

### **3.5.5 Maintainability**

The system will be maintained through software updates over the internet when needed. Updates should be relatively quick and be able to be completed before opening hours.

### **3.5.6 Portability**

Because the ticketing system runs through a browser, it should be able to be easily integrated into different locations. Installation and setup shouldn't require more than internet connection.

## **3.6 Inverse Requirements**

*State any \*useful\* inverse requirements.*

## **3.7 Design Constraints**

*Specify design constraints imposed by other standards, company policies, hardware limitation, etc. that will impact this software project.*

## **3.8 Logical Database Requirements**

*Will a database be used? If so, what logical requirements exist for data formats, storage capabilities, data retention, data integrity, etc.*

## **3.9 Other Requirements**

*Catchall section for any additional requirements.*

# **4. Analysis Models**

*List all analysis models used in developing specific requirements previously given in this SRS. Each model should include an introduction and a narrative description. Furthermore, each model should be traceable the SRS's requirements.*

## **4.1 Sequence Diagrams**

## **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**

# Software Design Specification

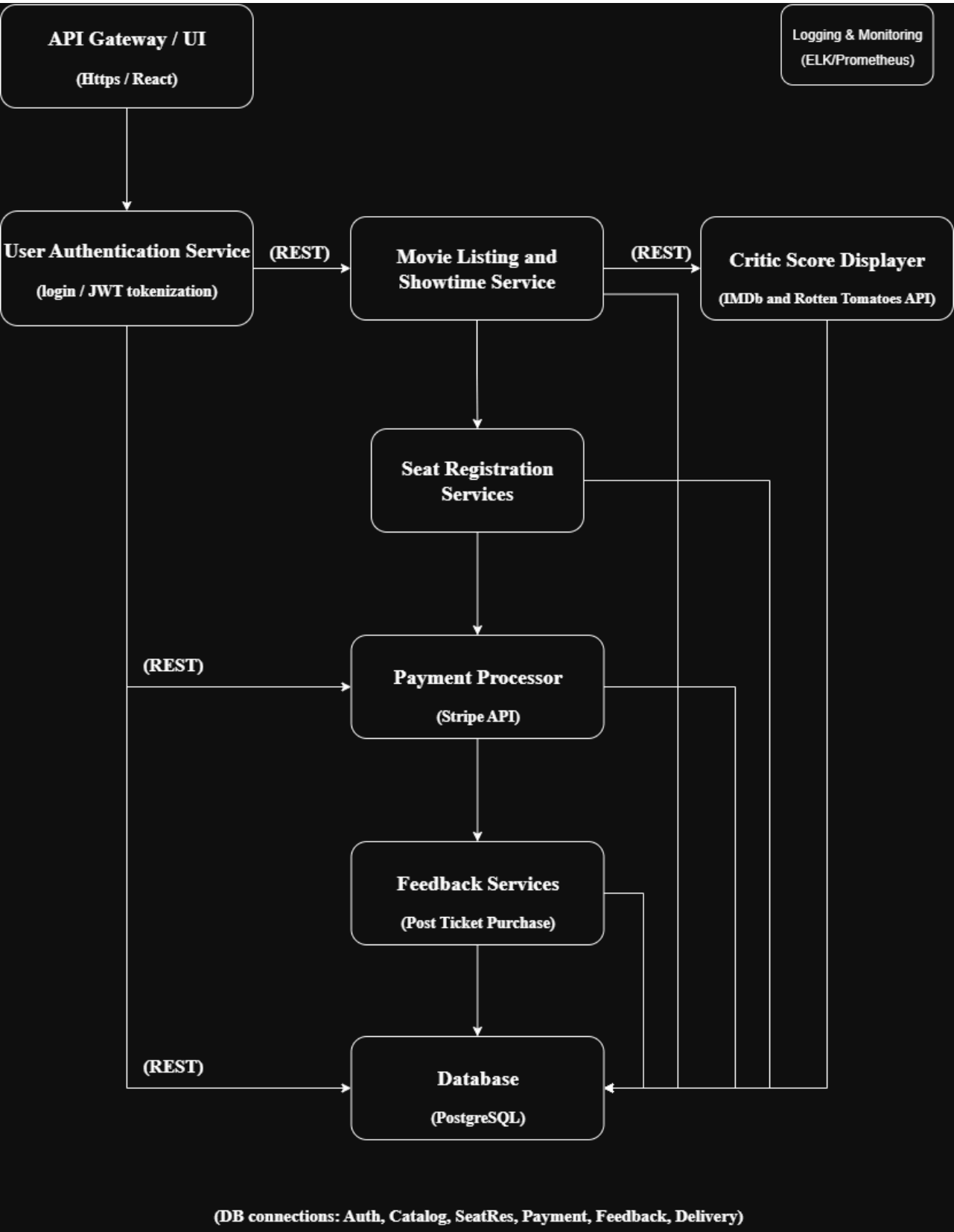
**Group 10**

**Anh Vo, Luis Do, Michael Pham**

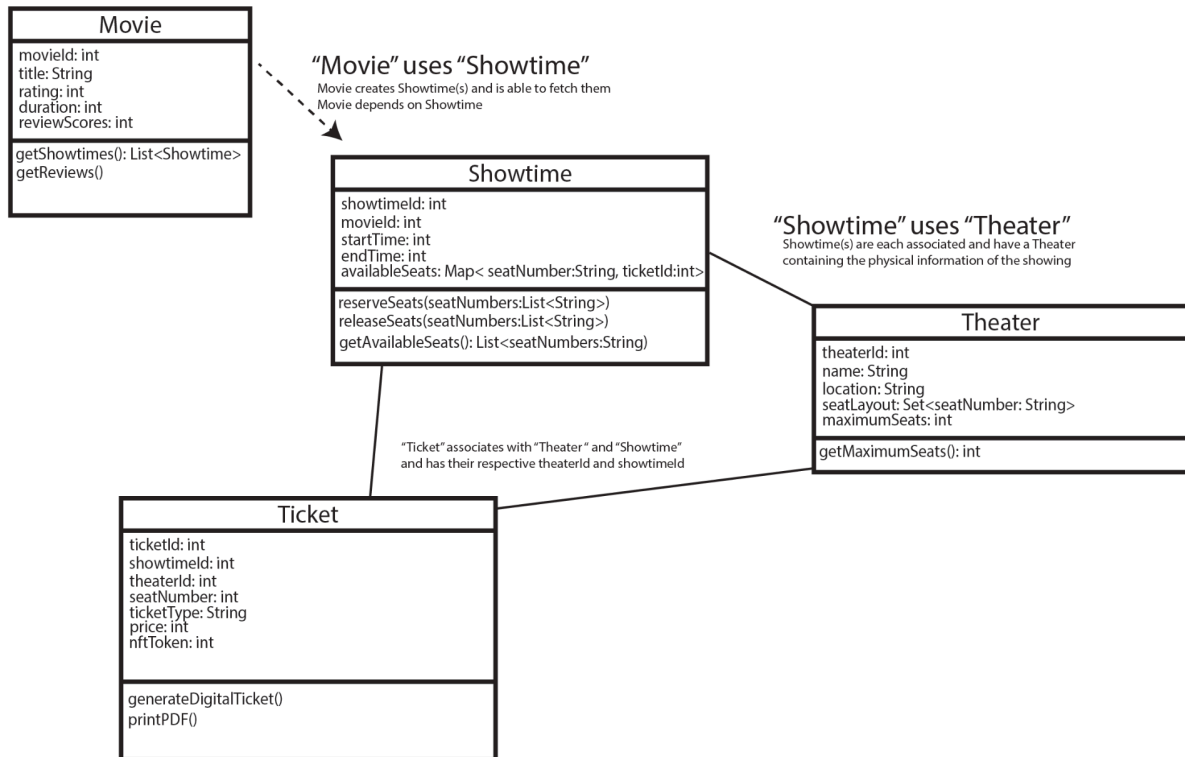


## System Description

All client interactions begin at the API Gateway (HTTPS/React), which routes requests to the User Authentication Service. Upon successful JWT validation (against PostgreSQL), authenticated requests flow into the Movie Listing & Showtime Service, which runs the Critic Score Displayer (IMDb/Rotten Tomatoes API). Seat selection requests are forwarded to the Seat Reservation Service. Once seats are held, the Payment Processor (Stripe API) charges the user and then displays the Feedback Service to capture post-purchase surveys. Finally, each service independently reads from and writes to the central PostgreSQL Database over secured SQL/TLS connections, and all components emit logs and metrics to the Logging & Monitoring stack (ELK/Prometheus) for centralized observability for developers.



## 6. UML Class Overview



### Movie:

- Attributes: movieId, title, rating, duration, reviewScores
- Functions: getShowtimes(), getReviews()
- The attributes contain basic information about the movie. The Movie class will create Showtime objects and is able to fetch showtimes for the movie with getShowtimes().
- Interaction:
  - When a user selects a movie, the system queries the Movie class using getShowtimes() to pull all upcoming showtimes. These showtimes are then presented alongside their associated theater information.

### Showtime:

- Attributes: showtimeId, movieId, startTime, endTime, availableSeats
- Function: reserveSeats(), releaseSeats(), getAvailableSeats()
- The attributes contain info about the movie during the showtime, including the start and end time of the showing. Showtime class will also keep track of, and manage available seats. Each Showtime class will have a Theater object that contains the information about the physical location of the theater and

maximum seating.

- Interaction:
  - If a reservation is canceled or times out, `releaseSeats()` is called to restore availability.
  - When a user selects a specific showtime, the system calls `getAvailableSeats()` to display open seats.
  - When the user reserves one or more seats, `reserveSeats()` is invoked, updating the internal availability map.

Theater:

- Attributes `theaterId`, `name`, `location`, `seatLayout`, `maximumSeats`
- Functions: `getMaximumSeats()`
- The attributes contain the information about the physical theater, including its location, name, and seating information.
- Interaction:
  - During Showtime creation, the theater details are linked to ensure the seat layout matches.
  - The seat layout is used when generating the `availableSeats` structure in the Showtime.
  - This allows the system to render an accurate seating chart for users.

Ticket:

- Attributes `ticketId`, `showtimeId`, `theaterId`, `seatNumber`, `ticketType`, `price`, `nftToken`
- Functions: `generateDigitalTicket()`, `printPDF()`
- The attributes contain the information about the ticket including information relevant for the consumer such as price, whilst holding information needed for the system such as seating and NFT token.
- Interaction:
  - Once a user completes seat selection and payment, a Ticket object is created.
  - The system sets `showtimeId` and `theaterId` from the selected Showtime and its linked Theater.
  - `generateDigitalTicket()` might be used to store or display a scannable digital pass, while `printPDF()` is used for traditional output.
  - The `nftToken` attribute optionally supports blockchain-based verification or collectible features.

Interaction Flow Example:

1. A user selects a movie from a list.
2. The system calls `getShowtimes()` from the Movie object to display all future showtimes.
3. The user selects a showtime, prompting the system to retrieve associated Theater information and seat layout.

4. The system displays available seats by calling `getAvailableSeats()` from the Showtime.
5. The user selects a seat and confirms the purchase.
6. The system calls `reserveSeats()` to lock the seat.
7. Upon payment, a Ticket object is created and linked to the selected Showtime, Theater, and seatNumber.
8. The system calls `generateDigitalTicket()` or `printPDF()` to deliver the ticket to the user.

## 7. Development Plan and Timeline

### Overview:

As stated above, the movie ticketing system will handle the seating and ticket sales for 20 theaters in the San Diego area, and the system will be implemented in devices inside theater lobby areas, allowing customers to easily purchase tickets and choose seating for upcoming showtimes.

The project development will follow a 16-week, agile-inspired iterative approach. The tasks are divided into four-week sprints with key phases including finalization, design, implementation, integration, testing, and deployment.

### Team Roles:

- Project Manager: Coordinates the project schedule and communicates with stakeholders.
- UX Designer: Crafts wireframes and prototypes to shape the user experience.
- Backend Engineer: Develops and maintains server-side APIs and business logic.
- Frontend Engineer: Implements client-side functionality and integrates APIs.
- Database Administrator: Designs and optimizes the database schema.
- QA Engineer: Creates test plans and conducts testing to ensure quality.
- DevOps Engineer: Builds and maintains CI/CD pipelines and deployment environments.

## Timeline with Tasks:

Task	Description	Roles	Start Week	End Week
Finalize Requirements	Refine and approve all requirements	Project Manager	1	2
UI/UX Design	Create wireframes and high-fidelity mockups	UX Designer	1	4
Database Schema Design	Define logical and physical database schema	Database Administrator	3	5
Backend API Development	Implement core RESTful services (Stripe API)	Backend Engineer	5	8
Frontend Development	Develop web UI components and integrate with APIs	Frontend Engineer	6	9
Integration	Integrate frontend, backend, and external services	Backend Engineer, Frontend Engineer	9	10
Testing and Quality Assurance	Perform unit, integration, and performance testing	QA Engineer	10	12
Deployment and DevOps	Configure CI/CD pipelines and deploy to production	DevOps Engineer	11	13
Pilot Launch	Internal demo and stakeholder feedback collection	Everyone	13	14
Final Release	Address issues and release version 1.0	Everyone	14	16

## 8. Test Plan

### Test Cases Spreadsheet

#### 8.1 Updated Design Specification

Additional changes were made to the UML class description since Assignment 2. Interactions between the classes and the flow of the system was expanded upon. Our existing classes (Movie, Showtime, Theater, Ticket) and their functions remain suitable for all planned tests.

#### 8.2 Test Plan Overview

This Test Plan defines how we will verify that each class and API behaves as specified, and validate by exercising full end-to-end purchase flows. We will cover three levels of testing: unit, integration, and system, to achieve comprehensive coverage.

8.3 Test Strategy and Target Components

Level	Target Component	Strategy and Test Vectors
Unit		
Integration		
System		

8.1 Updated Design Specification