# **Software Requirement**

# **Specifications**

**for**

**AtoZflix**

**Preparedby**

*Zain Baig 22k-4593 Asad Irfan 22k-4276*

*Copyright © 2024 by Campick*

# **Table Of Contents**

[Software Requirement 1](#_Toc49780)

[Specifications 1](#_Toc49781)

[**Table Of Contents** **2**](#_Toc49782)

[Software Requirements Specification - AtoZflix 6](#_Toc49783)

[1. Introduction 6](#_Toc49784)

[1.1 Purpose 6](#_Toc49785)

[1.2 Scope 6](#_Toc49786)

[1.2.1 Core Functionalities 6](#_Toc49787)

[1.2.2 Business Objectives 6](#_Toc49788)

[1.2.3 Target Benefits 6](#_Toc49789)

[1.3 Document Conventions 6](#_Toc49790)

[1.3.1 Priority Levels 6](#_Toc49791)

[1.3.2 Technical Conventions 7](#_Toc49792)

[1.4 References 7](#_Toc49793)

[1.4.1 Technical Documentation 7](#_Toc49794)

[1.4.2 API Documentation 7](#_Toc49795)

[1.4.3 Standards 7](#_Toc49796)

[**2. Overall Description** **7**](#_Toc49797)

[2.1 Product Perspective 7](#_Toc49798)

[2.1.1 Frontend Layer 7](#_Toc49799)

[2.1.2 Backend Services 7](#_Toc49800)

[2.1.3 External Integrations 7](#_Toc49801)

[2.2 Product Functions 8](#_Toc49802)

[2.2.1 User Authentication 8](#_Toc49803)

[2.2.2 Shop Management 8](#_Toc49804)

[2.2.3 Order System 8](#_Toc49805)

[2.2.4 Payment System 8](#_Toc49806)

[2.2.5 Analytics & Reporting 8](#_Toc49807)

[2.3 User Classes and Characteristics 9](#_Toc49808)

[2.3.1 Students 9](#_Toc49809)

[2.3.2 Teachers 9](#_Toc49810)

[2.3.3 Shop Owners 9](#_Toc49811)

[2.4 Operating Environment 9](#_Toc49812)

[2.5 Design and Implementation Constraints 9](#_Toc49813)

[2.5.1 Technical Constraints 9](#_Toc49814)

[2.5.2 Security Constraints 10](#_Toc49815)

[2.5.3 Business Constraints 10](#_Toc49816)

[2.5.4 Integration Constraints 10](#_Toc49817)

[2.7 Assumptions and Dependencies 10](#_Toc49818)

[2.7.1 Technical Dependencies 10](#_Toc49819)

[2.7.2 External Services 10](#_Toc49820)

[2.7.3 Assumptions 11](#_Toc49821)

[2.8 Process Model 11](#_Toc49822)

[2.8.1 Software Development Strategy 11](#_Toc49823)

[2.9 Project Plan 12](#_Toc49824)

[2.9.1 Timeline and Milestones 12](#_Toc49825)

[2.9.2 Resource Allocation 12](#_Toc49826)

[2.9.3 Deliverables 13](#_Toc49827)

[2.10 Feasibility Report 13](#_Toc49828)

[2.10.1 Technical Feasibility 13](#_Toc49829)

[2.10.2 Economic Feasibility 13](#_Toc49830)

[2.10.3 Operational Feasibility 13](#_Toc49831)

[2.12.4 Schedule Feasibility 13](#_Toc49832)

[2.10.5 Risk Assessment 14](#_Toc49833)

[2.11 Homogenization Process 14](#_Toc49834)

[2.11.1 Code Standards 14](#_Toc49835)

[2.11.2 Architecture Standardization 14](#_Toc49836)

[2.11.3 Testing Standards 14](#_Toc49837)

[2.11.4 Documentation Standards 14](#_Toc49838)

[2.11.5 Quality Assurance Process 15](#_Toc49839)

[2.11.6 Deployment Process 15](#_Toc49840)

[2.12 Use case descriptions 15](#_Toc49841)

[2.12.1 Use Case 1: Place Order 15](#_Toc49842)

[2.12.2 Use Case 2: Process Payment 16](#_Toc49843)

[2.12.3 Use Case 3: Shop Dashboard 16](#_Toc49844)

[2.12.4 Use Case 4: Order Management 17](#_Toc49845)

[2.12.5 Use Case 5: Audit Logging 17](#_Toc49846)

[2.12.6 Use Case 6: Revenue Tracking 18](#_Toc49847)

[2.13 Video Link 18](#_Toc49848)

[**3. UML Concepts and Associations** **18**](#_Toc49849)

[3.1 UML Associations 18](#_Toc49850)

[3.2 Realizations & Dependencies 19](#_Toc49851)

[3.3 Advanced UML Elements 19](#_Toc49852)

[**4. External Interface Requirements** **19**](#_Toc49853)

[4.1 User Interfaces 19](#_Toc49854)

[4.1.1 General Characteristics 19](#_Toc49855)

[4.1.2 General Characteristics 19](#_Toc49856)

[4.1.3 Software Components Requiring User Interfaces 19](#_Toc49857)

[4.2 Hardware Interfaces 20](#_Toc49858)

[4.2.1 Server Requirements 20](#_Toc49859)

[4.2.2 Client Requirements 20](#_Toc49860)

[4.2.3 Database Server 20](#_Toc49861)

[4.3 Software Interfaces 20](#_Toc49862)

[4.3.1 Database 20](#_Toc49863)

[4.3.2 Libraries and Tools 20](#_Toc49864)

[4.3.3 Data Items and Messages 20](#_Toc49865)

[4.3.4 Data Items and Messages 20](#_Toc49866)

[4.4 Communication Interfaces 21](#_Toc49867)

[4.4.1 Email 21](#_Toc49868)

[4.4.2 Web Browser 21](#_Toc49869)

[4.4.3 Network Server Communication 21](#_Toc49870)

[4.4.4 Message Formatting 21](#_Toc49871)

[4.4.5 Communication Standards 21](#_Toc49872)

[4.4.5 Data Transfer Rates and Synchronization 21](#_Toc49873)

[**5. System Features** **21**](#_Toc49874)

[5.1 Authentication System 21](#_Toc49875)

[5.1.1 Description 21](#_Toc49876)

[5.1.2 Functional Requirements 21](#_Toc49877)

[5.2 Order Management 22](#_Toc49878)

[5.2.1 Description 22](#_Toc49879)

[5.2.2 Functional Requirements 22](#_Toc49880)

[5.3 Payment System 22](#_Toc49881)

[5.3.1 Description 22](#_Toc49882)

[5.3.2 Functional Requirements 22](#_Toc49883)

[5.4 Shop Management 22](#_Toc49884)

[5.4.1 Description 23](#_Toc49885)

[5.4.2 Functional Requirements 23](#_Toc49886)

[5.5 Analytics and Reporting 23](#_Toc49887)

[5.5.1 Description 23](#_Toc49888)

[5.5.2 Functional Requirements 23](#_Toc49889)

[5.6 Real-time Notification System 23](#_Toc49890)

[5.6.1 Description 23](#_Toc49891)

[5.6.2 Functional Requirements 23](#_Toc49892)

[5.7 User Management 24](#_Toc49893)

[5.7.1 Description 24](#_Toc49894)

[5.7.2 Functional Requirements 24](#_Toc49895)

[5.8 Image Management 24](#_Toc49896)

[5.8.1 Description 24](#_Toc49897)

[5.8.2 Functional Requirements 24](#_Toc49898)

[**6. Other Nonfunctional Requirements** **24**](#_Toc49899)

[6.1 Performance Requirements 24](#_Toc49900)

[6.2 Security Requirements 24](#_Toc49901)

[6.3 Software Quality Attributes 25](#_Toc49902)

[6.4 Business Rules 25](#_Toc49903)

[**7. Other Requirements** **25**](#_Toc49904)

[**Appendix A: Glossary** **26**](#_Toc49905)

[**Appendix B:Analysis Models** **26**](#_Toc49906)

[1. Class Diagram 26](#_Toc49907)

[2. Component Diagram 27](#_Toc49908)

[3. Deployment Diagram 27](#_Toc49909)

[4. Entity Relation Diagram 28](#_Toc49910)

[5. Package Diagram 29](#_Toc49911)

[6. Use Case Diagram 29](#_Toc49912)

[7. Sequence Diagram 30](#_Toc49913)

[8. Activity Diagram 31](#_Toc49914)

[9. State Machine Diagram 32](#_Toc49915)

[10. Subsystem Diagram 32](#_Toc49916)

[11. System Architecture (2-Tier, 3-Tier, N-tier) 33](#_Toc49917)

[12. Entity Control Boundary class diagram 34](#_Toc49918)

[13. Communication Diagram 35](#_Toc49919)

[14. Collaboration Diagram 35](#_Toc49920)

# **Software Requirements Specification AtoZflix**

# **1. Introduction**

### 1.1 Purpose

### This document outlines the technical and functional specifications for AtoZflix, a comprehensive movie management and recommendation system. The primary aim of AtoZflix is to provide users with a personalized and engaging movie discovery experience through features like trending movies, advanced search and sorting, ratings, reviews, and watchlist management. The document serves as a reference for developers, stakeholders, and testers to ensure a streamlined development process and deliver a feature-rich application.

##### 1.2.1 Core Functionalities

 Highlight trending movies, cast, and crew.

 Advanced search and sorting options based on title, genre, actors, and release date.

 User interaction features like ratings, reviews, and watchlist management.

#####  Admin functionalities for managing movie data and user profiles.

##### 1.2.2 Business Objectives

##### AtoZflix aims to enhance user engagement through personalized features while driving data efficiency for administrators. It seeks to establish itself as a go-to platform for movie discovery and management, ultimately increasing user retention and satisfaction.

##### 1.2.3 Target Benefits

 Simplified movie discovery for users.

 Enhanced engagement through interactive features like ratings and reviews.

 Efficient database management for administrators.

###  Insights into trending content, improving user recommendations.

### 1.3 Document Conventions

##### 1.3.1 Priority Levels

The priority of requirements and functionalities is categorized as:

* **High Priority**: Features essential for the application’s core functionality, such as user registration, trending insights, and database management.
* **Medium Priority**: Secondary features like advanced sorting and analytics dashboards.
* **Low Priority**: Features that enhance user experience, such as UI themes or detailed explainability for recommendations.

##### 1.3.2 Technical Conventions

 **Programming Language**: Python for backend (Flask) and Svelte for frontend.

 **Database**: SQLite for local data storage.

 **API Standards**: RESTful APIs for communication between frontend and backend.

###  Data Formats: JSON for data exchange.

### 1.4 References

##### 1.4.1 Technical Documentation

 Flask and SQLite official documentation for backend development and database management.

#####  Svelte documentation for frontend development.

##### 1.4.2 API Documentation

 The Movie Database (TMDb) API documentation for fetching movie-related data.

#####  IMDb API for retrieving movie metadata.

##### 1.4.3 Standards

 W3C standards for web application development.

#  OWASP standards for ensuring application security.

# **2. Overall Description**

### 2.1 Product Perspective

AtoZflix is a web-based movie management and recommendation system designed to enhance the movie discovery process. It integrates with third-party APIs like TMDb and IMDb to fetch accurate movie metadata, trends, and ratings. The system combines content-based and collaborative filtering techniques to provide personalized recommendations. It also features an admin panel for effective database and user management, enabling efficient tracking and control of application data.

The product follows a three-tier architecture:

1. **Frontend Layer**: Built with Svelte, offering a responsive and user-friendly interface for seamless navigation.
2. **Backend Services**: Powered by Flask, handling user authentication, movie recommendations, and administrative tasks.
3. **Database**: Utilizes SQLite for storing user, movie, and interaction data, ensuring data consistency and integrity.

##### 2.1.3 External Integrations

1. **Payment Processing**
   1. Screenshot-based verification for secure transactions.
   2. Gemini AI for intelligent payment validation.
2. **Image Management**
   1. Cloudinary for efficient image storage and management.
   2. Image optimization for faster loading times.
3. **Authentication**
   1. Google OAuth for secure user authentication.
   2. JWT-based sessions for maintaining user state.
4. **Notifications**
   1. Email notifications via SMTP for user updates.
   2. Real-time socket updates for immediate communication.

### 2.2 Product Functions

AtoZflix offers a wide range of functionalities:

* **User Registration and Profile Management**: Allows users to create accounts, manage preferences, and update profiles.
* **Personalized Recommendations**: Provides suggestions based on user preferences and trends.
* **Movie Search and Details**: Enables advanced search with detailed information on selected movies.
* **Watchlist Management**: Users can create and manage a list of movies to watch later.
* **Ratings and Reviews**: Users can rate movies and provide feedback, influencing recommendations.
* **Admin Controls**: Admins can add, update, and delete movie and user data, track changes, and generate logs.
* **Trending Insights**: Highlights trending movies, genres, actors, and directors.

### 2.3 User Classes and Characteristics

AtoZflix caters to multiple user types:

1. **General Users**: Movie enthusiasts who use the platform for discovery, ratings, and reviews.
   * Characteristics: Diverse age group, primarily tech-savvy, interest in personalized recommendations.
2. **Administrators**: Responsible for managing database integrity and user activities.
   * Characteristics: Experienced in database operations, need access to admin controls and logs.

### 2.4 Operating Environment

 **Client Devices**: Desktop, laptop, tablet, and mobile devices with modern web browsers.

 **Supported Browsers**: Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.

 **Server Requirements**: Flask application server hosted locally or on cloud platforms like AWS or Heroku.

###  Database: SQLite with integration options for other relational databases like MySQL if scaling is needed

**2.4 Operating Environment**

* **Client Devices**: Desktop, laptop, tablet, and mobile devices with modern web browsers.
* **Supported Browsers**: Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge.
* **Server Requirements**: Flask application server hosted locally or on cloud platforms like AWS or Heroku.
* **Database**: SQLite with integration options for other relational databases like MySQL if scaling is needed.

**2.5 Design and Implementation Constraints**

**2.5.1 Technical Constraints**

* Use of SQLite limits concurrent access to data.
* Integration with third-party APIs depends on their availability and rate limits.
* Application must ensure compatibility with multiple screen sizes and devices.

**2.5.2 Security Constraints**

* Secure storage of sensitive user data (e.g., passwords, preferences).
* Implementation of OAuth 2.0 for secure API integrations.

**2.5.3 Business Constraints**

* Budget restrictions may limit cloud hosting options.
* Compliance with copyright regulations for displaying movie metadata and posters.

**2.5.4 Integration Constraints**

* Ensuring seamless communication between the Svelte frontend and Flask backend via REST APIs.

**2.6 User Documentation**

AtoZflix will include the following user documentation:

* **User Guide**: Step-by-step instructions for registration, navigation, and using core features.
* **Admin Manual**: Detailed guidelines on database management, user tracking, and log analysis.
* **API Documentation**: Explains how third-party APIs are utilized within the system.
* **FAQ Section**: Addresses common user queries about the platform.

**2.7 Assumptions and Dependencies**

**2.7.1 Technical Dependencies**

* Stable internet connection for fetching movie data via APIs.
* Availability of required third-party APIs (TMDb, IMDb) without significant downtime.

**2.7.2 External Services**

* TMDb API for movie metadata and poster images.
* SMTP service for email-based notifications and password recovery.

**2.7.3 Assumptions**

* Users will provide accurate preferences during registration.
* API rate limits will not hinder real-time functionality.
* User base initially limited to a manageable size for SQLite.

### 2.8 Process Model

##### 2.8.1 Software Development Strategy

The software development process for this project followed the Agile methodology, specifically utilizing Scrum as the framework. This approach allowed for flexibility, iterative progress, and continuous feedback, which were essential for adapting to changing requirements and ensuring high-quality deliverables. Below are the key components of the process model:

1. **Iterative Development**

The project was divided into multiple iterations (sprints), each lasting 2-4 weeks. Each sprint focused on delivering a potentially shippable product increment, allowing for regular assessment and adaptation.

1. **Sprint Planning**

At the beginning of each sprint, a planning meeting was held to define the sprint goal and select user stories from the product backlog.

The team estimated the effort required for each user story using story points.

1. **Daily Stand-ups**

Short daily meetings (15 minutes) were conducted to discuss progress, challenges, and plans for the day.

This promoted transparency and quick identification of blockers.

1. **Development and Testing**

Development adhered to best practices, including code reviews, pair programming, and coding standards.

Automated testing (unit tests, integration tests) was implemented to ensure code quality and facilitate continuous integration.

1. **Sprint Review(Future)**

At the end of each sprint, a review meeting was held to demonstrate the completed work to stakeholders.

Feedback was gathered to inform future sprints and adjust the product backlog as necessary.

1. **Sprint Retrospective(Future)**

After the sprint review, a retrospective meeting was conducted to reflect on the sprint process.

The team discussed what went well, what could be improved, and actionable steps for the next sprint.

1. **Continuous Integration and Deployment**

A CI/CD pipeline was established to automate the build, testing, and deployment processes.

This ensured that code changes were integrated frequently and that the application was always in a deployable state.

1. **Documentation**

Documentation was maintained throughout the development process, including user stories, technical specifications, and API documentation.

This facilitated knowledge sharing and onboarding of new team members.

1. **Adaptability**

The process was flexible enough to accommodate changes in requirements, technology, and team dynamics.

Continuous learning and improvement were encouraged to enhance team performance and product quality.

### 2.9 Project Plan

##### 2.9.1 Timeline and Milestones

The project was executed following a well-defined timeline, with key phases and milestones meticulously tracked to ensure adherence to deadlines:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Phase** | **Duration** | **Start Date** | **End Date** | **Status** |
| Requirements Analysis | 2 weeks | 01/09/2024 | 15/09/2024 | Completed |
| System Design | 2 weeks | 15/09/2024 | 01/10/2024 | Completed |
| Development Phase 1 | 4 weeks | 05/10/2024 | 03/11/2024 | Completed |
| Development Phase 2 | 2 weeks | 04/11/2024 | 20/11/2024 | Completed |
| Testing | 1 week | 21/11/2024 | 28/11/2024 | Completed |
| Deployment | 1 week | 21/11/2024 | 28/11/2024 | Completed |

##### 2.9.2 Resource Allocation

Resource allocation was optimized to ensure efficiency across all project phases:

|  |  |  |
| --- | --- | --- |
| **Resource Type** | **Allocation** | **Status** |
| Frontend Developers | 2 | Assigned |
| Backend Developers | 3 | Assigned |

##### 2.9.3 Deliverables

All project deliverables were completed and verified to meet quality standards:

* Requirements Documentation
* System Design Documents
* Frontend Application
* Backend API
* Database Schema
* Test Reports
* User Documentation

### 2.10 Feasibility Report

##### 2.10.1 Technical Feasibility

1. All required technologies were readily available.
2. The team possessed the necessary technical expertise.
3. Infrastructure requirements were met successfully.
4. Integration with existing systems was achieved without issues.

##### 2.10.2 Economic Feasibility

|  |  |  |
| --- | --- | --- |
| **Cost**  **Category** | **Amount PKR** | **Status** |
| Development | N/A | Approved |
| Infrastructure | N/A | Approved |
| Maintenance | N/A | Approved |
| Training | N/A | Approved |

##### 2.10.3 Operational Feasibility

1. Users expressed willingness to adopt the system.
2. Training requirements were fulfilled effectively.
3. Business processes were adapted successfully.
4. Stakeholder support was consistently secured.

##### 2.12.4 Schedule Feasibility

1. The project timeline was realistic and adhered to.
2. Resources were available as required.
3. All milestones were achieved as planned.
4. Dependencies were managed efficiently.

##### 2.10.5 Risk Assessment

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk** | **Probability** | **Impact** | **Mitigation Strategy** | **Status** |
| Technical Issues | Medium | High | Regular code reviews, thorough testing | Managed |
| Resource Availability | Low | Mediu  m | Backup resources identified | Managed |
| Scope Creep | Medium | High | Clear requirements, change control | Managed |
| Integration Challenges | Low | High | Early testing, proper documentation | Managed |

### 2.11 Homogenization Process

##### 2.11.1 Code Standards

* Consistent naming conventions established.
* Code formatting rules defined and enforced.
* Documentation standards were implemented.
* Version control practices were standardized across teams.

##### 2.11.2 Architecture Standardization

1. **Database Standardization**
   1. Naming conventions for tables and columns were enforced.
   2. Standardized data types were defined.
   3. Indexing strategies were implemented.
   4. Foreign key relationships were standardized.
2. **API Standardization**
   1. RESTful API principles were followed.
   2. Versioning practices were established.
   3. Consistent response formats were defined.

##### 2.11.3 Testing Standards

1. Unit testing requirements were met.
2. Integration testing procedures were established and followed.
3. Performance testing benchmarks were achieved.
4. Code coverage metrics were consistently met.

##### 2.11.4 Documentation Standards

1. API documentation format was standardized.
2. Code commenting requirements were defined.
3. README file structure and change log format were implemented.

##### 2.11.5 Quality Assurance Process

1. Comprehensive code review checklists were utilized.
2. Testing procedures ensured high-quality deliverables. 3. Performance benchmarks were rigorously monitored.

4. Security requirements were thoroughly met.

##### 2.11.6 Deployment Process

1. Standardized build and deployment processes were followed.
2. Environment configurations were documented.
3. Deployment checklists were utilized to minimize errors.
4. Rollback procedures were defined and tested.

### 2.12 Use case descriptions

##### 2.12.1 Use Case 1: Place Order

**Actor**: Student/Teacher

**Description**: A user places a new order from a shop. **References**: controllers/paymentControllers.js

**Pre-conditions:**

* User is authenticated.
* User has the role of 'student' or 'teacher'.
* If the user is a student, their alert count must be less than 3.

**Main Flow:**

1. User selects items from the shop menu.
2. System validates item availability and shop ID.
3. System calculates the total price.
4. System creates an order record.
5. System creates records for order items.
6. System sends a real-time notification.
7. System returns order confirmation.

**Alternative Flows:**

* **Invalid Items**: System returns an error message.
* **Student with 3+ Alerts**: System blocks the order.
* **Items Not Found**: System returns an error message.

##### 2.12.2 Use Case 2: Process Payment

**Actor**: Student/Teacher

**References**: controllers/paymentControllers.js

**Pre-conditions:**

* Order exists.
* User is authenticated.
* Payment amount matches the order total.

**Main Flow:**

1. User uploads a payment screenshot.
2. System verifies payment details.
3. System updates the order status.
4. System updates the payment status.
5. System updates the shop's revenue records.
6. System sends a payment confirmation email.

Alternative Flows:

* **Invalid Payment Amount**: System rejects the payment.
* **Screenshot Verification Fails**: System marks the payment as pending.
* **System Error**: System rolls back the transaction.

##### 2.12.3 Use Case 3: Shop Dashboard

**Actor**: Shop Owner **References**: Shop Dashboard

**Pre-conditions:**

● User is authenticated as a shop owner. ● The shop exists and is active.

**Main Flow**:

1. System fetches shop details.
2. System calculates top-selling items.
3. System retrieves recent orders.
4. System calculates revenue metrics.
5. System generates customer insights.
6. System displays dashboard data.

**Alternative Flows:**

* **No Sales Data**: System displays empty statistics.
* **System Error**: System shows an error message.

##### 2.12.4 Use Case 4: Order Management

**Actor**: Shop Owner

**References**: controllers/ordersControllers.js **Pre-conditions:**

● User is authenticated as a shop owner. ● Orders exist for the shop.

**Main Flow:**

1. System displays a list of orders.
2. Owner views order details.
3. Owner updates the order status.
4. System sends status update emails.
5. System updates order records.

**Alternative Flows:**

* **No Orders Found**: System displays an empty list.
* **Invalid Status Update**: System shows an error message.

##### 2.12.5 Use Case 5: Audit Logging

**Actor**: System

**References**: Audit Tables

**Pre-conditions:**

* Database triggers are active.
* An action is performed on monitored tables.

**Main Flow:**

1. System captures the change event.
2. System records the old data state.
3. System records the new data state.
4. System logs the user performing the change.
5. System timestamps the action.

**Alternative Flows:**

* **Trigger Failure**: System logs an error message.
* **Invalid Data Format**: System skips the logging process.

##### 2.12.6 Use Case 6: Revenue Tracking

**Actor**: Shop Owner

**References**: Revenue Tracking Module

**Pre-conditions:**

● The shop has active orders. ● Payment records exist.

**Main Flow:**

1. System calculates total sales.
2. System identifies top products.
3. System generates revenue reports.
4. System tracks sales metrics.
5. System displays revenue analytics.

**Alternative Flows:**

* **No Sales Data**: System displays zero metrics.
* **Calculation Error**: System displays an error message.

#### 2.13 Video Link

Link

### ****3. External Interface Requirements****

#### ****3.1 User Interfaces****

The user interface (UI) is designed to be intuitive, responsive, and visually appealing:

* **Login/Registration Pages**: Allow users to create accounts and manage profiles.
* **Home Page**: Displays trending movies, personalized recommendations, and search functionality.
* **Watchlist Page**: Enables users to view and manage their saved movies.
* **Admin Dashboard**: Provides admins with tools to manage movies, users, and analytics.

The UI uses a dark theme with shades of black (#121212) for the background and contrasting light text for readability.

#### ****3.2 Hardware Interfaces****

The hardware requirements for running AtoZflix include:

* **Client Devices**:
  + Minimum: Devices with 2GB RAM and modern browsers.
  + Recommended: Devices with 4GB RAM or more for smoother performance.
* **Server Requirements**:
  + Minimum: Dual-core processor, 4GB RAM, 20GB storage.
  + Recommended: Quad-core processor, 8GB RAM, 50GB storage.
* **Database Server**: Hosts the SQLite database, requiring minimal storage and processing power.

#### ****3.3 Software Interfaces****

* **Frontend**: Built with Svelte for responsiveness and smooth performance.
* **Backend**: Flask-based REST APIs for business logic and database operations.
* **Database**: SQLite for storing and managing user, movie, and interaction data.
* **Third-Party APIs**:
  + TMDb API: Fetch movie details, posters, and metadata.
  + IMDb API: Retrieve ratings and additional movie insights.

#### ****3.4 Communication Interfaces****

* **Email Notifications**: Used for account verification and password recovery.
* **REST API Endpoints**: Facilitates communication between the frontend and backend.
* **Web Sockets**: Enables real-time updates for notifications or trending insights.
* **Network Communication**: Uses HTTPS for secure data transfer between the client and server.

### ****4. System Features****

#### ****4.1 System Feature 1: Personalized Recommendations****

* **Description**: Suggests movies based on user preferences, past interactions, and trending data.
* **Functional Requirements**:
  1. Fetch user data and preferences from the database.
  2. Apply collaborative and content-based filtering algorithms.
  3. Display recommendations on the user’s dashboard.

#### ****4.2 System Feature 2: Watchlist Management****

* **Description**: Allows users to save and organize movies they want to watch later.
* **Functional Requirements**:
  1. Enable users to add/remove movies from the watchlist.
  2. Synchronize the watchlist across devices for the same user.
  3. Provide filters for sorting movies within the watchlist.

#### ****4.3 System Feature 3: Admin Dashboard****

* **Description**: A control panel for managing movies, users, and logs.
* **Functional Requirements**:
  1. Add, update, or delete movie records.
  2. View and manage registered users.
  3. Access audit logs and generate reports for system performance.

#### ****4.4 System Feature 4: User Ratings and Reviews****

* **Description**: Allows users to rate movies and provide feedback.
* **Functional Requirements**:
  1. Accept and store user ratings and reviews in the database.
  2. Aggregate ratings to calculate average scores for movies.
  3. Use ratings to refine recommendation algorithms.

### ****5. Other Nonfunctional Requirements****

#### ****5.1 Performance Requirements****

* System should handle up to 1,000 concurrent users with no significant degradation in response time (<2 seconds).
* Ensure API response times are below 500ms under normal load.
* Database queries must execute within 1 second for optimal user experience.

#### ****5.2 Safety Requirements****

* Back up database daily to prevent data loss.
* Ensure the system can recover from critical errors without compromising user data.

#### ****5.3 Security Requirements****

* Use HTTPS for all client-server communication.
* Encrypt sensitive user data (e.g., passwords) using industry-standard algorithms (e.g., SHA-256).
* Implement role-based access control (RBAC) for admin functionalities.

#### ****5.4 Software Quality Attributes****

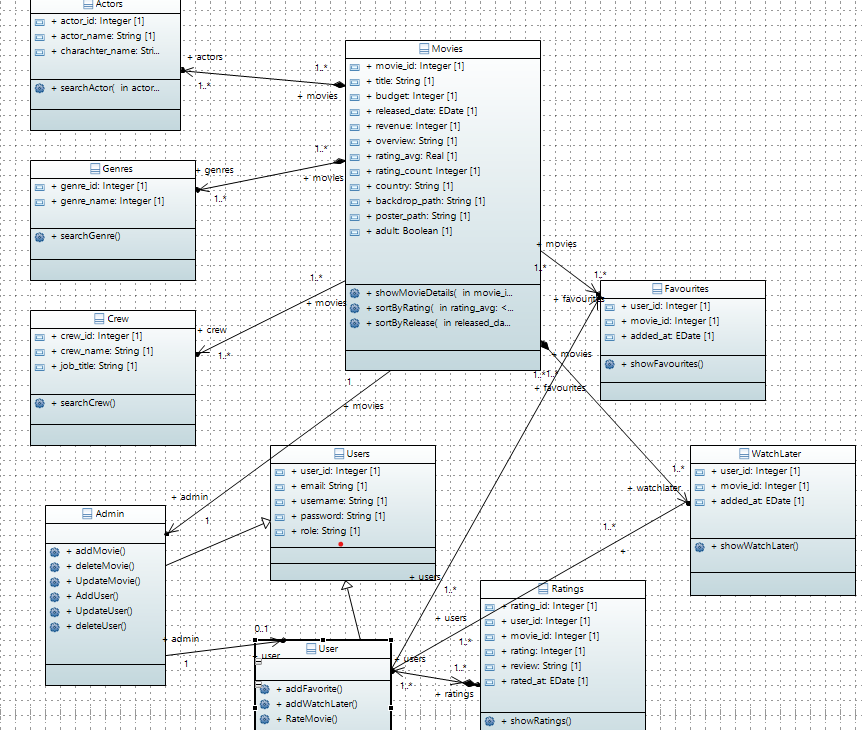
* **Usability**: Simple and intuitive user interface.
* **Reliability**: Ensure system uptime of 99.9%.
* **Maintainability**: Modular design for easy updates and debugging.
* **Scalability**: Ability to transition to a relational database like MySQL if user base grows.

#### ****5.5 Business Rules****

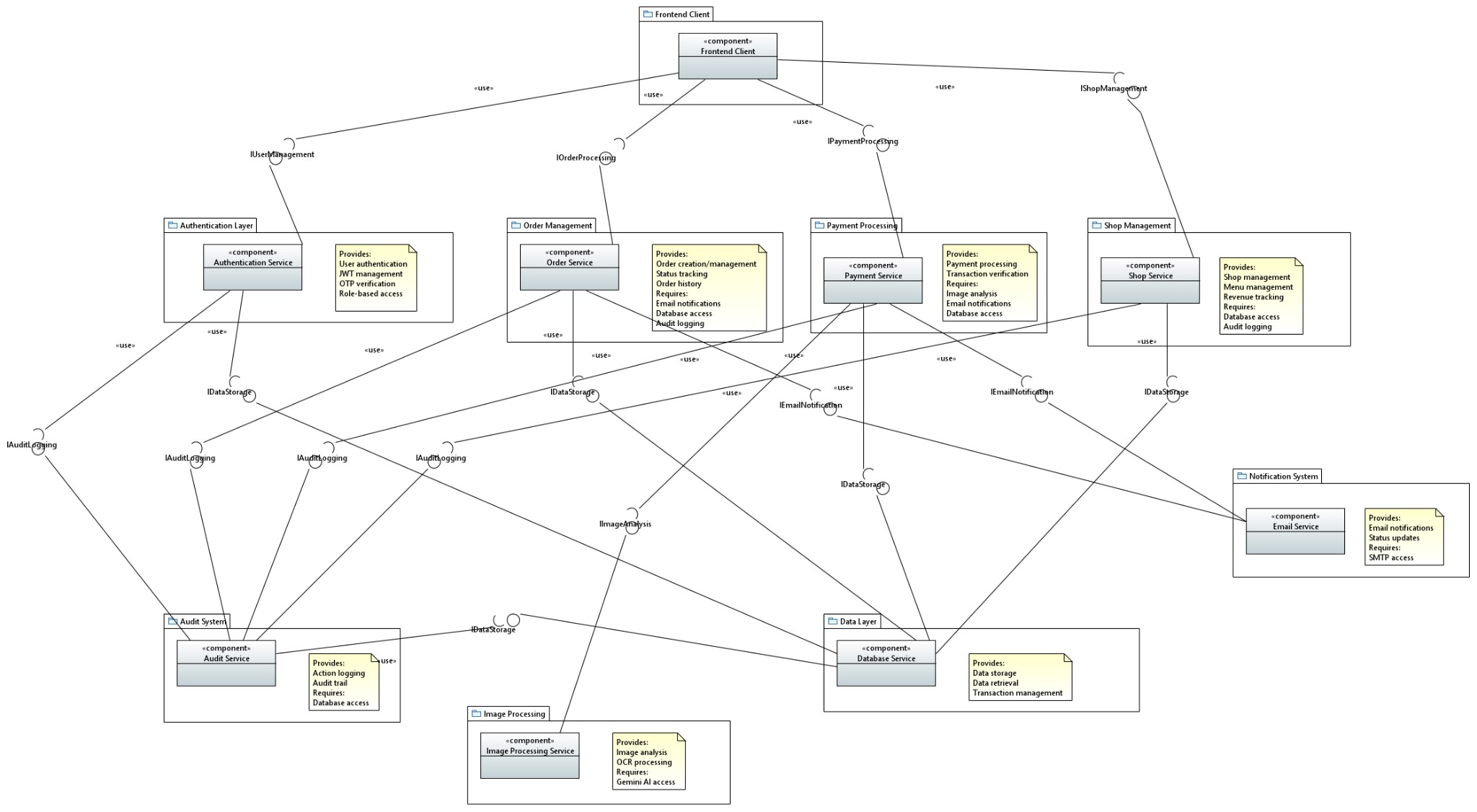
* Users must register to access personalized recommendations and watchlist features.
* Admins are the only users allowed to add or remove movies from the database.
* Ratings and reviews should be tied to unique user IDs to prevent duplication or manipulation.

# **Appendix B:Analysis Models**

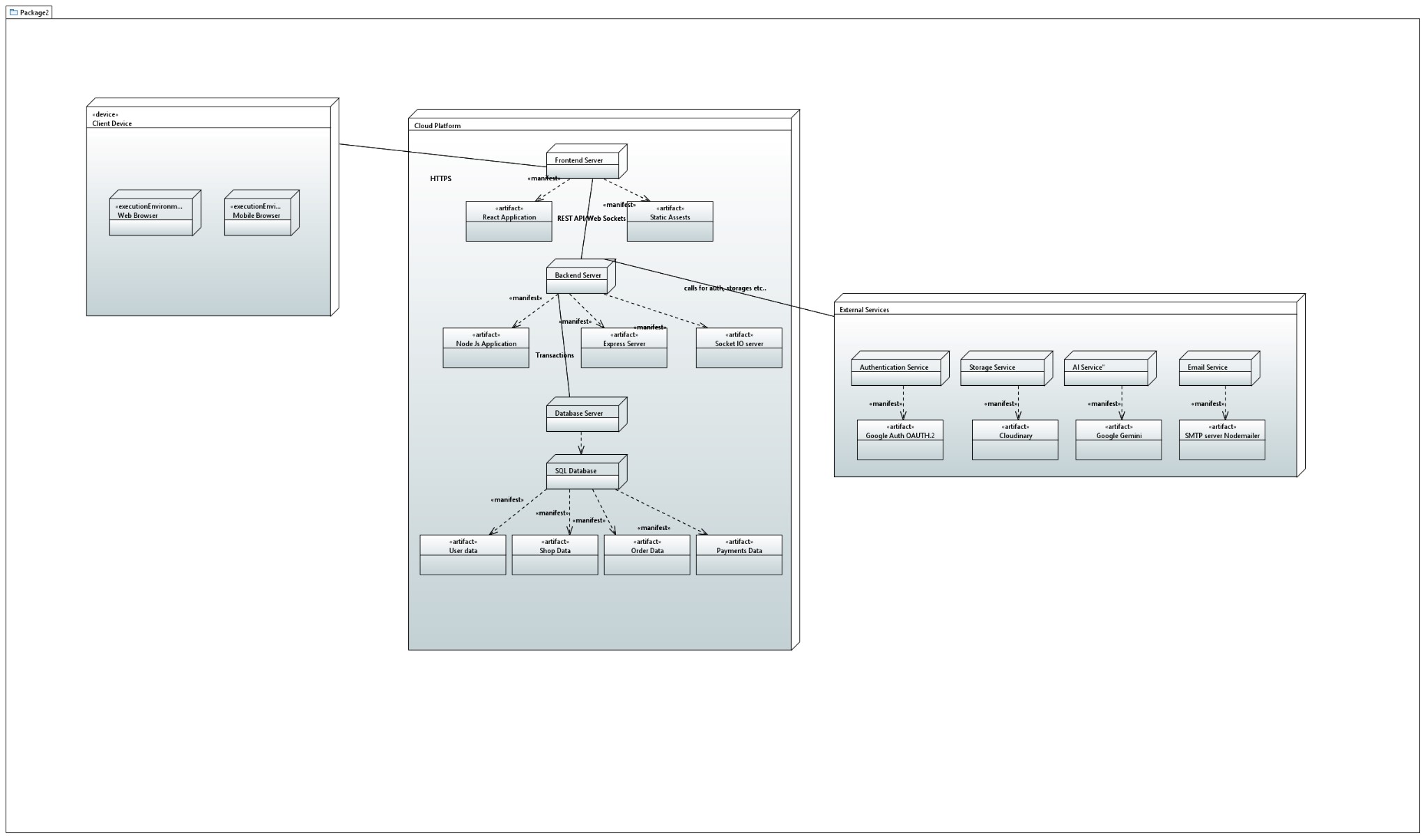
## 1. [Class](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889777/1_Class_Diagram_a0o8iy.png) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889777/1_Class_Diagram_a0o8iy.png)



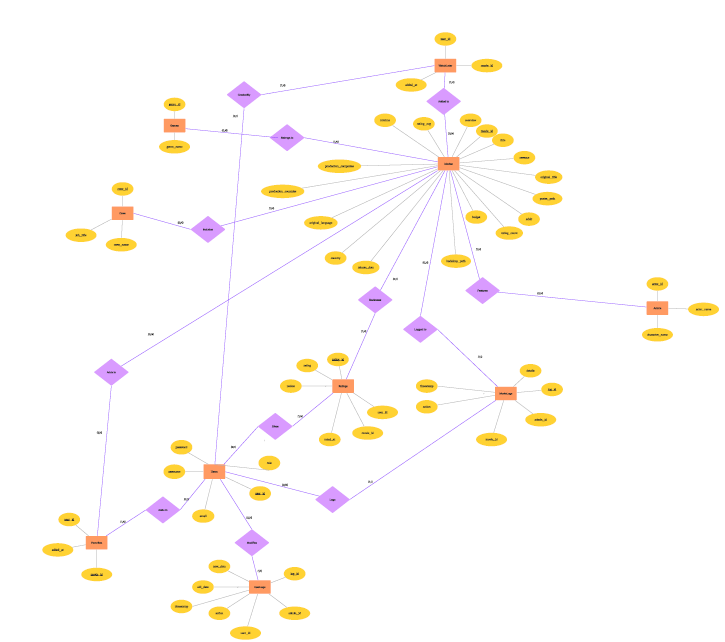
## 2. [Component](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889777/2_Component_Diagram_euncxi.png) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889777/2_Component_Diagram_euncxi.png)



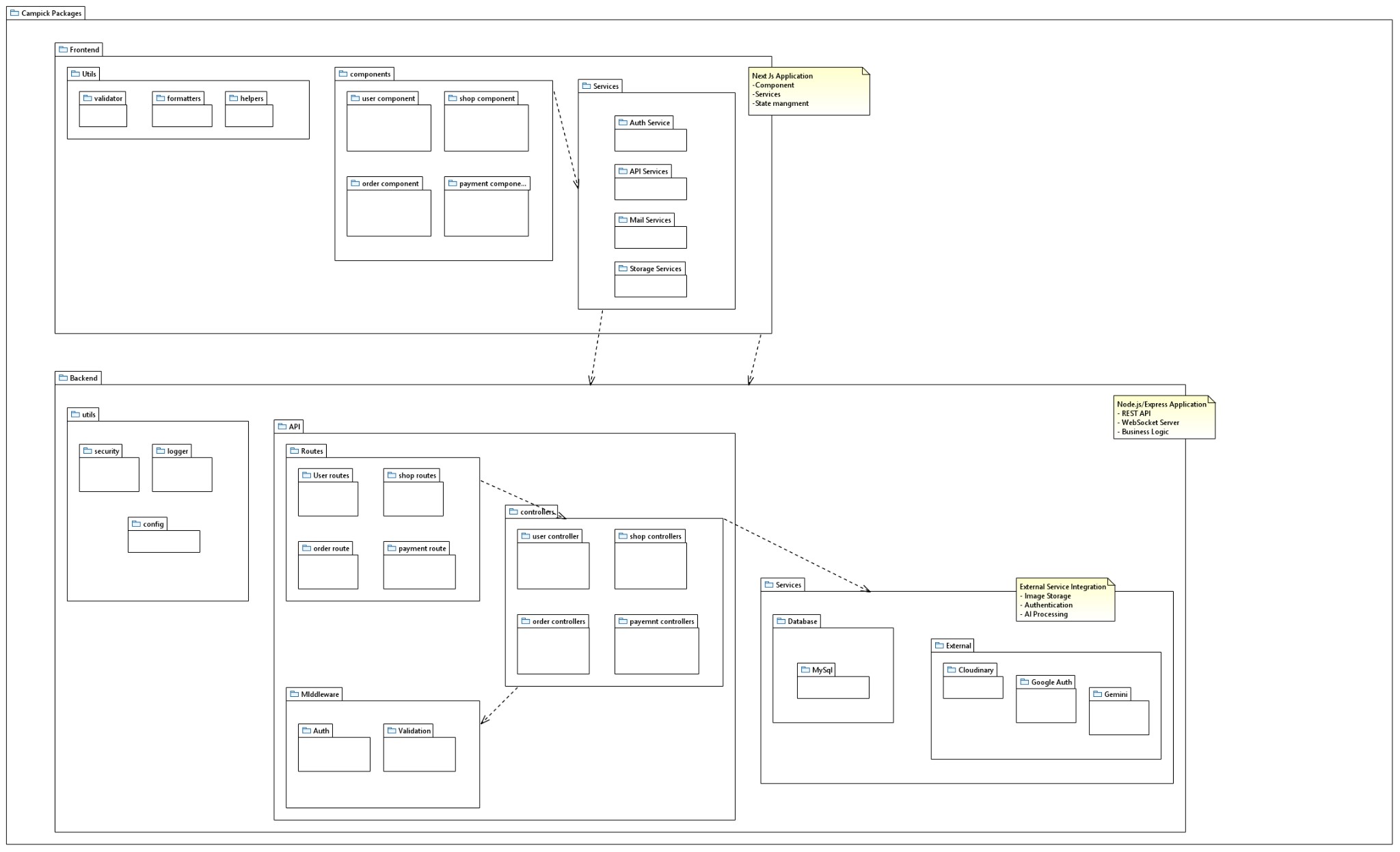
## 3. [Deployment](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889777/3_Deployment_Diagram_rdibew.png) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889777/3_Deployment_Diagram_rdibew.png)



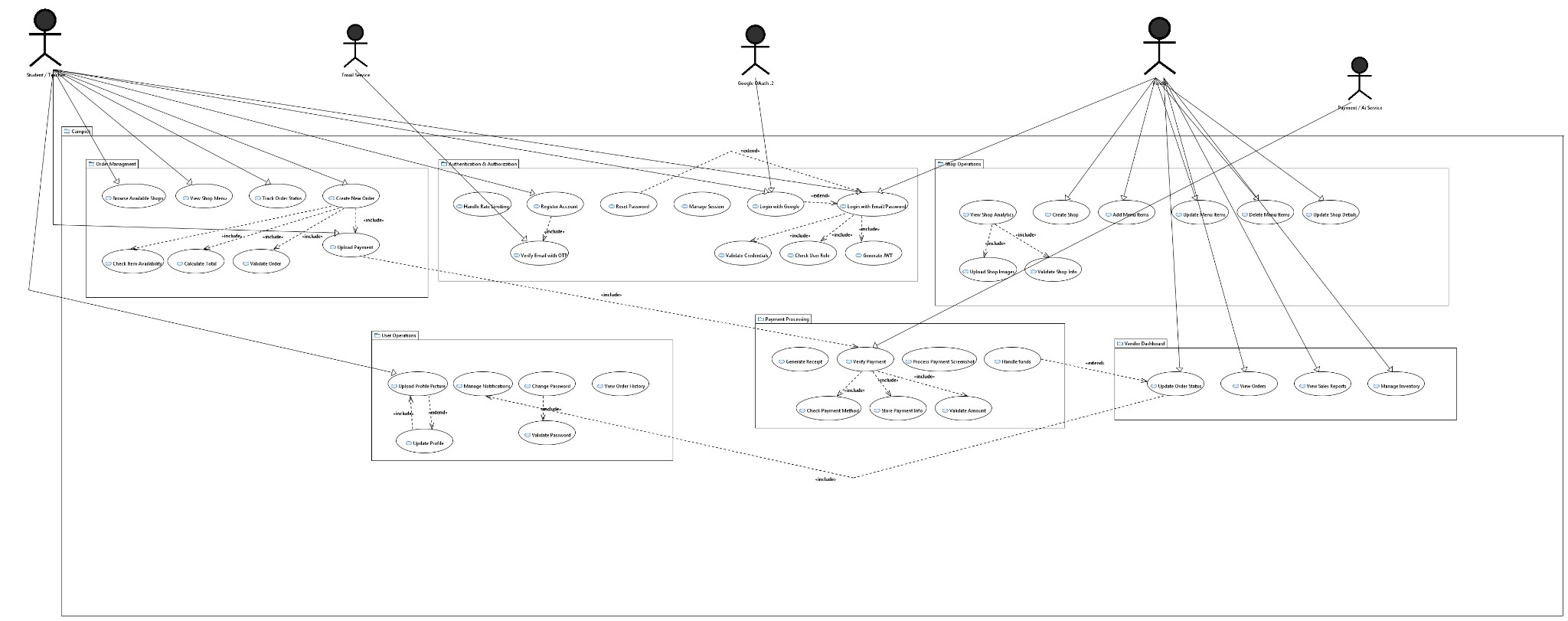
## 4. [Entity](https://res.cloudinary.com/dlthjlibc/image/upload/v1732893495/Campick_-_f_1_kh1zbd.jpg) [Relation](https://res.cloudinary.com/dlthjlibc/image/upload/v1732893495/Campick_-_f_1_kh1zbd.jpg) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732893495/Campick_-_f_1_kh1zbd.jpg)



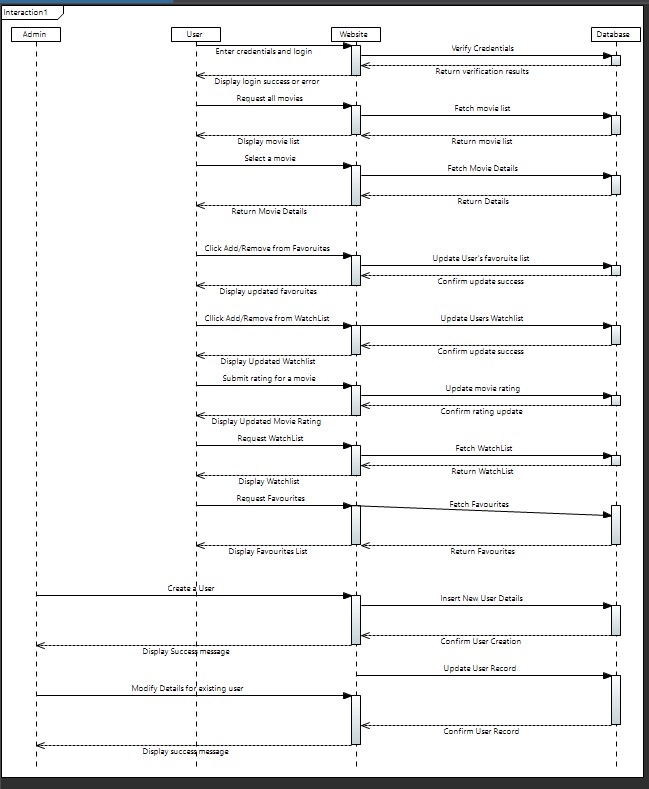
## 5. [Package](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889777/5_Package_Diagram_arpu5n.png) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889777/5_Package_Diagram_arpu5n.png)



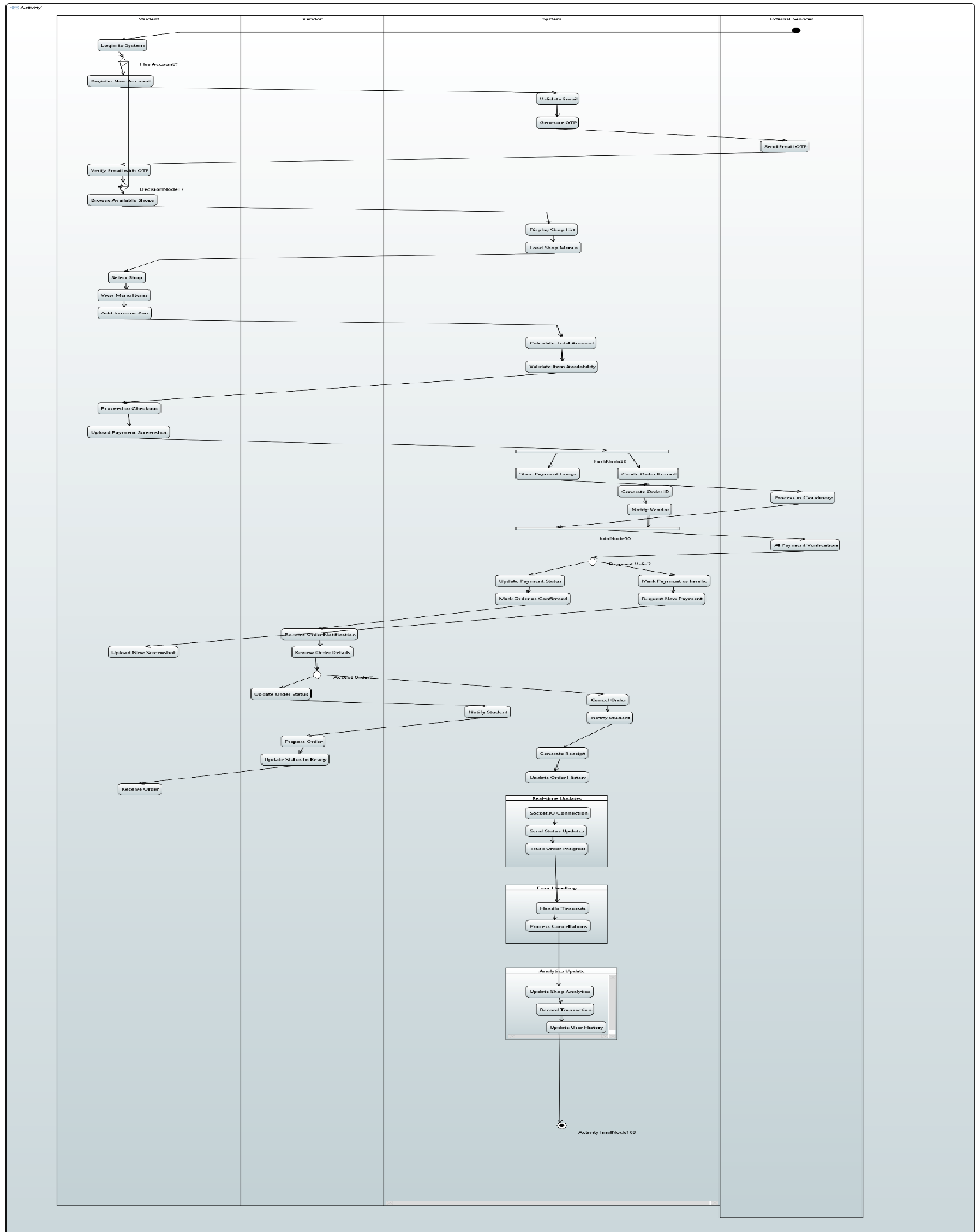
## 6. [Use](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889778/6_Use_Case_Diagram_sgfghr.png) [Case](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889778/6_Use_Case_Diagram_sgfghr.png) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889778/6_Use_Case_Diagram_sgfghr.png)



## 7. [Sequence](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889778/7_Sequence_Diagram_aspswv.png) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889778/7_Sequence_Diagram_aspswv.png)



## 8. [Activity](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889779/8_Activity_Diagram_prk2zr.png) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889779/8_Activity_Diagram_prk2zr.png)



9.[Collaboration](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889779/14_Collab_dbhrzk.png) [Diagram](https://res.cloudinary.com/dlthjlibc/image/upload/v1732889779/14_Collab_dbhrzk.png)

