Software Requirement Specifications

for

Campick

Prepared by

Hafiz M Abdullah 22k-4489

Asghar Ali 22k-4415

M Bilal 22k-4242

Table Of Contents

Software Requirements Specification - Campick NUCES

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a detailed description of Campick, the Campus Food Ordering System (CFOS). This document serves as a comprehensive guide for developers, project managers, and stakeholders, outlining the system's intended features, functionalities, and constraints. It aims to ensure a shared understanding of the system's requirements and facilitate effective communication among all parties involved in the project.

1.2 Scope

The CFOS aims to revolutionize campus food ordering through:

1.2.1 Core Functionalities

1. Comprehensive digital menu management for vendors.
2. Real-time order tracking to enhance user experience.
3. Secure payment processing to protect user data.
4. Automated order notifications to keep users informed.
5. Vendor analytics dashboard for performance insights.

1.2.2 Business Objectives

1. Reduce order processing time by 75% to improve efficiency.
2. Enhance vendor revenue tracking for better financial management.
3. Improve the overall student dining experience through streamlined processes.
4. Enable data-driven business decisions for vendors.

1.2.3 Target Benefits

1. Streamlined ordering process to minimize user effort.
2. Reduced wait times for food delivery.
3. Improved order accuracy to enhance customer satisfaction.
4. Enhanced financial tracking for better resource allocation.
5. Better resource management for vendors.

1.3 Document Conventions

1.3.1 Priority Levels

1. **Critical**: Features that are essential for basic system functionality and must be implemented.
2. **High**: Important features that are necessary for the system's initial launch.
3. **Medium**: Features that enhance the system but can be implemented after the launch.
4. **Low**: Features that are desirable for future enhancements and improvements.

1.3.2 Technical Conventions

1. Code adheres to ES6+ JavaScript standards for consistency and modern practices.
2. RESTful API endpoints are structured using kebab-case for clarity.
3. Database columns are named using snake_case for readability.
4. Component names follow PascalCase to align with industry standards.
5. Function names utilize camelCase for consistency in coding style.

1.4 References

1.4.1 Technical Documentation

1. Node.js v16+ Documentation for backend development.
2. Express.js 4.x Documentation for server-side framework guidance.
3. MySQL 8.0 Reference Manual for database management.
4. Socket.IO v4 Documentation for real-time communication.

1.4.2 API Documentation

1. Google OAuth 2.0 Documentation for authentication processes.
2. Cloudinary API Documentation for image management.
3. Gemini API Documentation for payment validation.

1.4.3 Standards

1. JWT RFC 7519 for secure token handling.
2. REST API Design Standards for consistent API development.
3. OWASP Security Guidelines for best practices in security.

2. Overall Description

2.1 Product Perspective

CFOS is a comprehensive system with the following architecture:

2.1.1 Frontend Layer

1. A user-friendly interface built on Next.js for an engaging user experience.
2. Responsive design to accommodate various devices and screen sizes.
3. Real-time updates facilitated by WebSocket technology.

2.1.2 Backend Services

1. A robust Node.js/Express.js server for handling requests and responses.

2. RESTful API endpoints for efficient data communication.
3. A WebSocket server for real-time communication and updates.

2.1.3 External Integrations

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

2.2 Product Functions

2.2.1 User Authentication

1. Email/password registration for user accounts.
2. Google OAuth integration for seamless login.
3. OTP verification for enhanced security.
4. Role-based access control to manage user permissions.
5. Session management for user state maintenance.

2.2.2 Shop Management

1. Shop profile creation for vendors.
2. Menu management for easy updates and changes.
3. Operating hours management for vendor availability.
4. Order processing for efficient handling of customer requests.
5. Analytics dashboard for performance tracking.

2.2.3 Order System

1. Cart management for user convenience.
2. Order placement for initiating transactions.
3. Real-time tracking for order status updates.
4. Order history for user reference.
5. Status updates to keep users informed.

2.2.4 Payment System

1. Payment screenshot upload for verification.
2. AI-powered verification for secure transactions.
3. Payment status tracking for user transparency.
4. Transaction history for user records.
5. Refund processing for customer satisfaction.

2.2.5 Analytics & Reporting

1. Sales reports for vendor insights.
2. Popular items tracking for inventory management.
3. Revenue analytics for financial oversight.
4. Customer insights for targeted marketing.

5. Performance metrics for operational improvements.

2.3 User Classes and Characteristics

2.3.1 Students

1. Primary system users with the following features:
    a. Order placement for food items.
    b. Payment management for transactions.
    c. Order tracking for real-time updates.
    d. History viewing for past orders.
2. Characteristics:
    a. Tech-savvy and familiar with online platforms.
    b. Frequent users of the system.
    c. Budget-conscious and value-driven.
    d. Time-sensitive and require quick service.

2.3.2 Teachers

1. Secondary system users with similar features to students:
    a. Higher priority orders for time-sensitive needs.
2. Characteristics:
    a. Less frequent users compared to students.
    b. Higher average order value due to larger meal requirements.
    c. Time-constrained and requires efficient service.

2.3.3 Shop Owners

1. Business users with the following features:
    a. Menu management for shop offerings.
    b. Order processing for customer requests.
    c. Analytics access for performance insights.
    d. Payment management for financial transactions.
2. Characteristics:
    a. Daily system usage for operational efficiency.
    b. Business-focused with a need for reliable service.
    c. Require technical support for system issues.

2.4 Operating Environment

1. Node.js runtime environment for backend services.
2. MySQL database for data storage and management.
3. Cloud hosting platform for scalability and reliability.
4. Modern web browsers for user access.
5. Mobile-responsive design for accessibility.
6. Secure HTTPS protocol for data protection.

2.5 Design and Implementation Constraints

2.5.1 Technical Constraints

1. Node.js version 16+ is required for compatibility.
2. MySQL 8.0+ database for data management.
3. HTTPS protocol is mandatory for secure communications.

4. Maximum file upload size is limited to 5MB.
5. API rate limiting is set to 100 requests per minute.
6. WebSocket connection limits to ensure performance.

2.5.2 Security Constraints

1. Email domain is restricted to nu.edu.pk for user verification.
2. JWT expiration is set to 24 hours for security.
3. Password requirements include:
    a. Minimum of 8 characters.
    b. Must contain numbers and special characters.
    c. Case sensitivity is required.
4. OTP validity is limited to 10 minutes for security.

2.5.3 Business Constraints

1. One shop is allowed per vendor for operational simplicity.
2. Maximum order value limits to prevent abuse.
3. Payment verification timeout is set to 30 minutes.
4. Order cancellation time limit is 5 minutes post-placement.
5. Menu item limits per shop to ensure manageability.

2.5.4 Integration Constraints

1. Cloudinary free tier limitations for image storage.
2. Google OAuth quotas for authentication requests.
3. SMTP daily email limits for notifications.
4. Gemini API request limits for payment processing.

2.7 Assumptions and Dependencies

2.7.1 Technical Dependencies

```
{
  "dependencies": {
    "node": ">=16.0.0",
    "express": "^4.18.0",
    "mysql2": "^3.0.0",
    "socket.io": "^4.0.0",
    "jsonwebtoken": "^9.0.0",
    "bcrypt": "^5.0.0",
    "cloudinary": "^1.0.0",
    "@google/generative-ai": "^1.0.0"
  }
}
```

2.7.2 External Services

1. **Google Cloud Platform**
    a. OAuth 2.0 for authentication.
    b. Gemini AI API for payment validation.
2. **Cloudinary**
    a. Image storage and management.
    b. Transformation API for image processing.
3. **SMTP Service**

a. Email delivery for notifications.
b. Templates for user communication.

2.7.3 Assumptions

1. Users have stable internet access for system functionality.
2. Vendors maintain accurate menu information for user satisfaction.
3. Users have access to smartphones or computers for ordering.
4. Payment screenshots are legible for verification.
5. The university email system is reliable for communication.

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.
2. **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.
3. **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.
4. **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.
5. **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.
6. **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.

7. **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.
8. **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.
9. **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	Medium	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**
    a. Naming conventions for tables and columns were enforced.
    b. Standardized data types were defined.
    c. Indexing strategies were implemented.
    d. Foreign key relationships were standardized.
2. **API Standardization**
    a. RESTful API principles were followed.
    b. Versioning practices were established.
    c. 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. UML Concepts and Associations

3.1 UML Associations

- **Dual Associations**: Represent relationships between two different classes, such as `User` and `Order`, where a user can place multiple orders.
- **Self-Associations**: Used in cases where a class relates to itself, such as a `Shop` that can have multiple `MenuItems`.

- **Aggregation**: Represents a "whole-part" relationship, such as a `Shop` containing multiple `MenuItems`.
- **Composition**: A stronger form of aggregation where the part cannot exist without the whole, such as `OrderItems` existing only within an `Order`.

3.2 Realizations & Dependencies

- **Interface Implementations**: The system uses interfaces for various components, such as `PaymentProcessor` for handling different payment methods.
- **Object Dependencies**: Objects like `Order` depend on `User` and `Shop` objects, indicating that an order cannot exist without a user and a shop.

3.3 Advanced UML Elements

- **Parameterized Classes**: Classes like `Order<T>` can be used to define orders for different types of items, allowing for flexibility in item types.
- **Enumerations**: Used for defining fixed sets of constants, such as `OrderStatus` with values like `pending`, `accepted`, `delivered`, etc.
- **Concurrent Objects**: The system may utilize concurrent objects for handling multiple user requests simultaneously, ensuring responsiveness.
- **Active Objects**: Objects that have their own thread of control, such as a `NotificationService` that sends notifications independently of user actions.
- **Subsystem Models**: The system can be divided into smaller modules, such as `UserManagement`, `OrderManagement`, and `PaymentProcessing`, each responsible for specific functionalities.

4. External Interface Requirements

4.1 User Interfaces

The user interface (UI) of the software product will be designed to ensure a seamless and intuitive experience for users. Below are the logical characteristics of the user interfaces:

4.1.1 General Characteristics

1. **Screen Layout:** The application will follow a responsive design to accommodate various screen sizes, including desktops, tablets, and mobile devices.
2. **Navigation**: A consistent navigation bar will be present on all screens, allowing users to access different sections of the application easily.

4.1.2 General Characteristics

1. Error messages will be displayed prominently in red text, with clear descriptions of the issue and suggested actions for resolution.
2. Messages will be concise and user-friendly, avoiding technical jargon.

4.1.3 Software Components Requiring User Interfaces

1. **User Authentication:** Login, registration, and password recovery screens.
2. **User Profile Management:** Interfaces for viewing and editing user profiles.

3. **Order Management:** Screens for creating, viewing, and managing orders.
4. **Shop Management:** Interfaces for shop owners to manage their shops and menu items.

4.2 Hardware Interfaces

4.2.1 Server Requirements

1. Minimum CPU: 2 cores for processing.
2. RAM: 4GB minimum for performance.
3. Storage: 20GB SSD for data management.
4. Network: 100Mbps minimum for connectivity.

4.2.2 Client Requirements

5. Modern web browser for user access.
6. Camera (for payment screenshots) for verification.
7. Minimum screen resolution: 320px width for usability.

4.2.3 Database Server

8. MySQL 8.0+ for data management.
9. 10GB initial storage for database needs.
10. Backup capability for data integrity.
11. Replication support for reliability.

4.3 Software Interfaces

The software product will connect with several external software components:

4.3.1 Database

1. **MySQL:** The application will use MySQL (version 8.0 or higher) for data storage and retrieval.
2. **Connection:** The application will utilize a connection pool to manage database connections efficiently.

4.3.2 Libraries and Tools

1. **Node.js:** The backend will be built using Node.js (version 14 or higher).
2. **Express.js:** The application will use Express.js (version 4.x) for routing and middleware.
3. **Nodemailer:** For sending emails, the application will utilize Nodemailer (version 6.x).

4.3.3 Data Items and Messages

1. **Incoming Data:** User registration details, order information, and payment data.
2. **Outgoing Data:** Confirmation emails, order status updates, and user notifications.

4.3.4 Data Items and Messages

1. The application will use RESTful APIs for communication between the frontend and backend, ensuring a clear separation of concerns and ease of integration.

4.4 Communication Interfaces

The software product will include several communication functions:

4.4.1 Email

The application will send emails for user registration, password recovery, and order confirmations using SMTP protocols.

4.4.2 Web Browser

The application will be accessible via modern web browsers (Chrome, Firefox, Safari, Edge) using HTTP/HTTPS protocols.

4.4.3 Network Server Communication

The application will communicate with the server using RESTful APIs over HTTPS, ensuring secure data transmission.

4.4.4 Message Formatting

JSON will be used for data exchange between the client and server, providing a lightweight and easy-to-parse format.

4.4.5 Communication Standards

1. **HTTP/HTTPS:** All communications will adhere to HTTP/HTTPS standards for secure data transfer.
2. **Security:** TLS/SSL encryption will be implemented to secure data in transit.

4.4.5 Data Transfer Rates and Synchronization

1. The application will be optimized for low-latency communication, ensuring quick response times for user actions.
2. Synchronization mechanisms will be implemented to handle concurrent data access and updates, particularly for order management and user profiles.

5. System Features

5.1 Authentication System

Priority: Critical

5.1.1 Description

A comprehensive authentication system supporting email/password and Google OAuth, with domain-restricted registration and OTP verification.

5.1.2 Functional Requirements

- **REQ-1**: Email domain verification (nu.edu.pk only)
- **REQ-2**: OTP verification via email
- **REQ-3**: Google OAuth integration

- **REQ-4**: JWT-based session management
- **REQ-5**: Password encryption and validation
- **REQ-6**: Role-based access control (student, teacher, shop_owner)
- **REQ-7**: Account verification status tracking

5.2 Order Management

Priority: Critical

5.2.1 Description

Real-time order processing system with status tracking and notifications.

5.2.2 Functional Requirements

- **REQ-8**: Order creation and validation
- **REQ-9**: Real-time status updates via WebSocket
- **REQ-10**: Order history tracking
- **REQ-11**: Multiple item support per order
- **REQ-12**: Order cancellation within time limit
- **REQ-13**: Status change notifications
- **REQ-14**: Price calculation and validation

5.3 Payment System

Priority: Critical

5.3.1 Description

AI-powered payment verification system using screenshot analysis and multiple payment methods.

5.3.2 Functional Requirements

- **REQ-15**: Payment screenshot upload and storage
- **REQ-16**: Gemini AI integration for verification
- **REQ-17**: Multiple payment method support
  - JazzCash
  - EasyPaisa
  - SadaPay
  - NayaPay
- **REQ-18**: Payment status tracking
- **REQ-19**: Automated verification process
- **REQ-20**: Manual verification fallback

5.4 Shop Management

Priority: High

5.4.1 Description

Comprehensive shop management system for vendors with menu and order control.

5.4.2 Functional Requirements

- **REQ-21**: Shop profile creation and management
- **REQ-22**: Menu item CRUD operations
- **REQ-23**: Order processing controls
- **REQ-24**: Revenue tracking and analytics
- **REQ-25**: Shop statistics dashboard
- **REQ-26**: Contact information management
- **REQ-27**: Payment method configuration

5.5 Analytics and Reporting

Priority: Medium

5.5.1 Description

Data-driven insights and reporting system for business intelligence.

5.5.2 Functional Requirements

- **REQ-28**: Sales analytics and trends
- **REQ-29**: Popular items tracking
- **REQ-30**: Customer order patterns
- **REQ-31**: Revenue analysis
  - Daily revenue
  - Monthly trends
  - Payment method distribution
- **REQ-32**: Performance metrics
- **REQ-33**: Custom report generation

5.6 Real-time Notification System

Priority: High

5.6.1 Description

Multi-channel notification system for order and payment updates.

5.6.2 Functional Requirements

- **REQ-34**: WebSocket real-time updates
- **REQ-35**: Email notifications
- **REQ-36**: Order status notifications
- **REQ-37**: Payment verification alerts
- **REQ-38**: System alerts and warnings
- **REQ-39**: Notification preferences

5.7 User Management

Priority: High

5.7.1 Description

User profile and preference management system with role-based features.

5.7.2 Functional Requirements

- **REQ-40**: Profile management
- **REQ-41**: Role-based access control
- **REQ-42**: Alert count tracking
- **REQ-43**: Order history access
- **REQ-44**: Payment history tracking
- **REQ-45**: Account verification status

5.8 Image Management

Priority: Medium

5.8.1 Description

Cloud-based image storage and processing system for menu items and payments.

5.8.2 Functional Requirements

- **REQ-46**: Cloudinary integration
- **REQ-47**: Image upload and storage
- **REQ-48**: Image optimization
- **REQ-49**: Payment screenshot processing
- **REQ-50**: Menu item image management
-

6. Other Nonfunctional Requirements

6.1 Performance Requirements

- Response time should be less than 2 seconds for user actions.
- Real-time updates should occur in less than 1 second.
- The system should maintain 99.9% uptime for reliability.
- Support for concurrent users to ensure scalability.

6.2 Security Requirements

- Data encryption to protect sensitive information.
- Secure authentication processes to prevent unauthorized access.
- Payment data protection to ensure user financial security.
- Role-based access control to manage user permissions effectively.
- Input validation to prevent security vulnerabilities.

6.3 Software Quality Attributes

- **Scalability**: The system should efficiently handle an increasing number of users and transactions without performance degradation. It should support horizontal scaling by adding more servers as needed.
- **Maintainability**: Code should be modular and well-documented to facilitate easy updates and bug fixes. The system should adhere to coding standards and best practices.
- **Reliability**: The system should be available 99.9% of the time, with minimal downtime for maintenance. It should have failover mechanisms in place to ensure continuous operation.
- **Usability**: The user interface should be intuitive and easy to navigate for all user classes. User feedback should be incorporated into design iterations.
- **Performance**: The system should respond to user actions within 2 seconds under normal load conditions. It should efficiently manage database queries and API calls.

6.4 Business Rules

- **Vendor Verification Process**: All vendors must undergo a verification process by the university before creating a shop profile. This includes background checks and compliance with university policies.
- **Payment Processing Rules**: Payments must be processed securely, and users should receive confirmation of payment status. Refunds must be initiated within 24 hours of a cancellation request.
- **Order Cancellation Policies**: Users can cancel orders within 5 minutes of placement. After this period, the order cannot be canceled, and the user will be charged.
- **User Rating System**: Users can rate their experience with vendors after order completion. Ratings will be visible to other users and will influence vendor visibility.
- **Refund Policies**: Refunds will be processed within 5-7 business days. Users must provide valid reasons for refunds, which will be reviewed by the admin.

7. Other Requirements

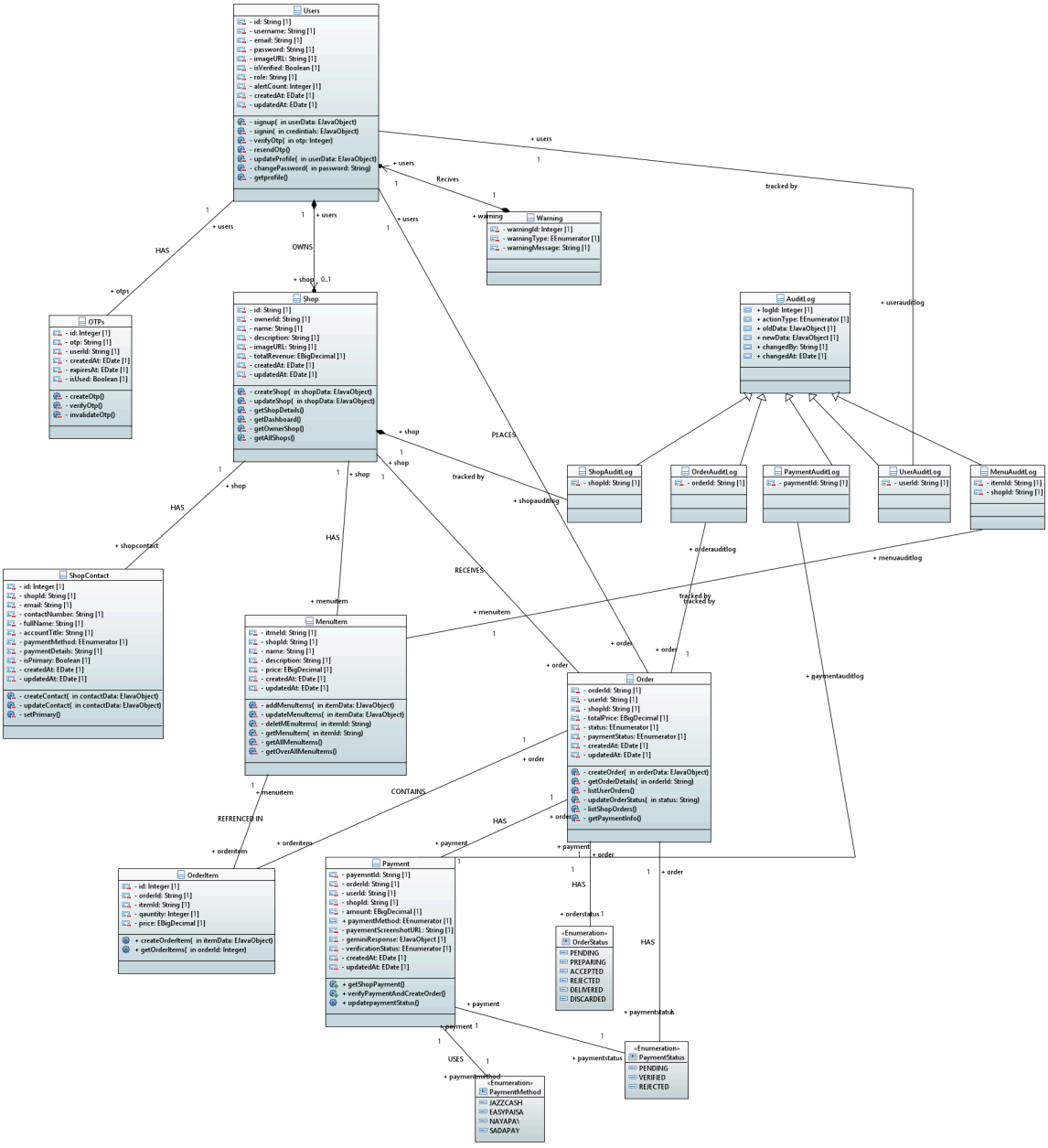
- **Backup and Recovery Procedures**: The system must implement daily backups of the database and critical files. Recovery procedures should be tested quarterly to ensure data integrity.
- **Data Retention Policies**: User data will be retained for a minimum of 5 years for compliance purposes. After this period, data will be anonymized or deleted based on user consent.
- **Compliance with University Regulations**: The system must adhere to all university policies regarding data privacy, user consent, and financial transactions.
- **Audit Trail Requirements**: All user actions, especially those involving financial transactions, must be logged for auditing purposes. Logs should be retained for at least 2 years.
- **System Monitoring and Logging**: The system should implement monitoring tools to track performance metrics and error rates. Alerts should be configured for critical failures.

Appendix A: Glossary

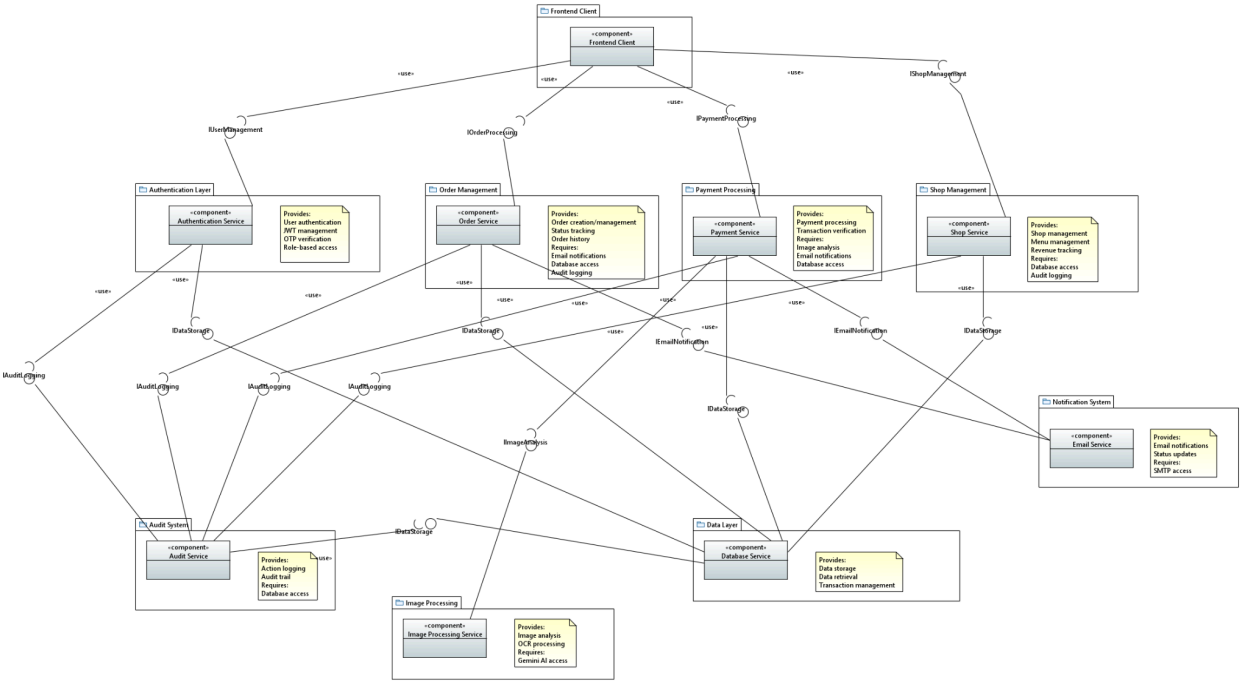
1. **CFOS**: Campus Food Ordering System
2. **JWT**: JSON Web Token
3. **API**: Application Programming Interface
4. **OTP**: One-Time Password
5. **UI**: User Interface
6. **UX**: User Experience

Appendix B:Analysis Models

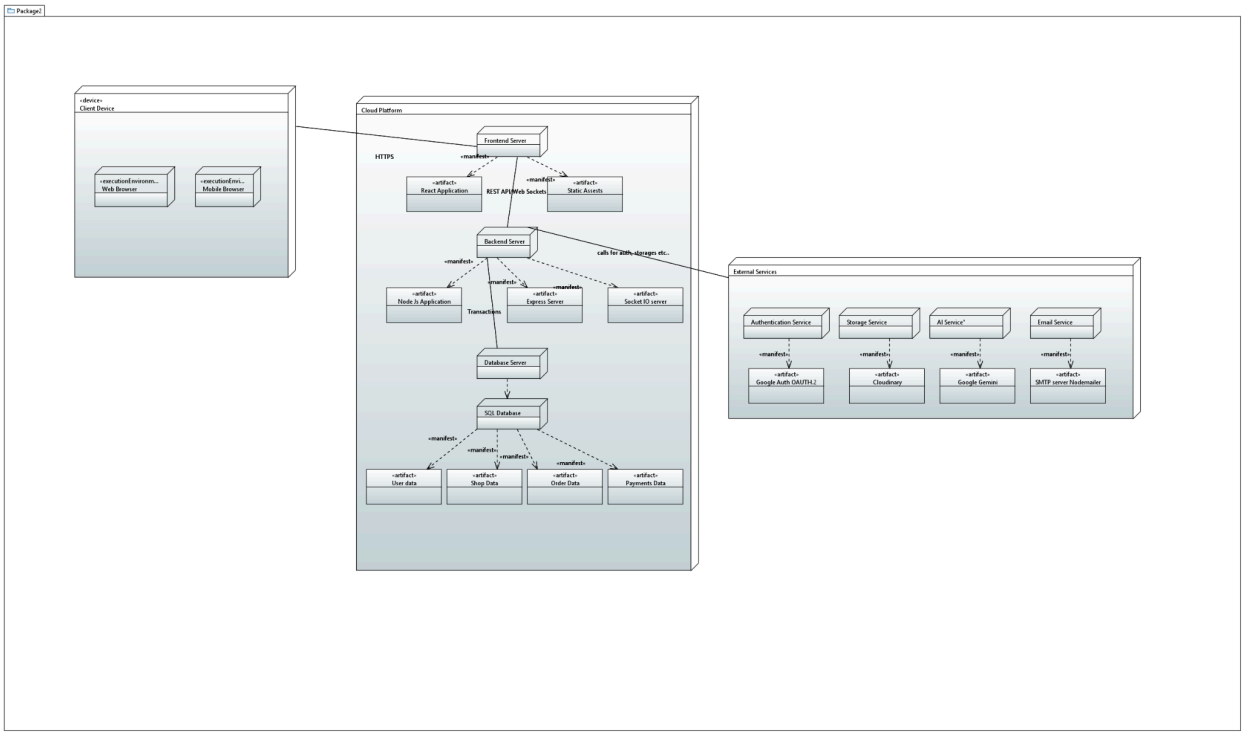
1. Class Diagram



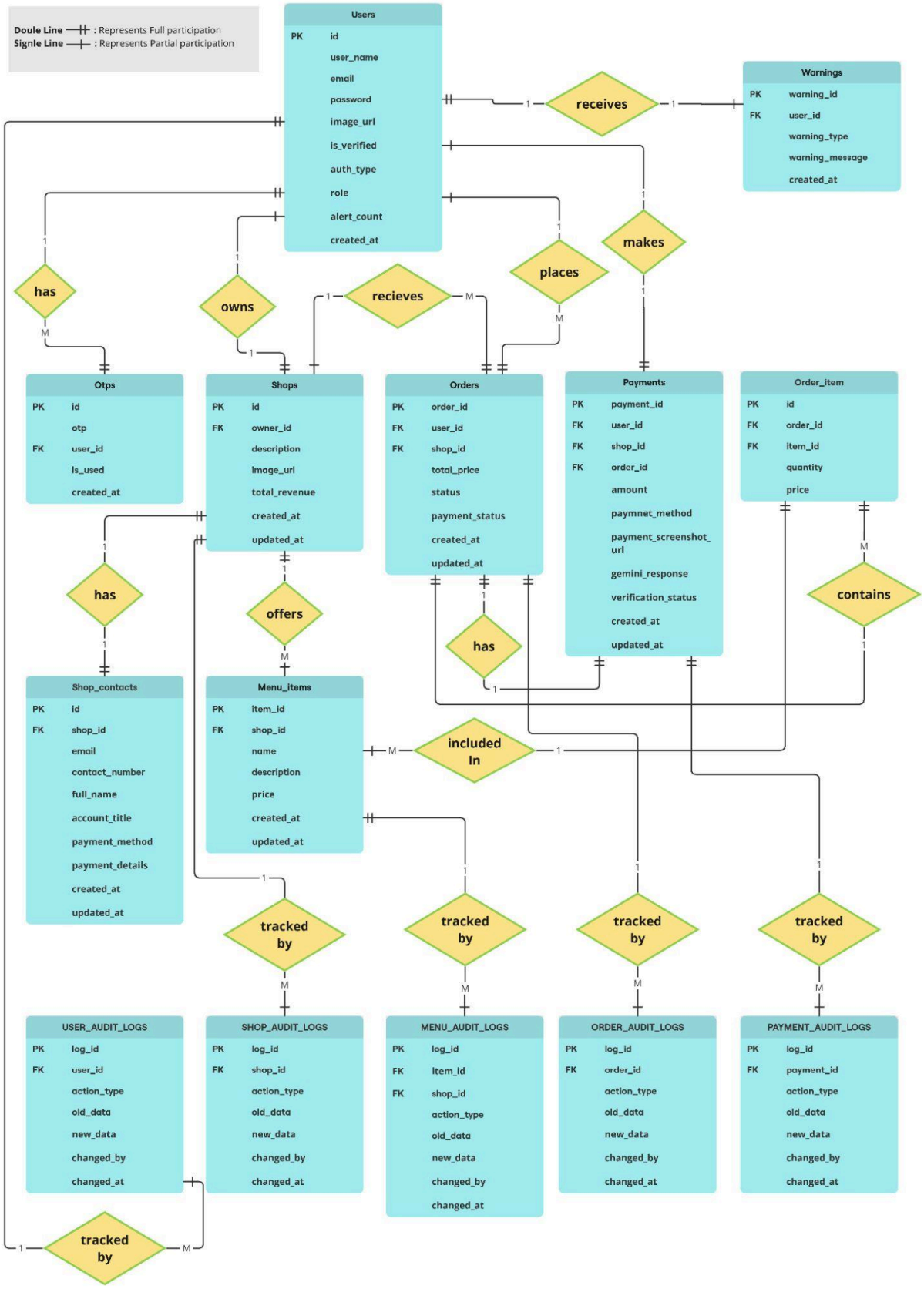
2. Component Diagram



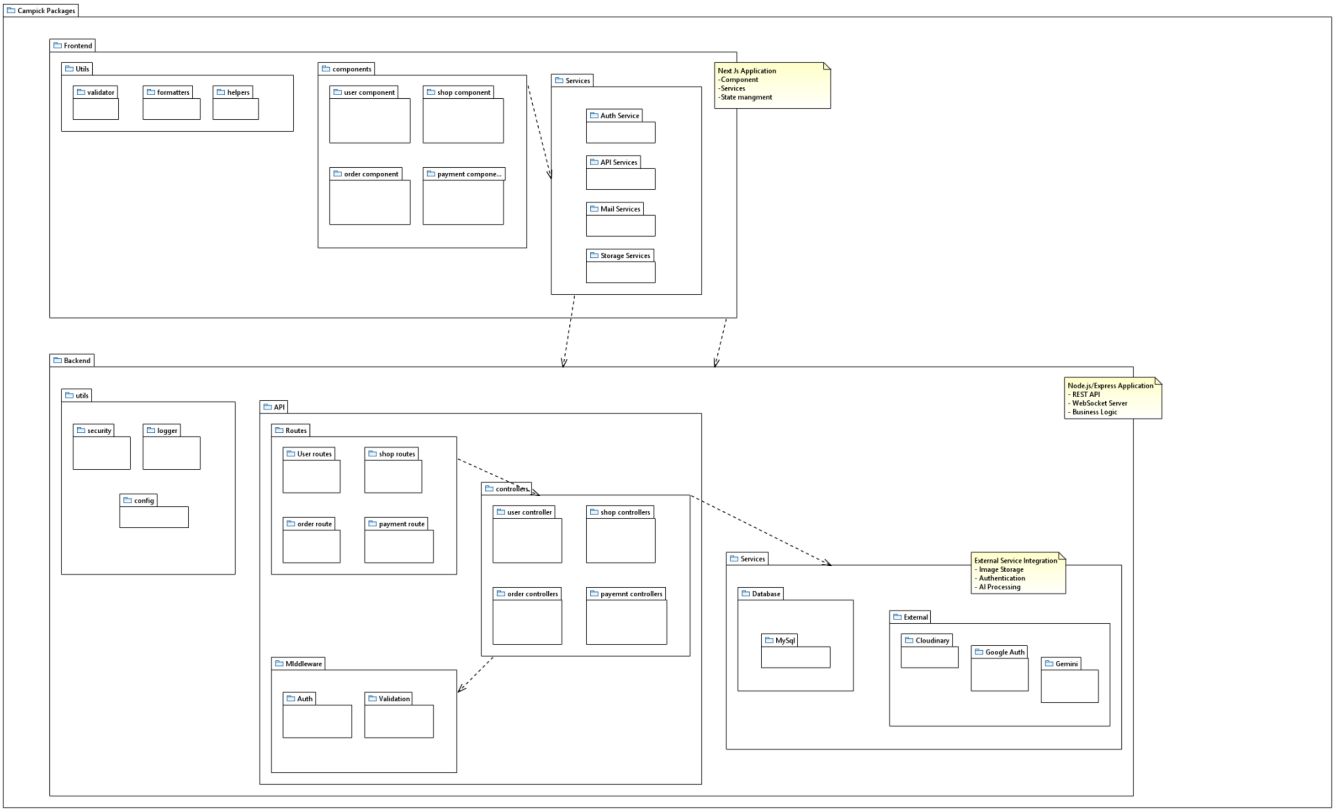
3. Deployment Diagram



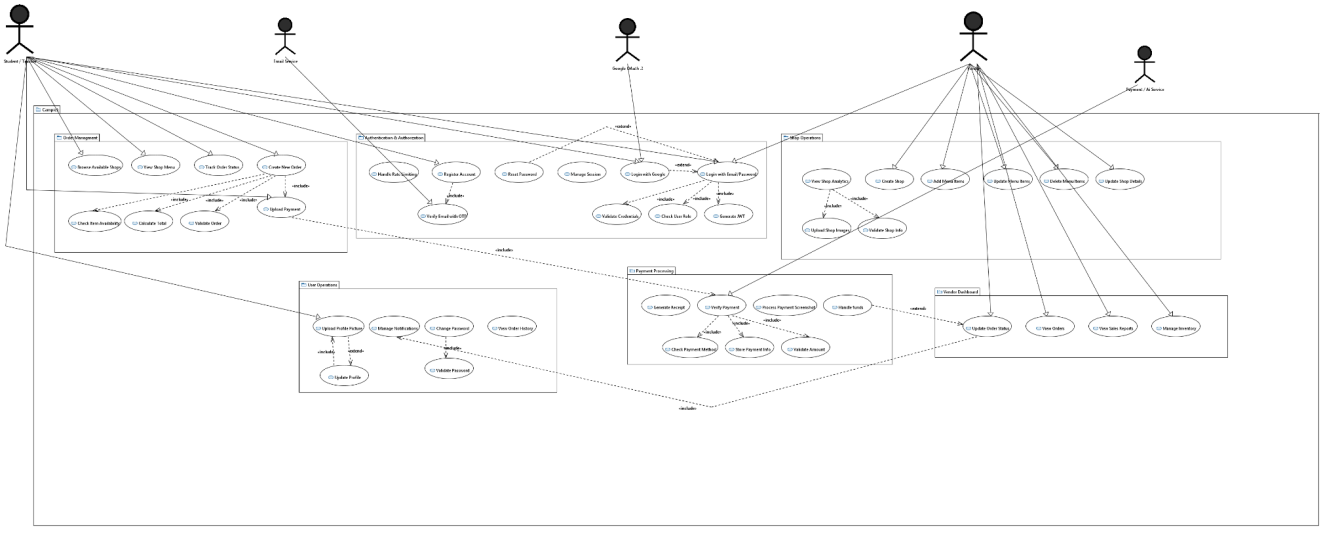
4. Entity Relation Diagram



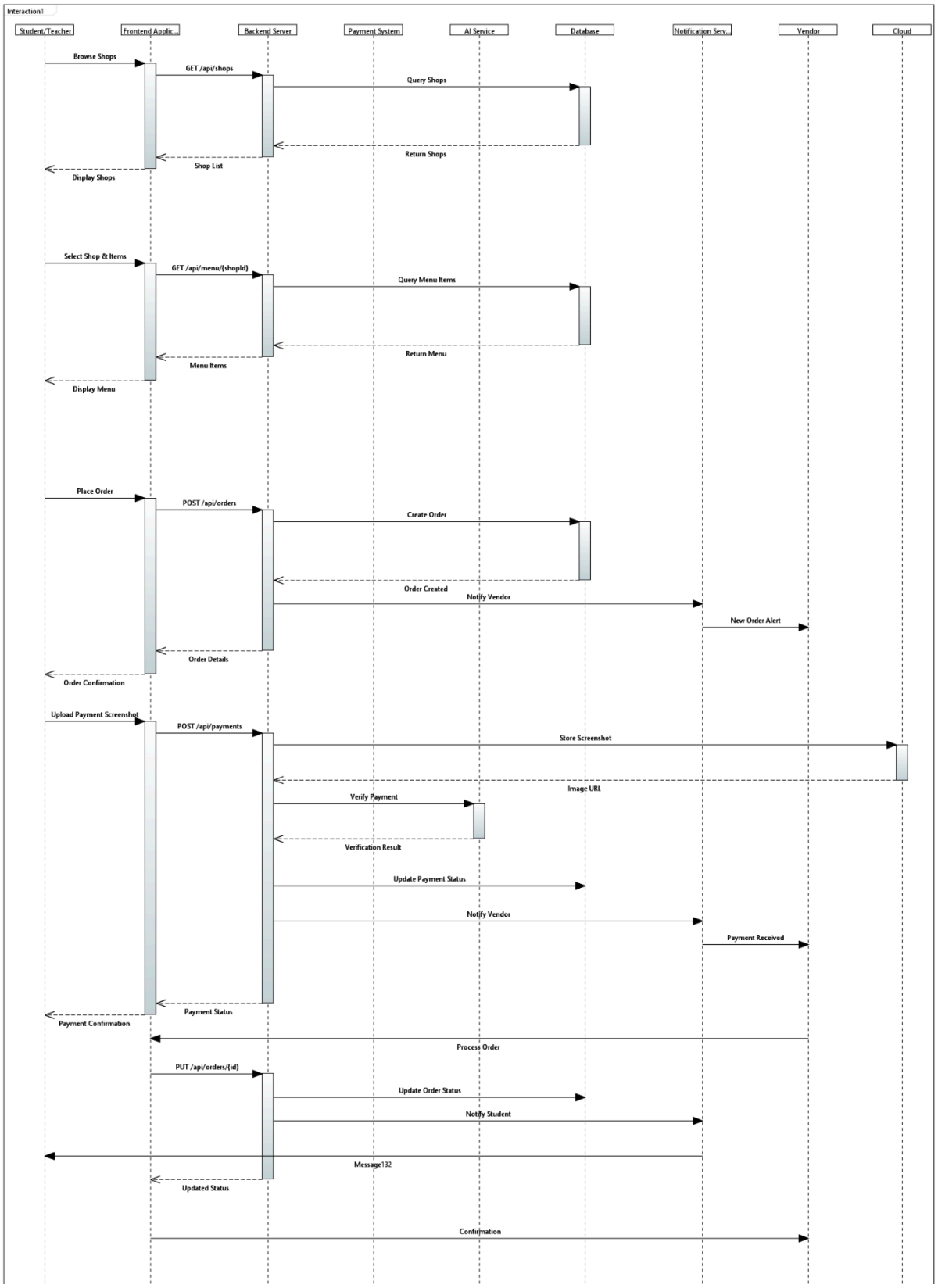
5. Package Diagram



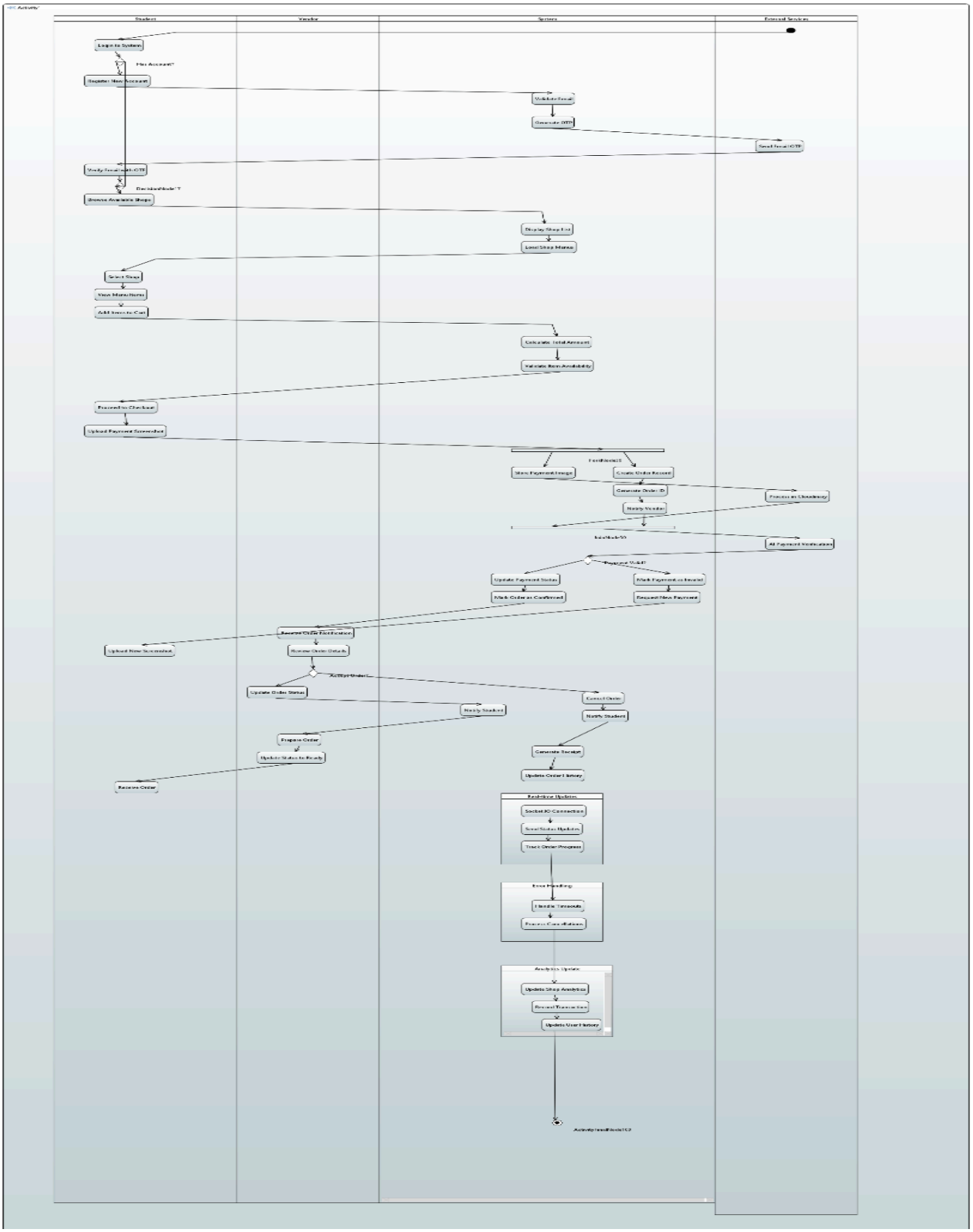
6. Use Case Diagram



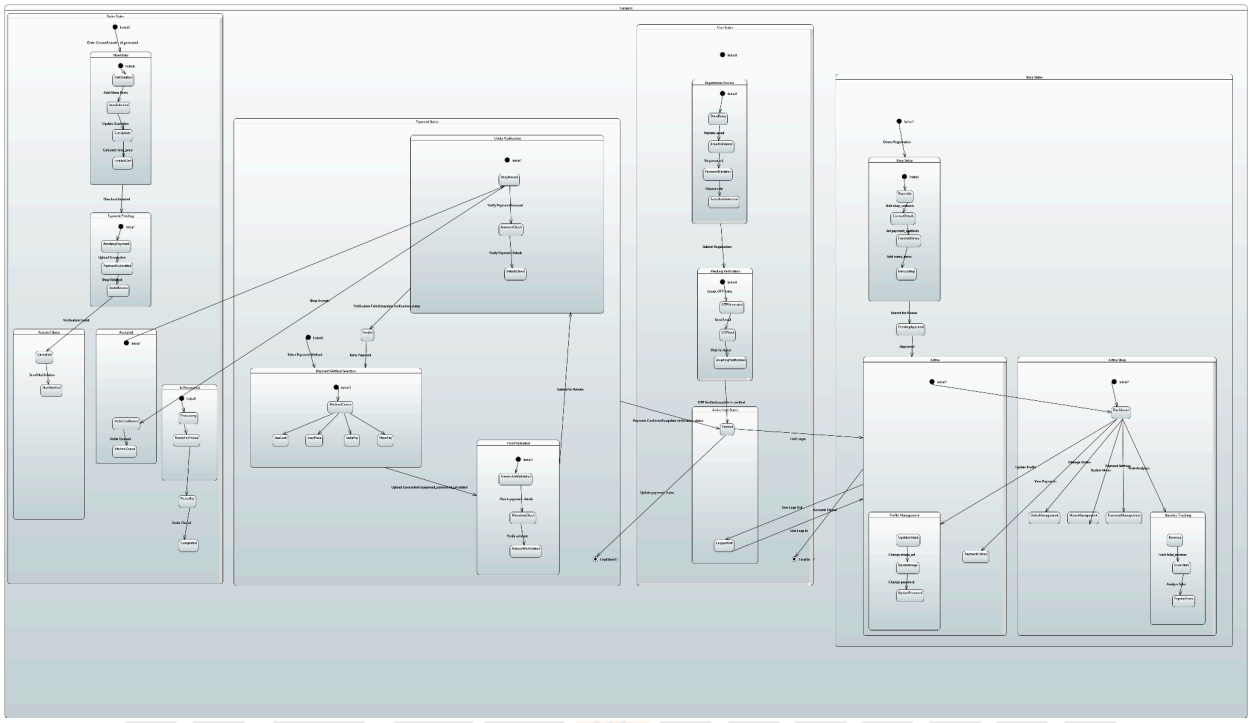
7. Sequence Diagram



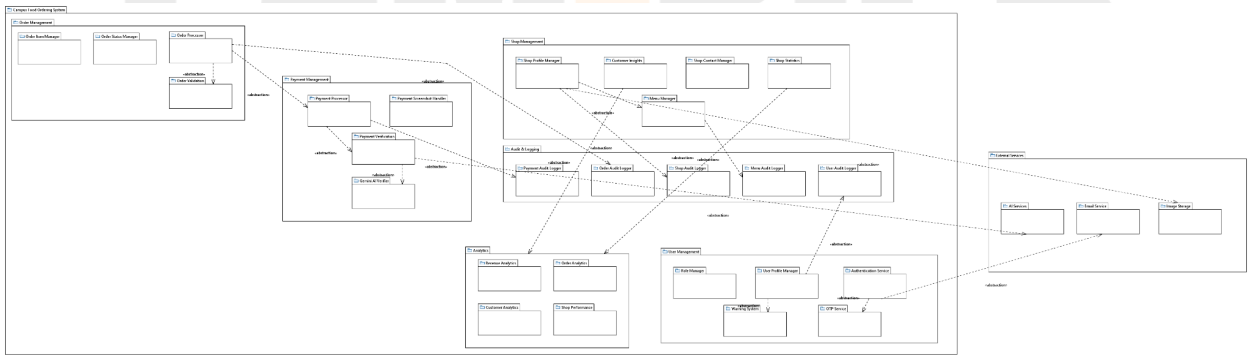
8. Activity Diagram



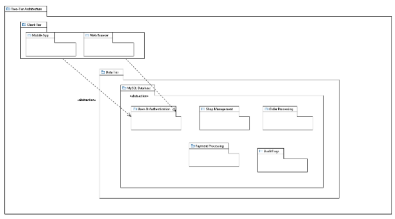
9. State Machine Diagram

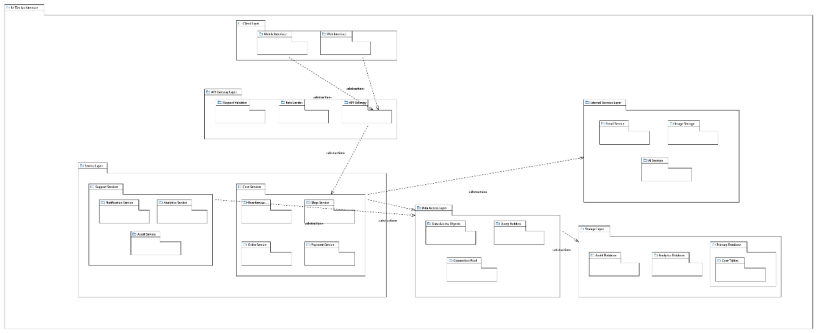


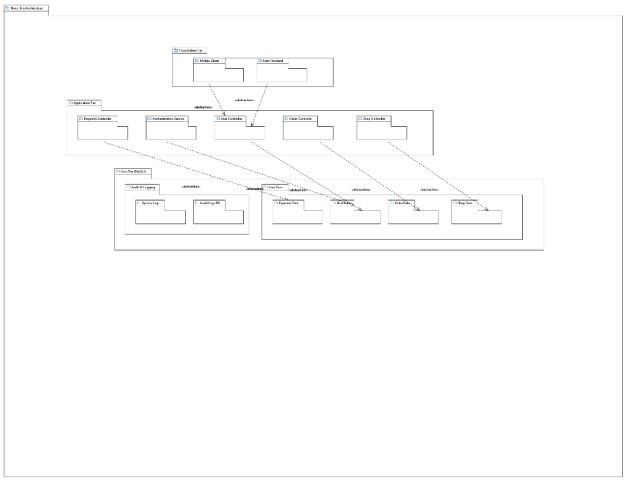
10. Subsystem Diagram



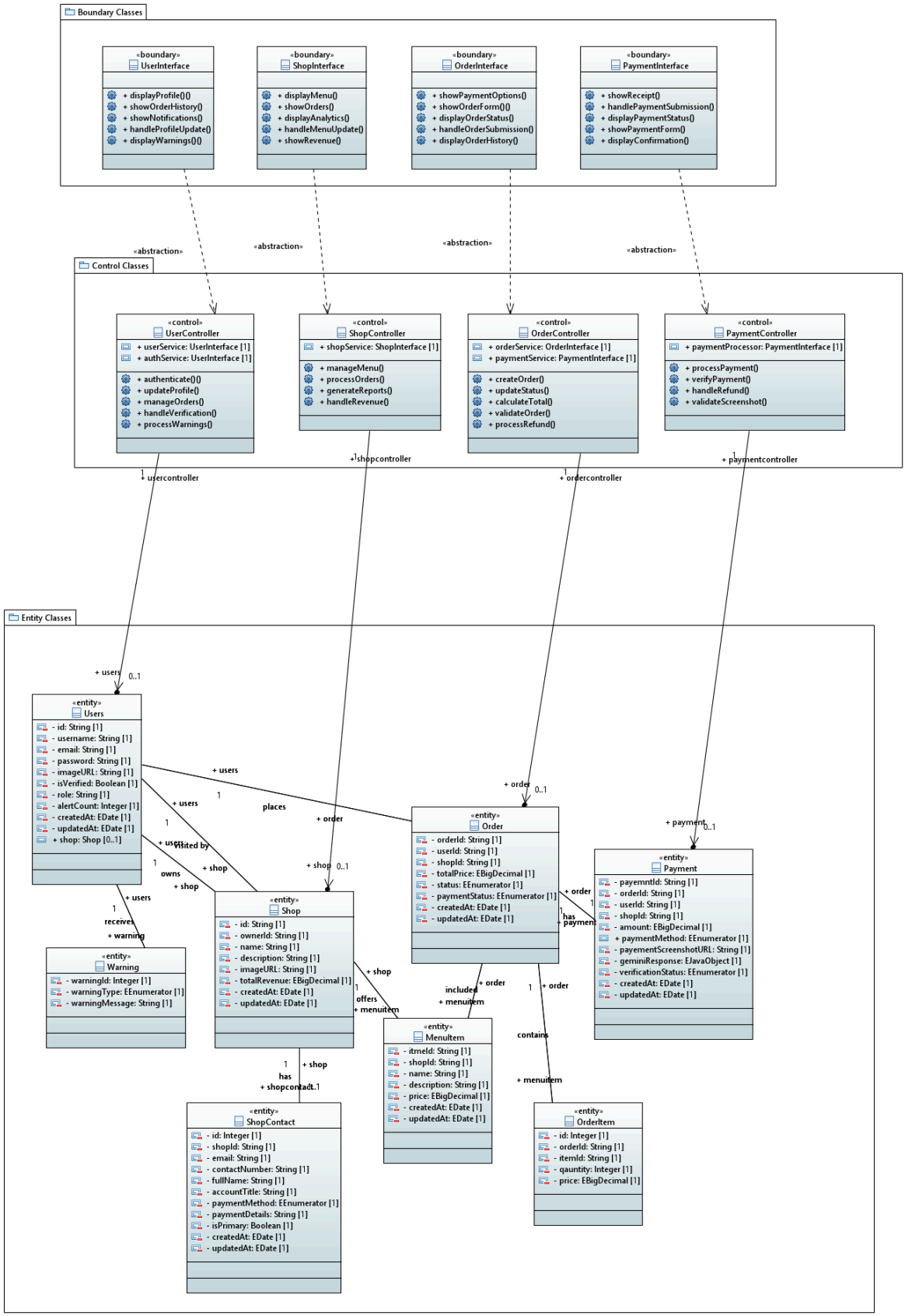
11. System Architecture (2-Tier, 3-Tier, N-tier)



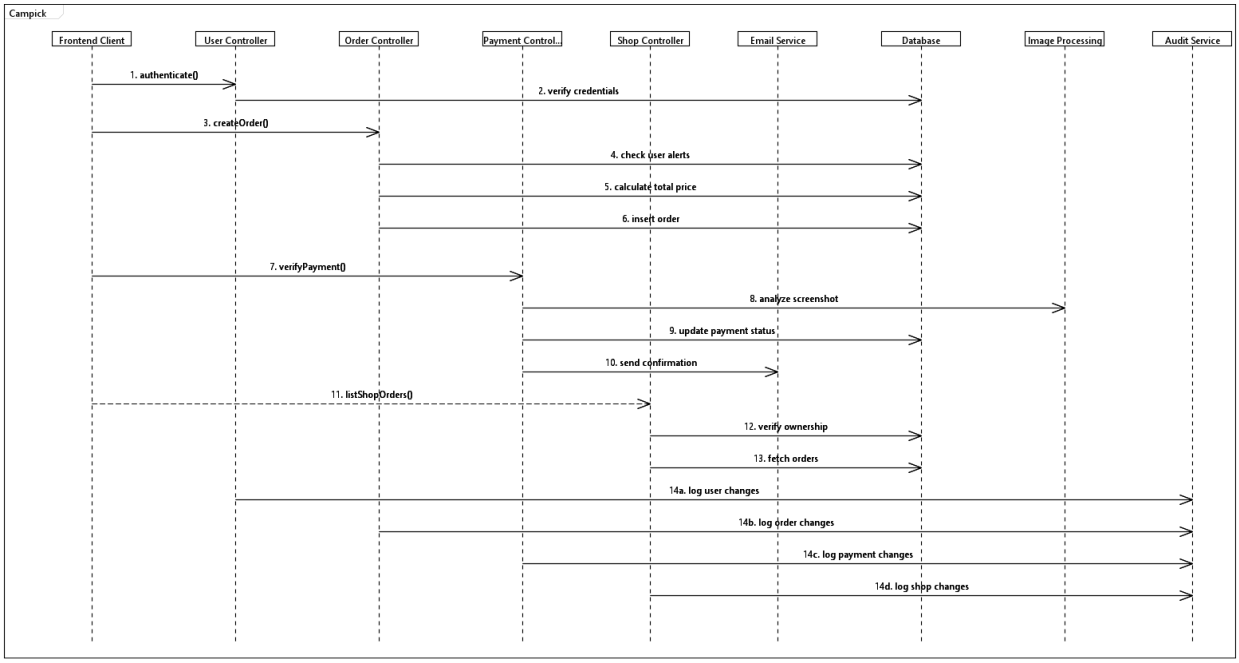




12. Entity Control Boundary class diagram



13.Communication Diagram



14.Collaboration Diagram