Software Design Specifications

for

POINT BASED SHARING SYSTEM

Prepared by:
ABHIRAM KARTHIKEYA – SE22UARI155
SANJAY VARMA – SE22UARI006
NIKITA LAKKOJU – SE22UARI112
HARSHITHA – SE22UARI204
YASHWANTH – SE22UARI100
SRIKRUTH VARMA – SE22UARI054

Document Information

Title:

POINT BASED TRADING SYSTEM

Project Manager:

Document Version No: 1
Document Version Date: 2/04/25

Prepared By: codeXcrew

Preparation Date: 2/04/25

Version History

Ver. No.	Ver. Date	Revised By	Description	Filename

Table of Contents

1 Introduction

The Point-Based Sharing System Software Design Specification gives a thorough description of the software architecture and design that underlies the system's core functionality. The document guarantees that everyone involved, from developers to testers and project managers, understands how the system is built and how its components relate to each other.

1.1 Purpose

The intent of this Software Design Specification is to present a thorough and organized design plan for the Point-Based Sharing System, according to the requirements defined in the Software Requirements Specification (SRS). This document acts as a bridge between the requirement phase and the implementation phase, where all stakeholders are ensured to have a clear technical vision of how the system will be constructed.

This document is targeted at the following audiences:

Developers: In order to comprehend the architecture, component interaction, and implementation details.
Testers: To ensure that the system is going to fulfil all the requirements and decide on testing strategies accordingly.
Project Managers and Stakeholders: To monitor design decisions and ensure that they support project objectives.
Future Maintenance Teams: To aid in future improvements or debugging activities with a clear point of reference.
The organisation of this report consists of parts describing the architecture of the system, component-level design, data flow, interface specifications, security measures, and performance objectives.

1.2 Scope

This Software Design Specification is used for the overall technical design of the Point-Based Sharing System. It establishes the software architecture, component structure, data models, and system module interactions. This document has an impact on the implementation, testing, deployment, and subsequent maintenance of the system.
The scope is the design of all functional aspects like user authentication, profile management, stock management, point tracking, transaction processing, user ratings, notifications, and admin controls. It also encompasses integration with external services like payment gateways, cloud storage, and identity management systems.
All subsystems and components outlined in this report will be implemented according to the requirements detailed in the SRS, and are important to the system working as anticipated.

1.3 Definitions, Acronyms, and Abbreviations

This section gives the definition of terms, acronyms, and abbreviations employed across this Software Design Specification for clarity and consistency:
SDD – Software Design Document: A document describing the system's design architecture.
SRS – Software Requirements Specification: A document that captures the system's functional and non-functional requirements.
API – Application Programming Interface: A set of rules that enables different software components to interact.
UI – User Interface: The area where human-machine interactions take place.

DB – Database: An organized collection of data stored in a computer system.
JWT – JSON Web Token: A small, URL-safe method of expressing claims in a secure exchange of information.
IAM – Identity and Access Management: A system of managing digital identities and access entitlements.
REST – Representational State Transfer: An architectural style applied to web services.
CRUD – Create, Read, Update, Delete: Simple operations for persistent storage.
JSON – JavaScript Object Notation: Lightweight data-interchange format.
HTTPS – Hypertext Transfer Protocol Secure: Secure protocol for communication over a computer network.

These definitions are necessary for comprehending the technical descriptions and design aspects within this document.

1.4 References

This design document refers to the following materials:

-IEEE Software Engineering Standards – for structure and formatting.

-SRS Document – for requirements and use cases.

-Use Case and Class Diagrams – for visual design references.

-Microservice Architecture Document – for module breakdown.

-Express.js Documentation

-MySQL Documentation

-Node.js Documentation

-REST API Design Guidelines

-Vite Documentation

-React Documentation

2 Use Case View

This section presents the comprehensive set of use cases derived from the SRS document for the Point-Based Lending System. These use cases represent critical interactions between the system and its primary actors – the User (which includes Borrower and Lender roles) and Admin. The use cases cover the major functional requirements and are essential for understanding the software design, as they help identify system responsibilities, data flow, and interaction patterns.
Primary Use Cases:
Register/Login: Allows users to sign up or authenticate into the system.
Forgot Password (<<extend>>): Optional flow from login for password recovery.
Browse Available Items: Enables users to explore items shared by others.
Request to Rent/Buy: Initiate a transaction by requesting an item.
Pay: Handle payment using the point-based system.
Confirm Transactions: Users confirm completion of borrowing/lending.
Deduct Points: System deducts from the borrower's account.
Refund Points: Refunds to the user in case of cancellation or issue.
List Items/Ride: Lenders can list items or rides to share.
Approve Request: Lenders accept incoming requests from borrowers.
Receive Points: Points credited to lenders post-confirmation.
Manage Listing (<<include>>): Edit or remove listed items.
Cancel Listing (<<extend>>): Optional flow to withdraw a listed item.
Leave a Review (<<extend>>): Provide feedback post-transaction.

Report Issue (<<extend>>): Users can report problems with a transaction.
Negotiations (<<include>>): Allows borrower and lender to discuss terms before approval.
Add Points: Users can top-up their point balance manually.

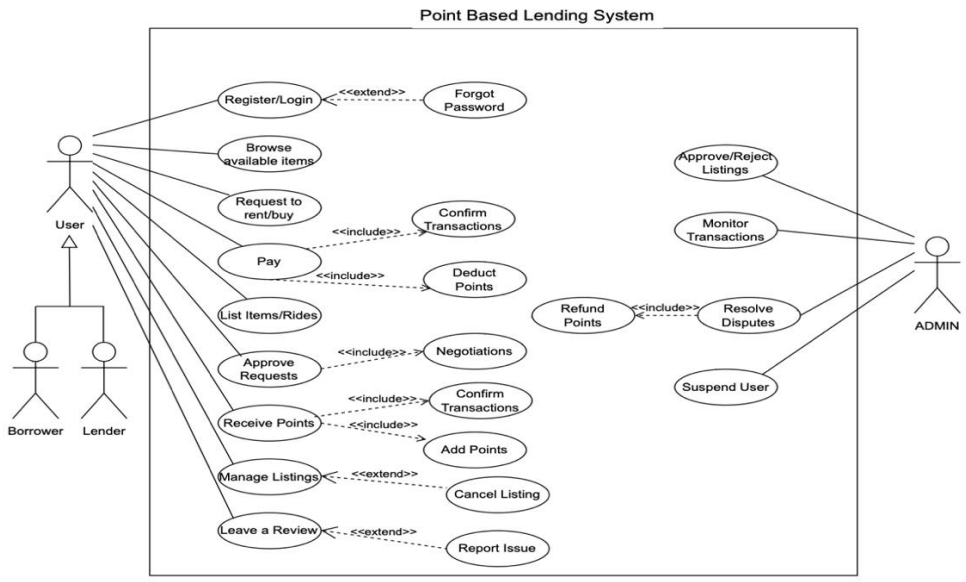
Admin Use Cases:
Approve/Reject Listings: Admin reviews and moderates user listings.
Monitor Transactions: Tracks and audits ongoing transactions.
Resolve Disputes: Handles conflict resolution between users.
Suspend User: Temporarily or permanently disables a user's account.

Actors:
User: General actor who can either be a Borrower or Lender.
Borrower: Requests and rents items.
Lender: Shares and approves requests for items.
Admin: System administrator responsible for moderation and maintenance.

Use Case Relationships:
Forgot Password extends Register/Login
Negotiations includes Approve Request
Manage Listing includes Cancel Listing
Leave a Review and Report Issue extend post-transaction processes

Purpose of This Diagram:
This use case diagram is central to the system's design as it captures all the critical user and admin functionalities. It supports the development of design elements like authentication modules, transaction handlers, point management, and admin controls, providing a high-level view of system behavior.

Use Case Diagram:

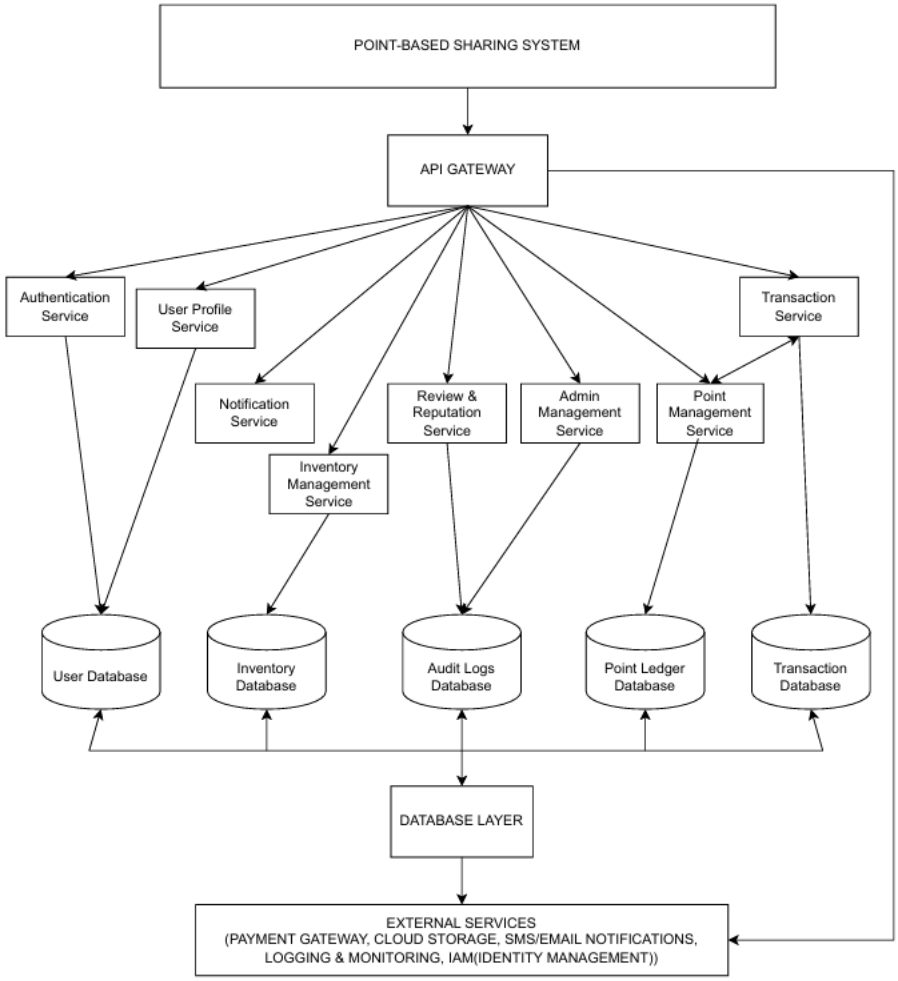


2.1 Use Case

Use Case 1 – Request to Rent/Buy

Use Case Name: Request to Rent/Buy
Actor(s): User (Borrower)
Brief Description:
This use case enables a user (borrower) to request something posted by another user (lender) for rent or purchase through the point-based system.
Preconditions:
The user has to be registered and logged in.
The item has to be posted and available.
The user has to have enough points to send the request.
Postconditions:
A rental request is sent to the lender.
Request status is changed to "Pending Approval."
Points are held (if any) until lender reply.
Basic Flow:
User views available items.
User views an item and clicks "Request to Rent/Buy."
System verifies availability and user point balance.
System creates a request transaction and informs the lender.
Request status is made "Pending Approval."
Alternative Flows:
Low Points:
System shows an error message and invites the user to add more points.
Item Unavailable:
System blocks the request and informs the user that the item is no longer in stock.
Lender Rejection:
When the lender rejects the request, the system changes the status of the request to "Rejected" and frees the held points.
Exceptions:
Network failure or server error when submitting the request: system retries or requests the user to retry later.

3 Design Overview



The Point-Based Lending System is a web application designed for student-to-student item and service sharing. While the point system is planned for future implementation, the current version focuses on enabling product listings, borrowing, and basic authentication. The application is built with an **Express.js** backend and **MySQL** as the relational database. The frontend interacts with the backend through REST APIs. The design prioritizes simplicity, modularity, and clean data handling.

3.1 Design Goals and Constraints

Design Goals:

- *Build a web application to allow students to list, borrow, and manage items.*
- *Enable basic authentication with login and signup.*
- *Keep design modular with RESTful API architecture.*
- *Separate roles for admin (product management) and general users.*
- *Use Express.js for scalable and maintainable backend development.*
- *Ensure that the system is extendable in the future (e.g.,for further feature implementation like point system, profiles).*

Constraints:

- *Backend must use **Express.js** and **MySQL** as the database.*
- *Frontend interacts only with essential backend endpoints.*
- *Limited development time and team size.*

- *Must be hostable using simple deployment methods (e.g., AWS EC2 or similar).*
- *Health check and monitoring limited to basic actuator-style endpoints.*

3.2 Design Assumptions

- *All users must register and login to access product functionalities.*
- *The admin role is manually assigned (no user role management implemented).*
- *User data is minimal and stored in a normalized structure in MySQL.*
- *There is no frontend-side caching; all data is fetched from backend as needed.*
- *Admin endpoints are protected (assumed) via token or simple role-checking.*
- *Only essential frontend pages are built (homepage, product listing, admin panel).*
- *No third-party authentication providers (like Google OAuth) are used.*

3.3 Significant Design Packages

The system uses a layered architecture with clear separation of concerns. It consists of a Vite-powered frontend and an Express.js backend, structured into the following core modules:

- ***AuthService***
  Manages user authentication (signup and login).
  Routes: /api/auth/signup, /api/auth/login
- ***ProductService***
  Retrieves product data for users.
  Routes: /api/products, /api/products/refresh, /api/store, /api/homepage
- ***AdminService***
  Handles product management actions like add, update, and delete.
  *Routes: /api/admin/**
- ***HealthService***
  Provides basic monitoring and diagnostics.
  Routes: /actuator/health, /actuator/env, /actuator/mappings
- ***DatabaseService***
  Central MySQL handler used by all services for querying and updating data.

The frontend consumes these REST APIs and presents the user interface. Each backend module is modular, relies on the DatabaseService, and is designed for easy maintenance and future scalability.

3.4 Dependent External Interfaces

The application currently does not depend on any external third-party APIs or applications for core functionality. All data processing and logic are handled internally using Express.js and a local MySQL database.
However, for system diagnostics, Spring Boot-style actuator endpoints are used internally for monitoring (e.g., /actuator/health, /actuator/env, etc.), though these are exposed by the same application and not external.

The table below lists the public interfaces this design requires from other modules or applications.

External	Module Using the	Functionality/

Application and Interface Name	Interface	Description
None	N/A	All modules work with internal interfaces and database only.

3.5 Implemented Application External Interfaces (and SOA web services)

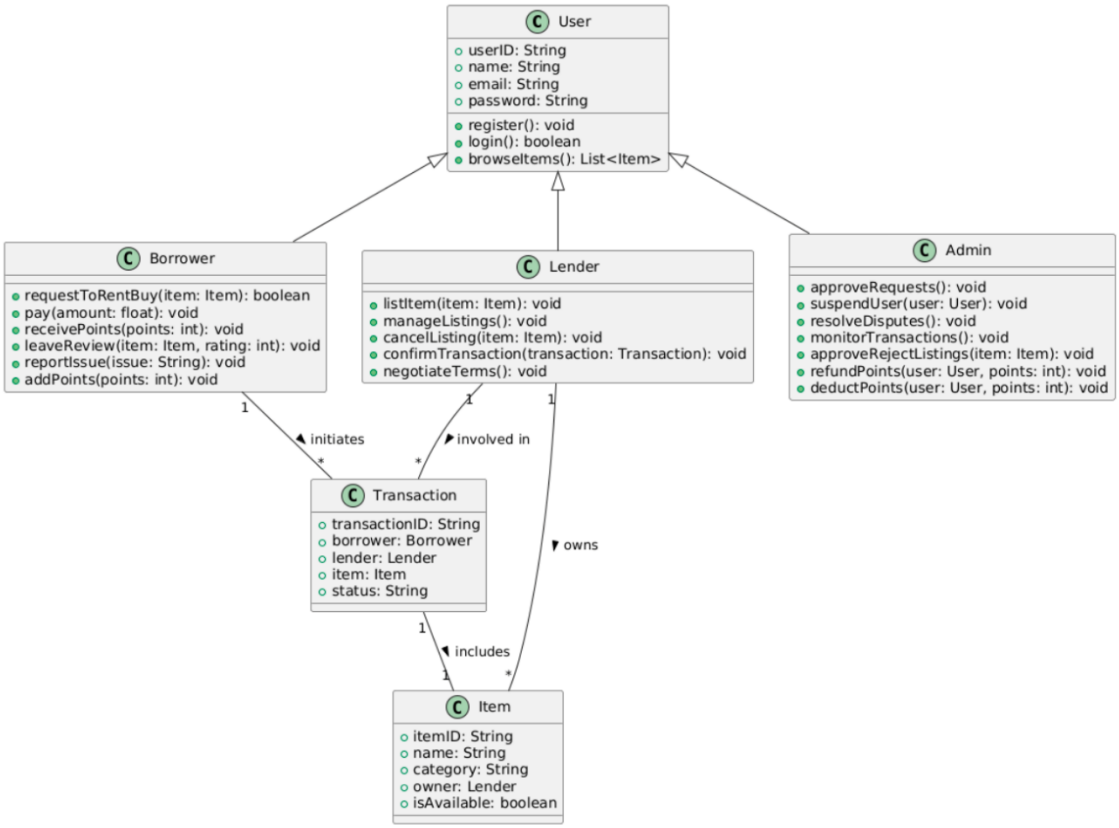
The application exposes RESTful APIs that are consumed by the frontend and potentially other services. Each backend module is responsible for a specific set of endpoints:

The table below lists the implementation of public interfaces this design makes available for other applications.

Interface Name	Module	Description
/api/auth/signup	AuthService	Handles user registration and stores data in MySQL
/api/auth/login	AuthService	Authenticates user login requests
/api/products	ProductService	Fetches all available product data
/api/products/refresh	ProductService	Refreshes the list of products
/api/homepage	ProductService	Sends data for homepage display
/api/store	ProductService	Gets store-specific products
/api/admin/add-product	AdminService	Adds a new product to the database
/api/admin/remove-product	AdminService	Deletes a selected product
/api/admin/update-product	AdminService	Updates details of a product
/actuator/health	HealthService	Checks application health status
/actuator/env	HealthService	Displays environment configuration
/actuator/mappings	HealthService	Lists available endpoints and mappings

4 Logical View

This section outlines the Detailed Design of the Point-Based Sharing System, indicating how various modules and classes collaborate to fulfil fundamental use cases such as lending, borrowing, and approving transactions. The logical view encompasses class decomposition and sequence flow.



4.1 Design Model

This system follows an object-oriented design approach. The software is decomposed into modules representing core roles and functions of the system. These modules are implemented using the following significant classes:

a. Classes Involved

***User**: Represents all system users (borrowers, lenders, admins)*
***Borrower** (inherits from User): Can borrow items*
***Lender** (inherits from User): Can list items for lending*
***Admin** (inherits from User): Manages transactions and users*
***Item**: Represents each item available for sharing*
***Transaction**: Captures and tracks borrow/lend operations*

b. Class Attributes and Responsibilities

Class	*Attributes*	*Responsibilities*

User	*userID, name, email, password*	*Base class for authentication and account management*
Borrower	*Inherits from User; borrowedItems, points*	*Request items, view borrowing history*
Lender	*Inherits from User; listedItems*	*List items, approve/deny borrow requests*
Admin	*Inherits from User; adminID*	*Review and approve transactions, manage user accounts*
Item	*itemID, name, category, owner (Lender), isAvailable*	*Describes item metadata and availability status*
Transaction	*transactionID, borrower, lender, item, status*	*Tracks the lifecycle and status of borrow/lend transactions*

These classes collectively enable the functionality and logic for borrowing, lending, tracking items, and system moderation.

4.2 Use Case Realization

*The following describes the realization of the **Borrow Item** use case, using the sequence of interactions from your lab document:*

Use Case: Borrow Item

***Actors**: Borrower, Lender, Admin**Objects**: Borrower, Lender, System, Item, Transaction, Admin*

***Basic Flow**:*

Login

Borrower -> System: Login ()
System -> Borrower: Login successful

Browse and Select Item

Borrower -> System: Browse items
System -> Borrower: Display available items

Borrowing Request

Borrower -> Lender: Request to rent item
Lender -> System: Confirm item availability
System -> Lender: Item is available

Lender Approves

Lender -> Borrower: Approve request

Transaction and Payment

Borrower -> System: Pay using points
System -> Transaction: Create transaction

Admin Review (Alternative Flow)

System -> Admin: Request approval for transaction
Admin -> System: Approve/Reject transaction

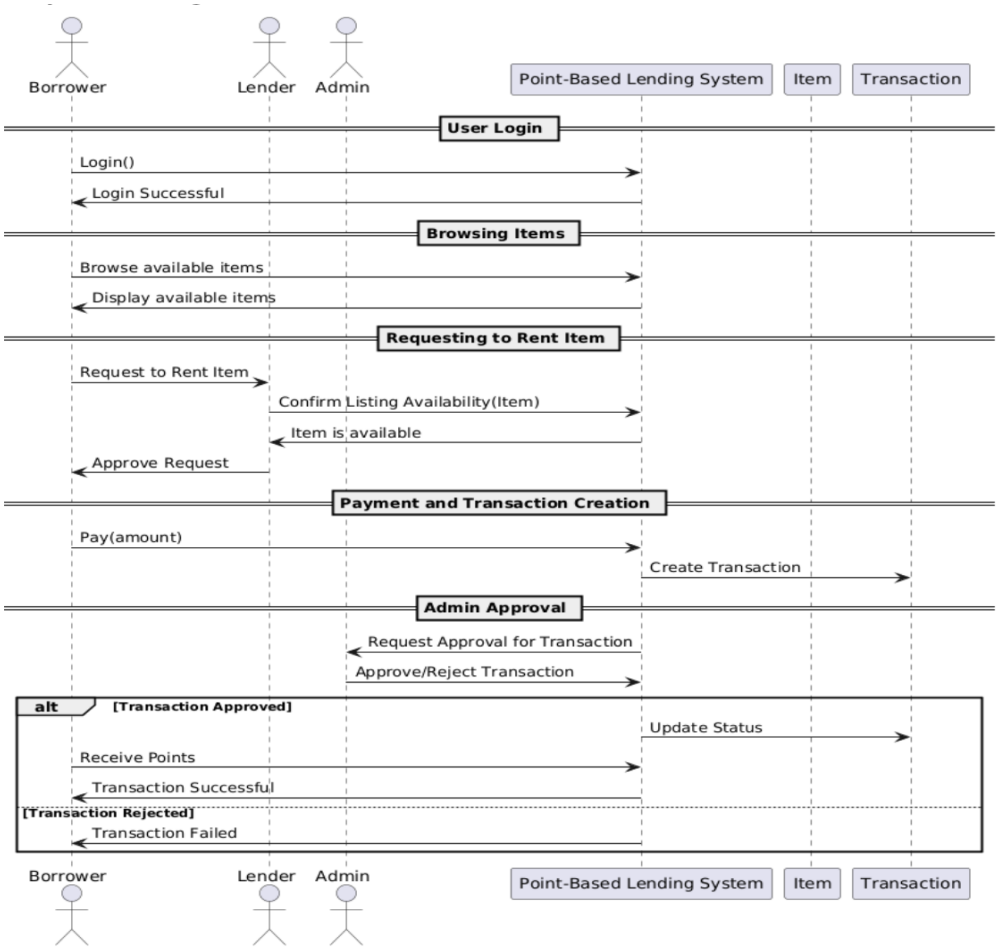
***If Approved**:*

System -> Transaction: Update status
System -> Borrower: Notify transaction success

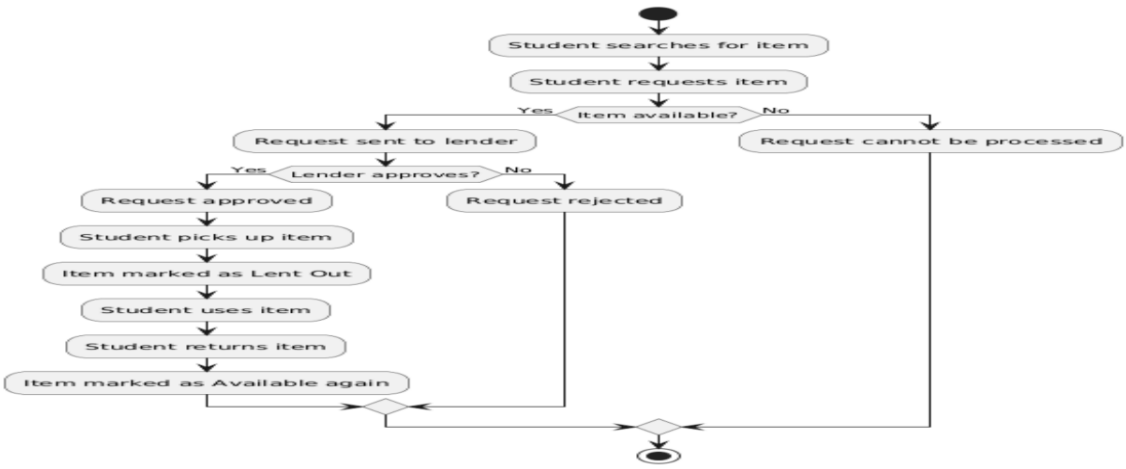
***If Rejected**:*

System -> Borrower: Notify transaction failure

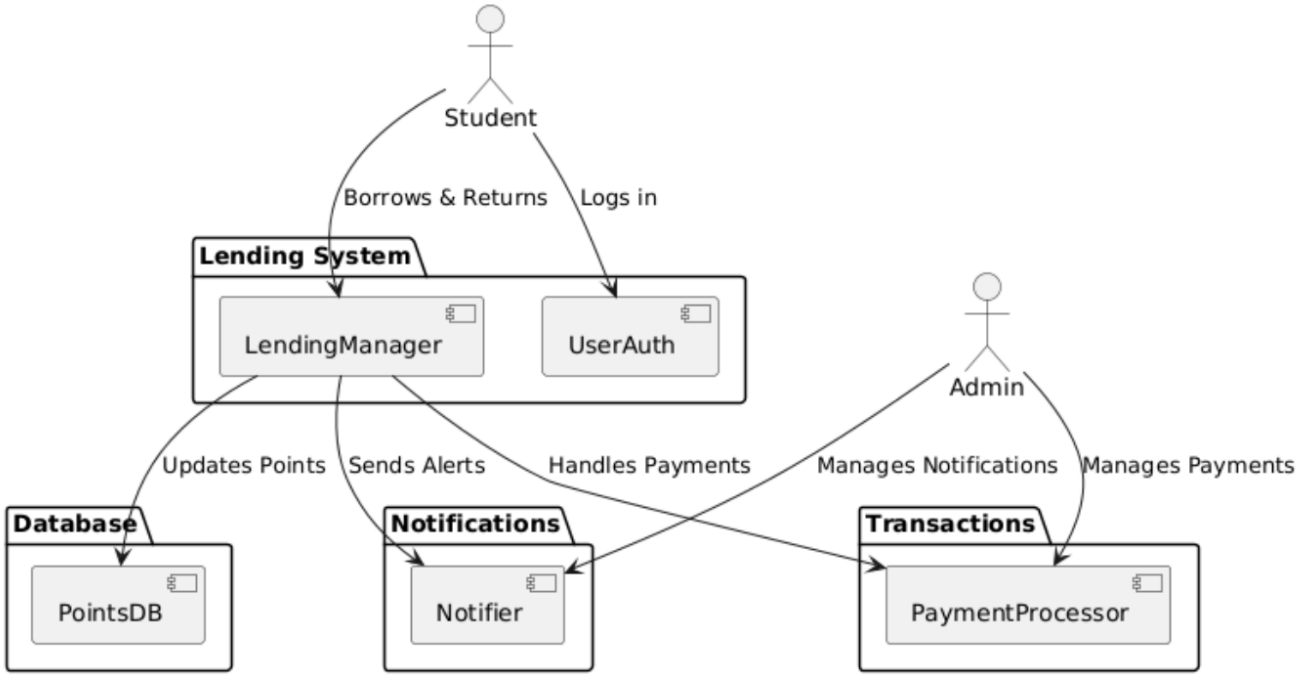
SEQUENCE DIAGRAM



ACTIVITY DIAGRAM



COMPONENT DIAGRAM



5 Data View

This section describes the persistent data storage perspective of the Point-Based Sharing System. It outlines how data is organized, stored, and related to support the system's core functionalities, such as user management, item sharing, transaction processing, and administrative oversight. The system uses a MySQL relational database to store persistent data, ensuring efficient querying and data integrity.

5.1 Domain Model

The domain model represents the core entities and their relationships within the Point-Based Sharing System. These entities are derived from the system's functional requirements and use cases, such as user registration, item listing, transaction processing, and review management.

Core Entities:

User: Represents an individual registered on the platform (can act as a Borrower, Lender, or Admin).

Item: Represents an item available for sharing or borrowing.

Transaction: Captures the details of a borrowing/lending operation.

Review: Stores feedback and ratings provided post-transaction.

Entity Relationships:

A User can list multiple Items as a Lender and request multiple Items as a Borrower (one-to-many).

An Item is owned by one User (Lender) and can be part of multiple Transactions (one-to-many).

A Transaction involves one Borrower, one Lender, and one Item (many-to-one relationships).

A Review is linked to a Transaction and provided by a User (Borrower or Lender) (many-to-one).

5.2 Data Model (Persistent Data View)

The data model defines the relational database schema used to store persistent data in MySQL. It includes tables corresponding to the domain model entities, with fields designed to support the system's functionality. The tables are normalized to reduce redundancy and ensure data consistency.

Tables:

Users: Stores user account information.

Items: Stores details of items available for sharing.

Transactions: Tracks borrowing/lending activities.

Reviews: Records feedback and ratings after transactions.

5.2.1 Data Dictionary

Entity	*Attribute*	*Description*	*Data Type*	*Constraints*
Users	*user_id*	*Unique identifier for each user*	*INT*	*PRIMARY KEY, AUTO_INCREMENT*
	name	*Full name of the user*	*VARCHAR(100)*	*NOT NULL*
	email	*User's email address (used for login)*	*VARCHAR(255)*	*NOT NULL, UNIQUE*
	password_hash	*Encrypted password for secure authentication*	*VARCHAR(255)*	*NOT NULL*
	role	*User role (e.g., Borrower, Lender, Admin)*	*ENUM*	*NOT NULL, DEFAULT 'User'*
	created_at	*Timestamp of account creation*	*DATETIME*	*NOT NULL*
Items	*item_id*	*Unique identifier for each item*	*INT*	*PRIMARY KEY, AUTO_INCREMENT*
	name	*Name of the item*	*VARCHAR(100)*	*NOT NULL*
	category	*Category of the item (e.g., Electronics, Books)*	*VARCHAR(50)*	*NOT NULL*
	owner_id	*Foreign key linking to the Lender (Users table)*	*INT*	*FOREIGN KEY (user_id)*
	is_available	*Availability status of the item*	*BOOLEAN*	*DEFAULT TRUE*
	description	*Brief description of the item*	*TEXT*	*NULL*

Transactions	*transaction_id*	*Unique identifier for each transaction*	*INT*	*PRIMARY KEY, AUTO_INCREMENT*
	borrower_id	*Foreign key linking to the Borrower (Users)*	*INT*	*FOREIGN KEY (user_id)*
	lender_id	*Foreign key linking to the Lender (Users)*	*INT*	*FOREIGN KEY (user_id)*
	item_id	*Foreign key linking to the Item*	*INT*	*FOREIGN KEY (item_id)*
	status	*Transaction status (e.g., Pending, Approved, Completed)*	*ENUM*	*NOT NULL*
	start_date	*Date the transaction begins*	*DATETIME*	*NOT NULL*
	end_date	*Date the transaction ends (if applicable)*	*DATETIME*	*NULL*
Reviews	*review_id*	*Unique identifier for each review*	*INT*	*PRIMARY KEY, AUTO_INCREMENT*
	transaction_id	*Foreign key linking to the Transaction*	*INT*	*FOREIGN KEY (transaction_id)*
	reviewer_id	*Foreign key linking to the User providing review*	*INT*	*FOREIGN KEY (user_id)*
	rating	*Numerical rating (1 to 5 stars)*	*INT*	*NOT NULL, CHECK (1-5)*
	comments	*Text feedback or comments*	*TEXT*	*NULL*
	created_at	*Timestamp of review submission*	*DATETIME*	*NOT NULL*

6 Exception Handling

The Point-Based Sharing System employs structured exception handling to make application behaviour robust and reliable. This section describes the kinds of exceptions that can arise, their handling, logging, and follow-up actions that are needed.

6.1 Types of Exceptions and Circumstances

Exception	*Circumstance*	*Handling*
AuthenticationFailureException	*Invalid login credentials or expired session tokens*	*Prompt user to re-enter credentials; reroute to login page*
ItemUnavailableException	*Item requested has already borrowed or marked unavailable*	*Display alert and recommend similar available items*
InvalidDiscountCodeException	*User uses an incorrect or expired discount code*	*Inform user and allow retry with valid code*
DatabaseConnectionException	*System is unable to reach the database*	*Trigger retry mechanism, notify admin if persistent*
TransactionFailureException	*Failure during point deduction or item status update*	*Rollback transaction, notify user of failure, and log full context*
NotificationSendFailureException	*Failure in sending SMS or email notification*	*Retry via backup channel (e.g., switch from SMS to email) and log for audit*
UnauthorizedAccessException	*User attempts to perform admin-level actions without proper privileges*	*Show access denied message and log unauthorized access attempt*

6.2 Logging Strategy

All exceptions are logged to a centralized Audit Logs Database. All log entries include:
Timestamp
Exception type
Affected service/module
User ID (if applicable)
Stack trace or error message
Suggested or automated recovery action

6.3 Follow-Up Actions

User-visible errors (e.g., invalid credentials, out-of-stock items) induce immediate user notification and retry possibilities.
System-level errors (e.g., DB failure) trigger automated notifications to the administrator for immediate investigation.
Critical mistakes (e.g., multiple unauthorized access attempts) are escalated for manual inspection by the admin staff and can result in temporary account suspension or rate-limiting.

7 Configurable Parameters

This section outlines the parameters used by the application that are configurable. Since this is an Express.js and MySQL-based system, the configuration is handled through environment variables or configuration files. Each parameter includes a name, its definition and usage, and whether it is dynamic (i.e., can be changed without restarting the application).

This table describes the simple configurable parameters (name / value pairs).

Configuration Parameter Name	Definition and Usage	Dynamic?
IMAGE_PATH	Base URL for accessing images from backend API	Yes
PORT	Port on which the image server runs	No
ENABLE_ADMIN	Enables admin-specific functionality	No
IGNORE_IMAGE_FOLDERS	Folders to ignore while serving images	No
DB_HOST	Hostname for MySQL database	No
DB_PORT	Port number for MySQL	No
DB_NAME	Name of the database	No
DB_USER	Username for DB authentication	No
DB_PASSWORD	Password for DB authentication	No
DB_URL	JDBC URL for MySQL connection	No
SERVER_PORT	Port for backend (auth service) server	No
JWT_SECRET	Secret key used to sign JWT tokens	No
JWT_EXPIRATION	JWT token expiration time in milliseconds	No
API_REQUEST_LOGGING	Enables or disables API logging	Yes
VITE_BACKEND_API_BASE_URL	Backend base URL used in frontend	No
VITE_BACKEND_API_PREFIX	Backend API prefix for routing	No
VITE_AUTH_API_BASE_URL	Auth API base URL used in frontend	No
VITE_AUTH_API_BASE_PREFIX	Auth API prefix	No
VITE_ADMIN_USERNAME	Username for admin login	No
VITE_ADMIN_PASSWORD	Password for admin login	No
VITE_ADMIN_PRODUCT_TAGS	Tags used to mark products as special categories	Yes
FRONTEND_URLS	Whitelisted frontend URLs for CORS	Yes

8 Quality of Service

This section outlines design considerations concerning the operational quality of the system—specifically areas of availability, security, performance, and monitoring.

8.1 Availability

The Point-Based Sharing System is architected to have an extremely high level of availability with a 99.9% uptime requirement met by institutional as well as user expectations. Following are the design strategies implemented towards this end:
Microservices Architecture: Each of the services works independently, diminishing the effect of single service failure.
Redundant Server Deployment: Major services are deployed in redundant copies to prevent single points of failure.
Scheduled Maintenance Windows: Maintenance and housekeeping are performed at low-usage times to minimize disruption.
Health Checks and Failover: Automated health monitoring ensures that failed services are automatically restarted or redirected to redundant instances.
Availability will be impacted at mass inventory import, bulk points updates, or migration of data—thus these operations are either scheduled or queued to execute in the background asynchronously.

8.2 Security and Authorization

The layout follows robust security and authorization practices based on the following needs:

Secure Authentication: All users authenticate through JWT tokens through institutional single sign-on (SSO) or password-based login securely.
Role-Based Access Control (RBAC): Admin and User roles are enforced at the service level to deny unauthorized access.
Data Encryption: Sensitive data is encrypted in transit (through HTTPS) and at rest (through AES-256).
Session Management: Expired sessions are automatically invalidated to avoid unauthorized reuse.
Audit Logging: All important actions (such as transactions, login attempts, and admin modifications) are logged and stored in an Audit Logs Database.
Self-Service Access Management: Admins can grant/revoke roles and privileges through the Admin Management Service.

These features comply with GDPR and India's IT Act, with the provision for users to delete their accounts and data permanently.

8.3 Load and Performance Implications

The system is built to handle expected loads while ensuring smooth performance:
1.Supports 100+ concurrent users without major lag.
2.Dashboard and key pages load within 2–3 seconds on 4G.
3.Point updates and reminder actions respond within 1 second.

4.Indexed database for faster queries on items, users, and points.
5.Load testing simulates peak lending periods for performance tuning.
6.Image compression and CDN used for faster item image delivery.

8.4 Monitoring and Control

Monitoring tools and health checks are integrated into the backend to maintain system stability:
1.Health checks for all microservices (borrow, return, reminder, etc.).
2.Logs maintained for logins, borrow/return actions, and point updates.
3.Alerts for overdue items, failed services, and login anomalies.
4.Admin dashboard shows metrics: active users, lending activity, uptime.
5.Auto-recovery scripts restart failed services and run daily checks.