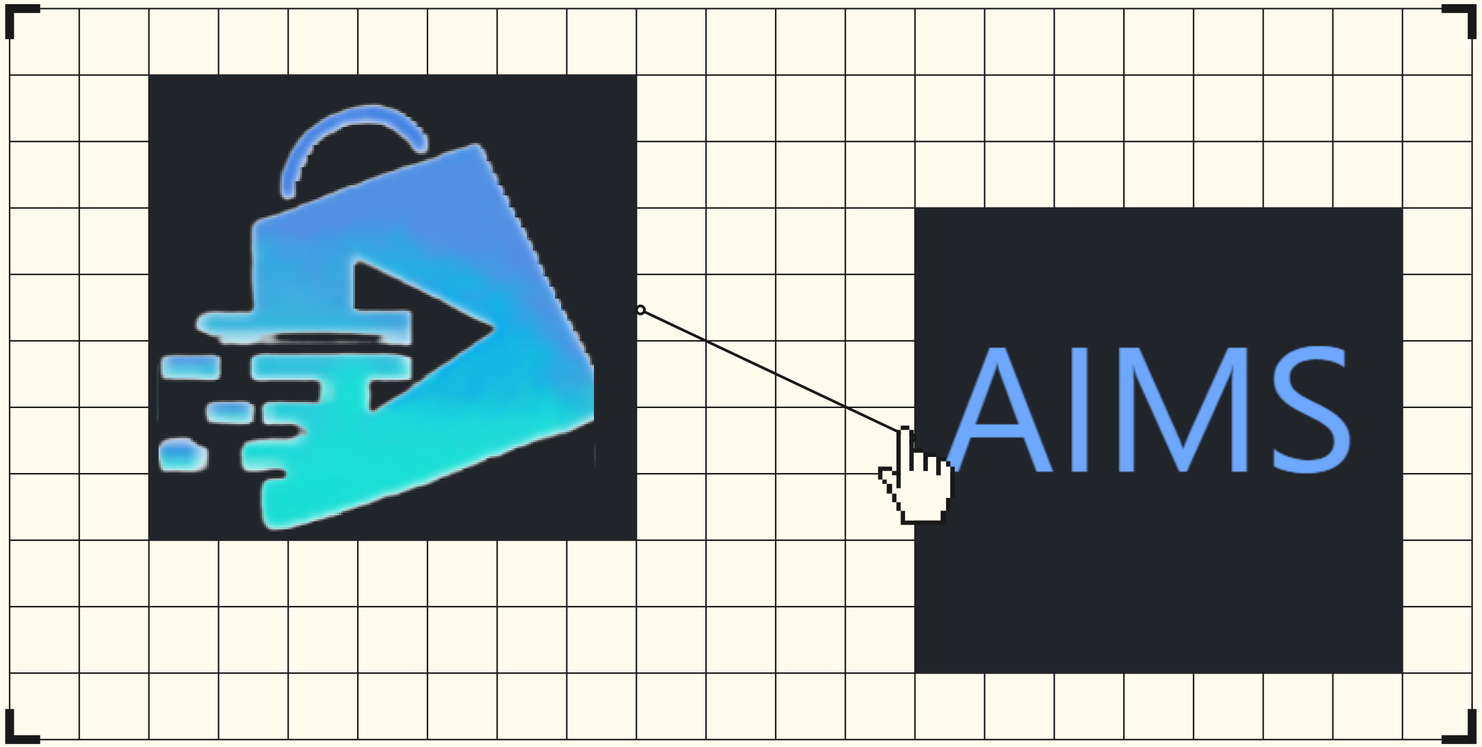
AIMS STORE APPLICATION

GROUP 18

AGENDA ★

1. Introduction
2. Overall description
3. Architectural design
4. Detailed design
5. Demonstration



Member of Group

Nguyen Viet Thang

Team member - Backend

Nguyen Duc Thang

Leader & Frontend

Pham Tuan Tai

Team member - Backend

Nguyen Tien Thanh

Team member - Backend

Vo Van Thanh

Team member - Backend

PART 1: INTRODUCTION

GROUP 18

INTRODUCTION

Search Product

- AIMS is designed to necessitate the needs for customers who wish to search and purchase any products from AIMS.

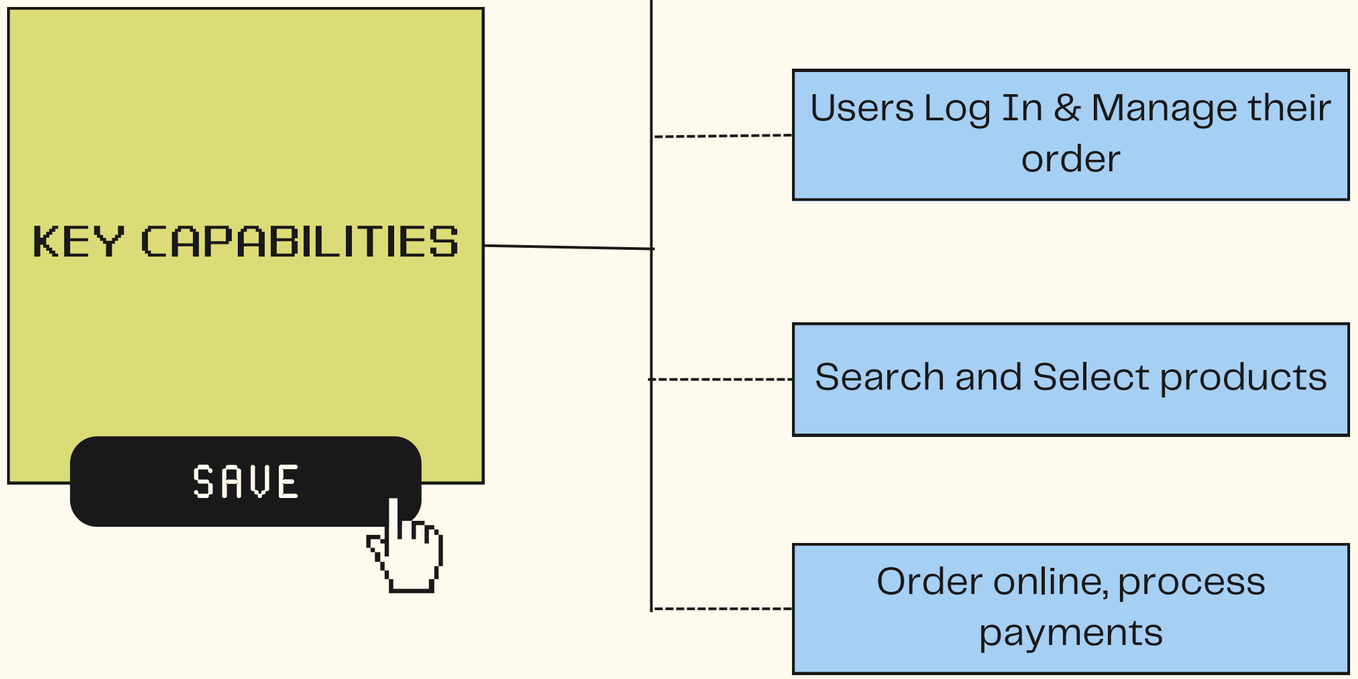
- AIMS satisfies the demand of administrator to create, modify, delete or display different products in the system.

Online Store

SCOPE

AIMS will automate core operations of an online store for selling, managing media products including: books, cds, and dvds

It will manage the store's inventory, facilitate purchase transactions at point-of-sale, process customer orders

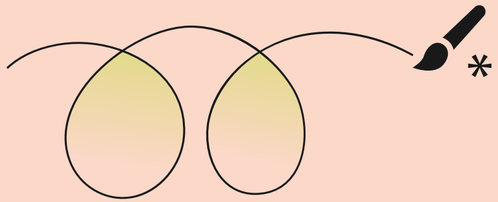


PART 2: OVERALL DESCRIPTION

GROUP 18

GENERAL OVERVIEW

Search Product

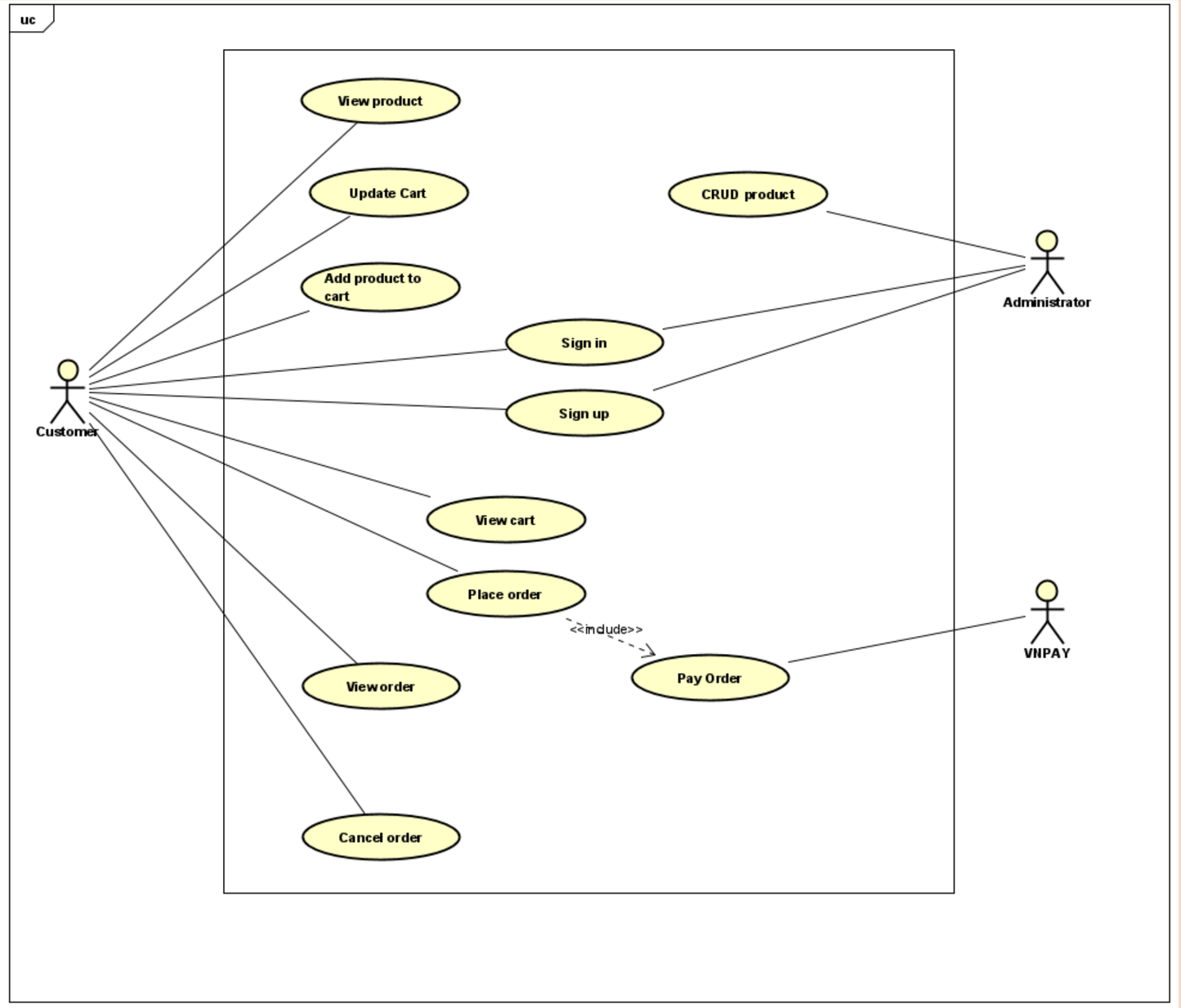


The AIMS will be a web-based application, with both a backend system and a frontend website

Customers and manager access the system via a web-based user interface,. The system will integrate with a database to retrieve products

UPLOAD

GENERAL USE CASE



ASSUMPTION

- Continuously available network and server infrastructure.
- Relational database MySQL to store and manage bookstore data
- Access to modern web browser
- Standard Payment Protocol
- Serve a single retail bookstore outlet initially
- Basic computer literacy

CONSTRAINTS

- Fixed storage and performance
- Responsive design
- Dependence on network

RISKS

- Security vulnerabilities due to bugs or flaws
- Integration failures with third parties
- Performance degradation:

ASSUMPTIONS/CONSTRAINTS/RISKS

PART 3: OVERALL DESCRIPTION

GROUP 18

ARCHITECTURAL DESIGN

BACK TO AGENDA PAGE

The Business Logic Tier,

Empowered by Spring Boot, forms the core of the application where the business rules and application logic reside

The Presentation Tier,

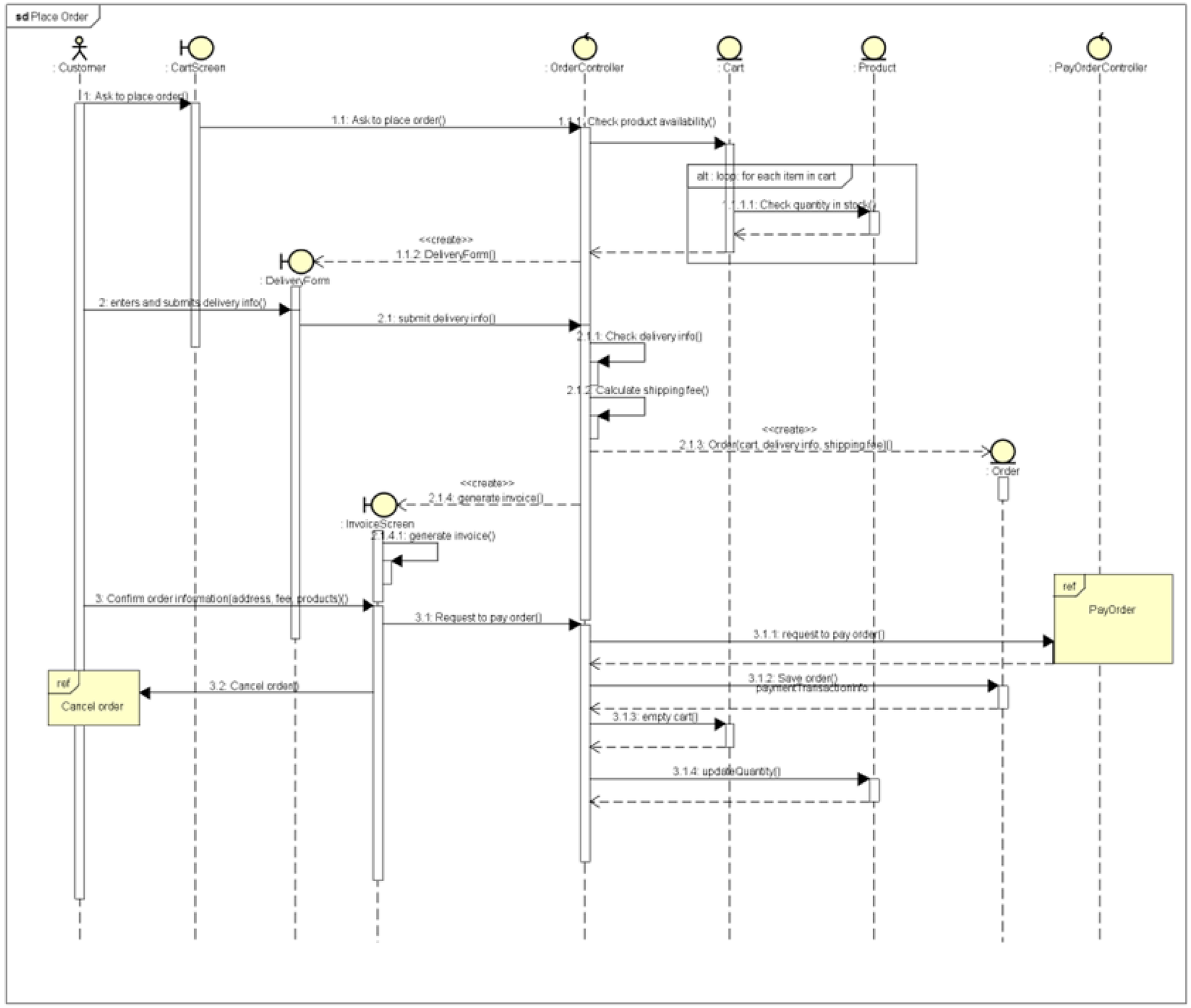
Implemented with ReactJS, serves as the user interface layer. It is responsible for rendering the visual components of the application, managing client-side logic, and facilitating user interactions

Data Access Tier,

Tier encompasses the database and the data management framework. This tier is responsible for the storage, retrieval, and manipulation of application data

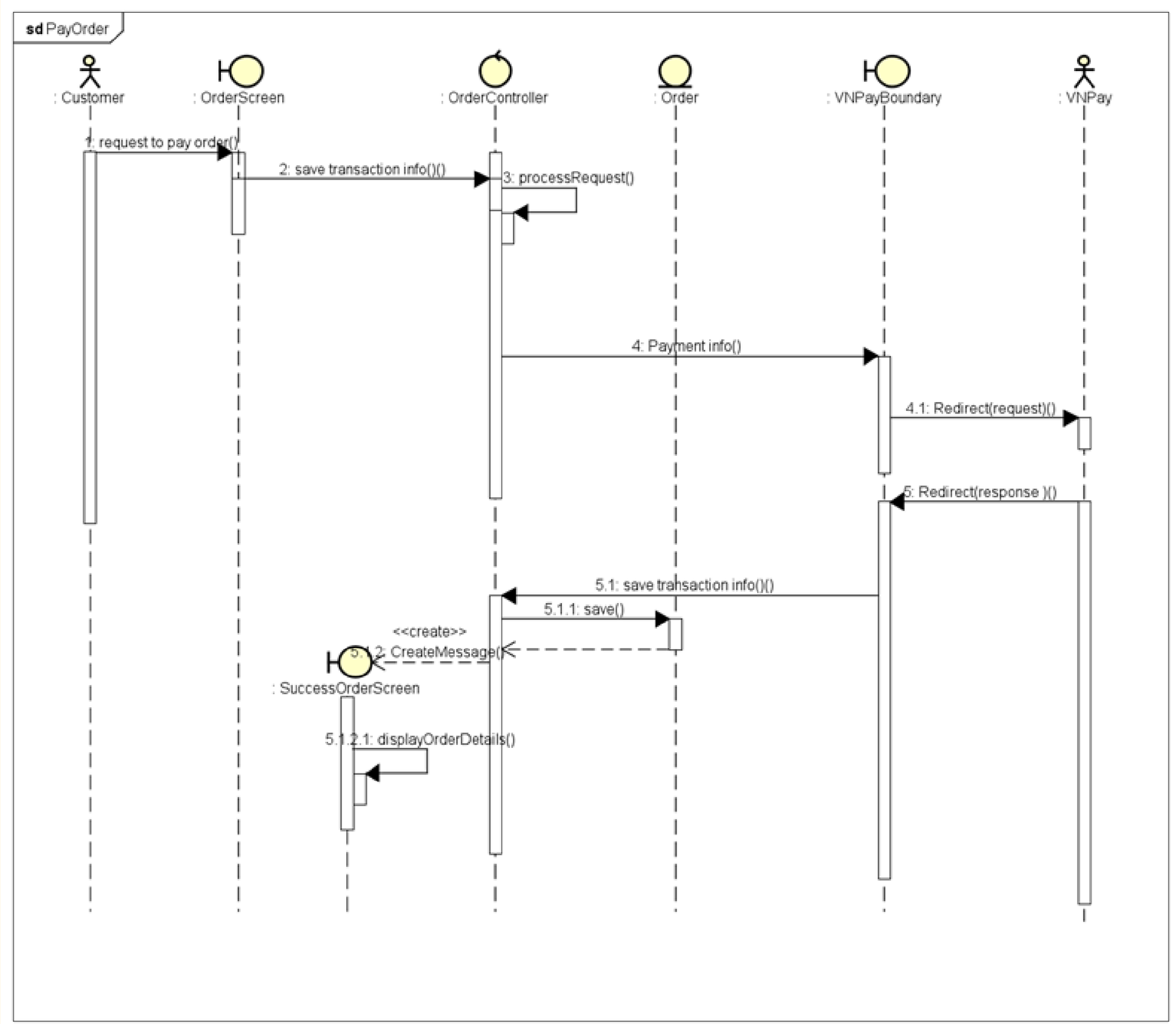
INTERACTION DIAGRAM

1. PLACE ORDER



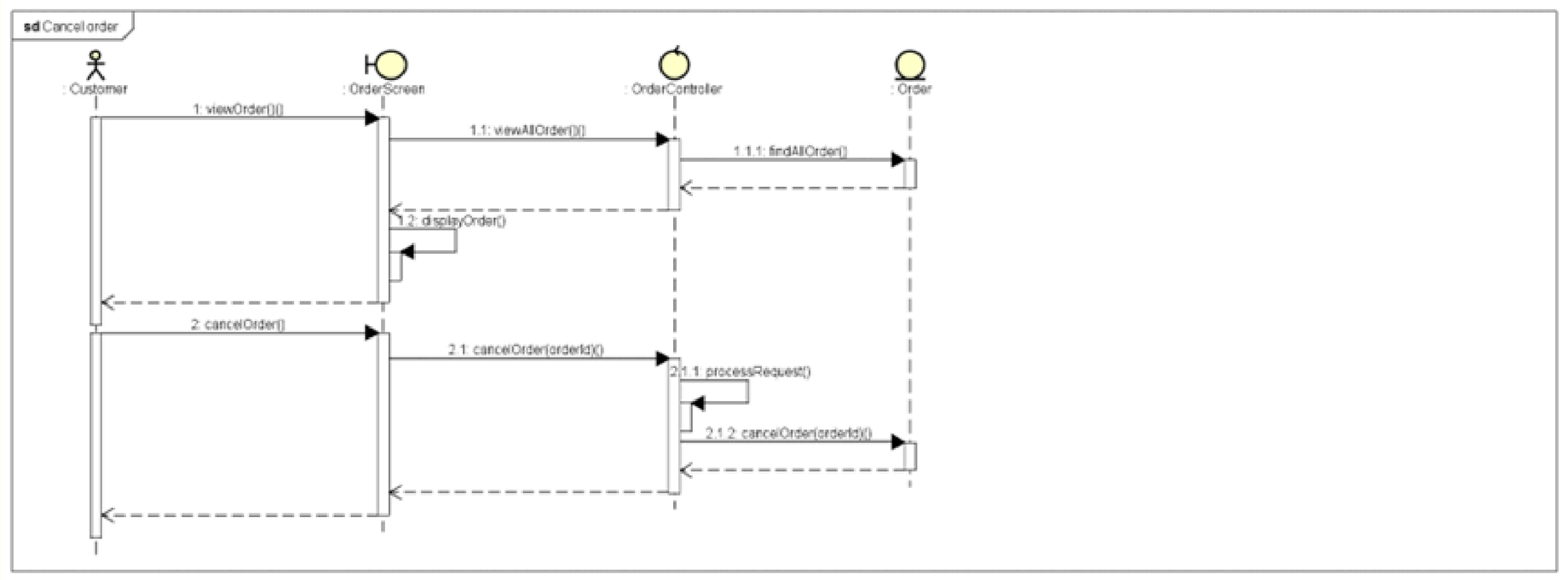
INTERACTION DIAGRAM

2. PAY ORDER



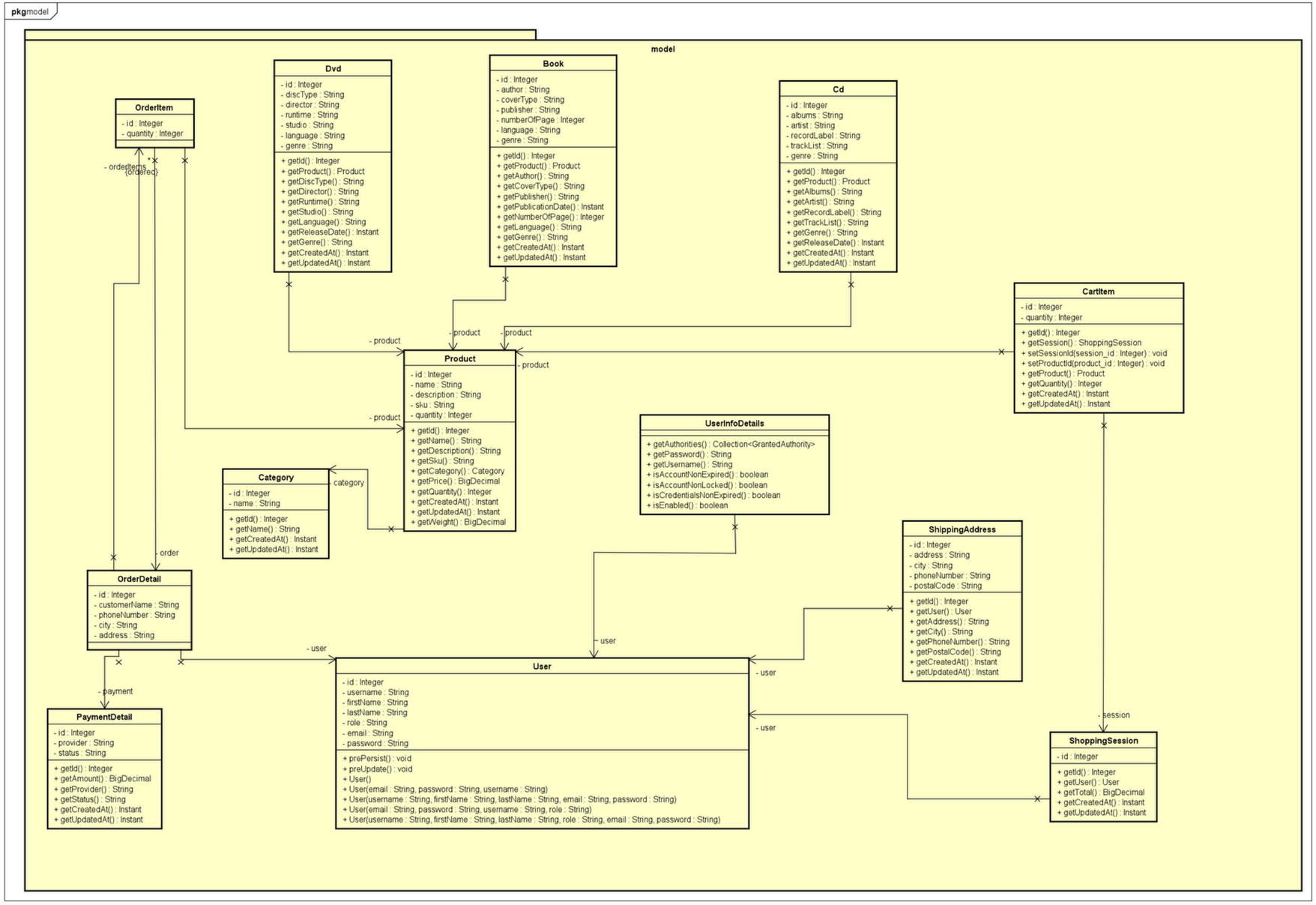
INTERACTION DIAGRAM

3. CANCEL ORDER



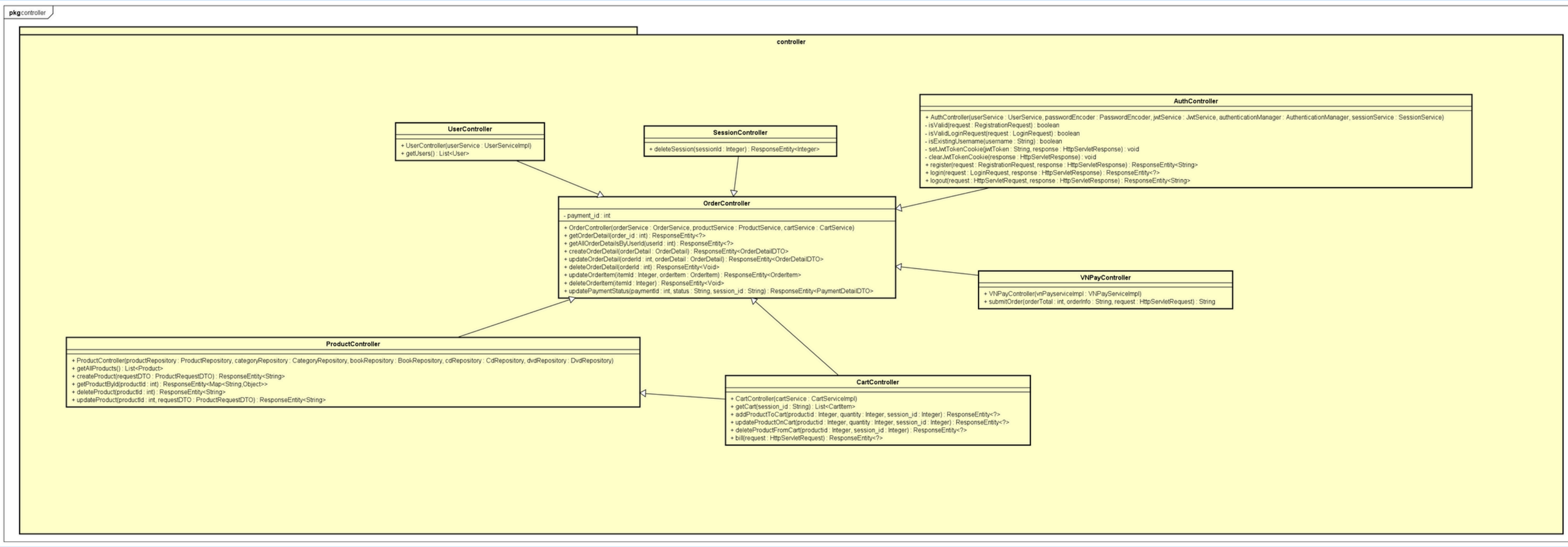
CLASS DIAGRAM

Model



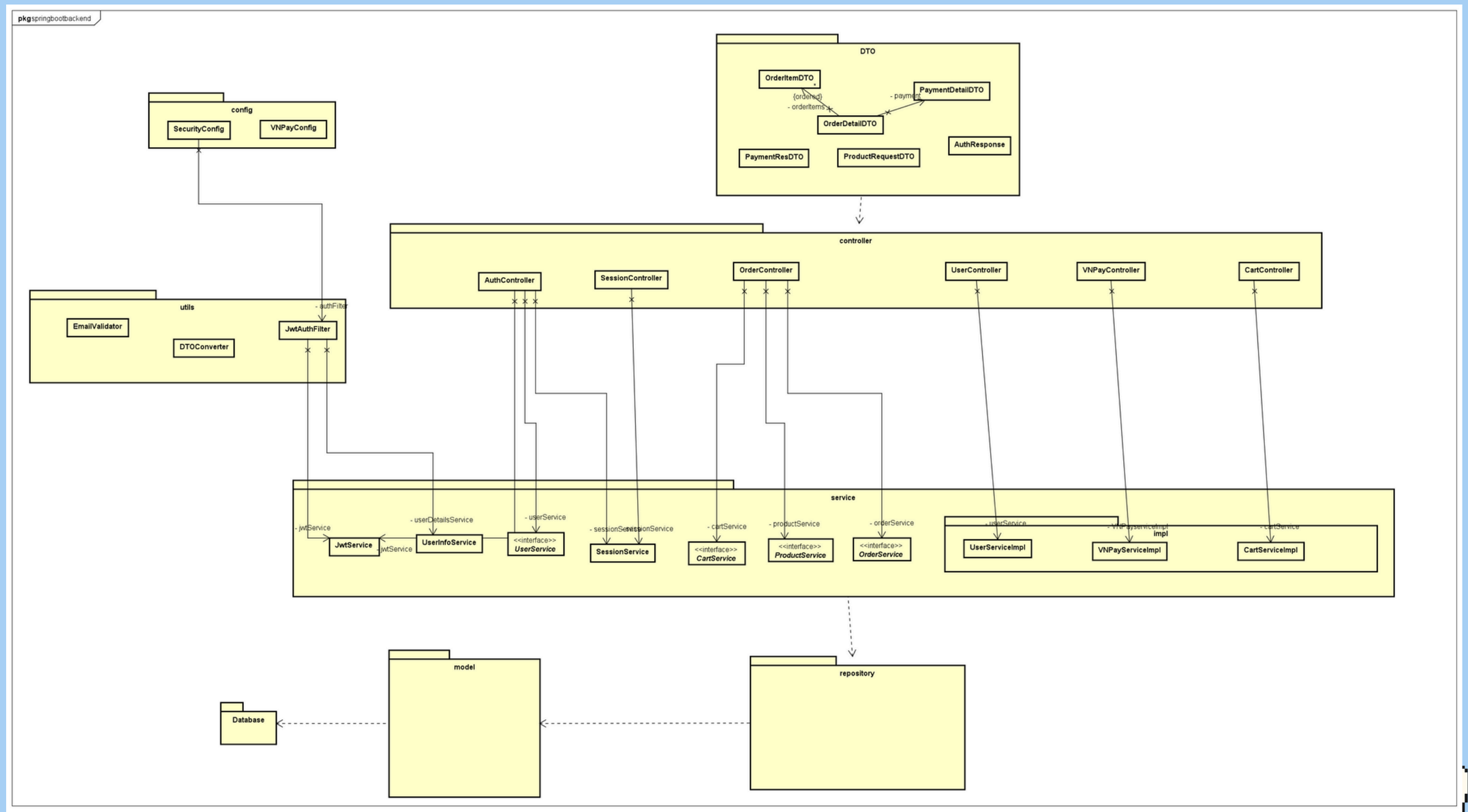
CLASS DIAGRAM

Controller



CLASS DIAGRAM

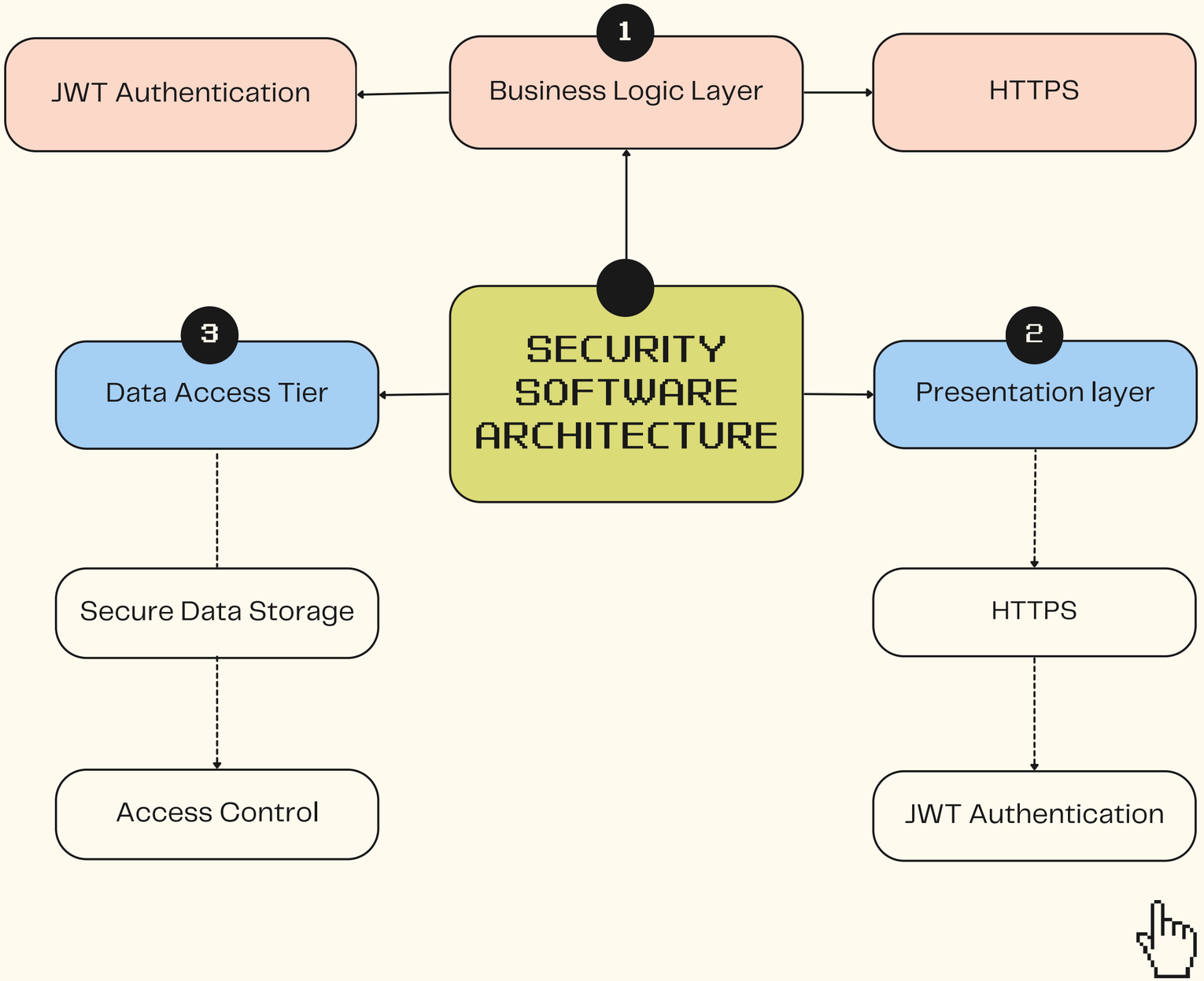
Unified



1. ReactJS and Spring Boot, focusing on HTTPS and JWT (JSON Web Tokens) provides a robust security foundation.

2. We aim to secure the system by implementing HTTPS for secure communication and JWT for secure, stateless authentication

Security



PART 4: DETAILED DESIGN

GROUP 18

1. USER INTERFACE DESIGN

Display

- Colors: 16,777,216 colors.
- Resolution:
  XS: Up to 576px
  SM: 576px to 768px
  MD: 768px to 992px
  LG: 992px to 1200px
  XL: 1200px and above

Screen

- Button Location: Bottom center (Submit, Cancel, Back, etc.)
- Messages Location: Top-right to bottom
- Title Location: Top center
- Alphanumeric Consistency: Thousand separator comma; strings include characters, digits, commas, dots, spaces, underscores, hyphens

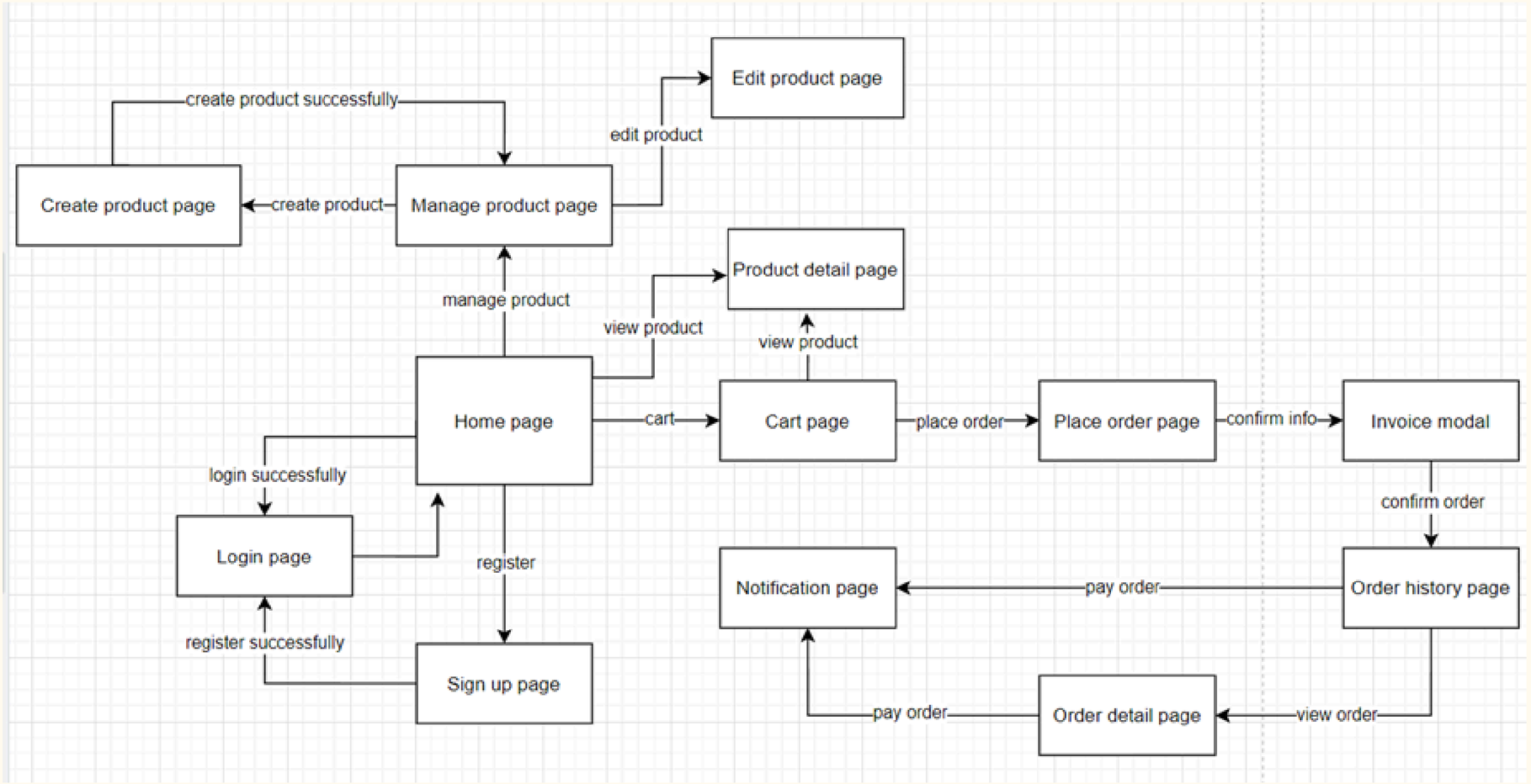
Control

- Text Size: Medium (14px), Headlines (24px)
- Font: Segoe UI
- Color: #000000
- Input Check: Validate for non-empty and correct format (email, phone)
- Focus Sequence: Logical tab order, separate screens, no stack frames
- Direct Input
- No Shortcuts: Navigation via buttons/links
- Back Buttons: Prominently placed
- Close Button: “Cancel” at bottom of screen

Error Handling

- Notifications: Inform users of problems with messages

SCREEN TRANSITION DIAGRAM

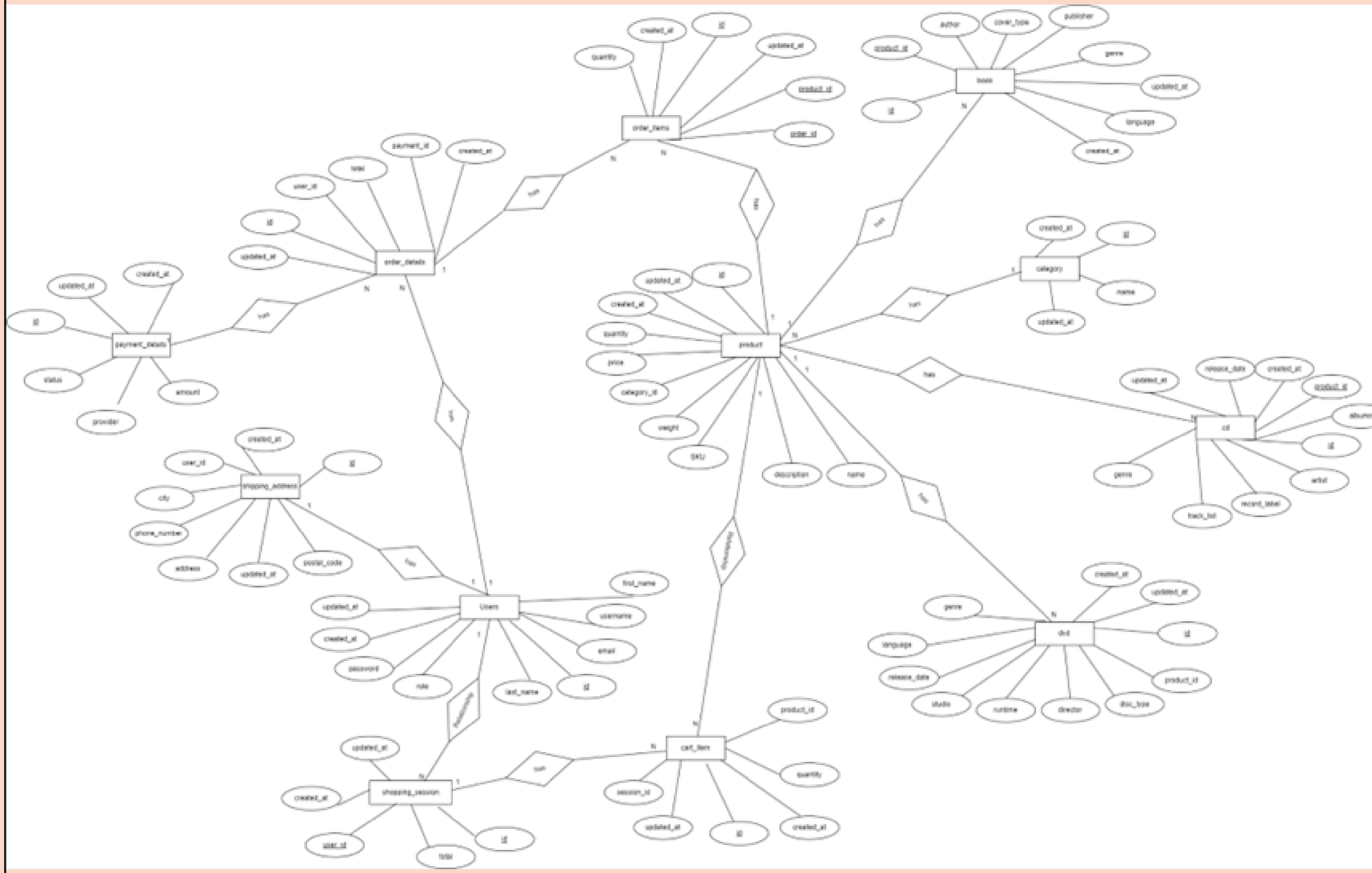


2. DATA MODELING

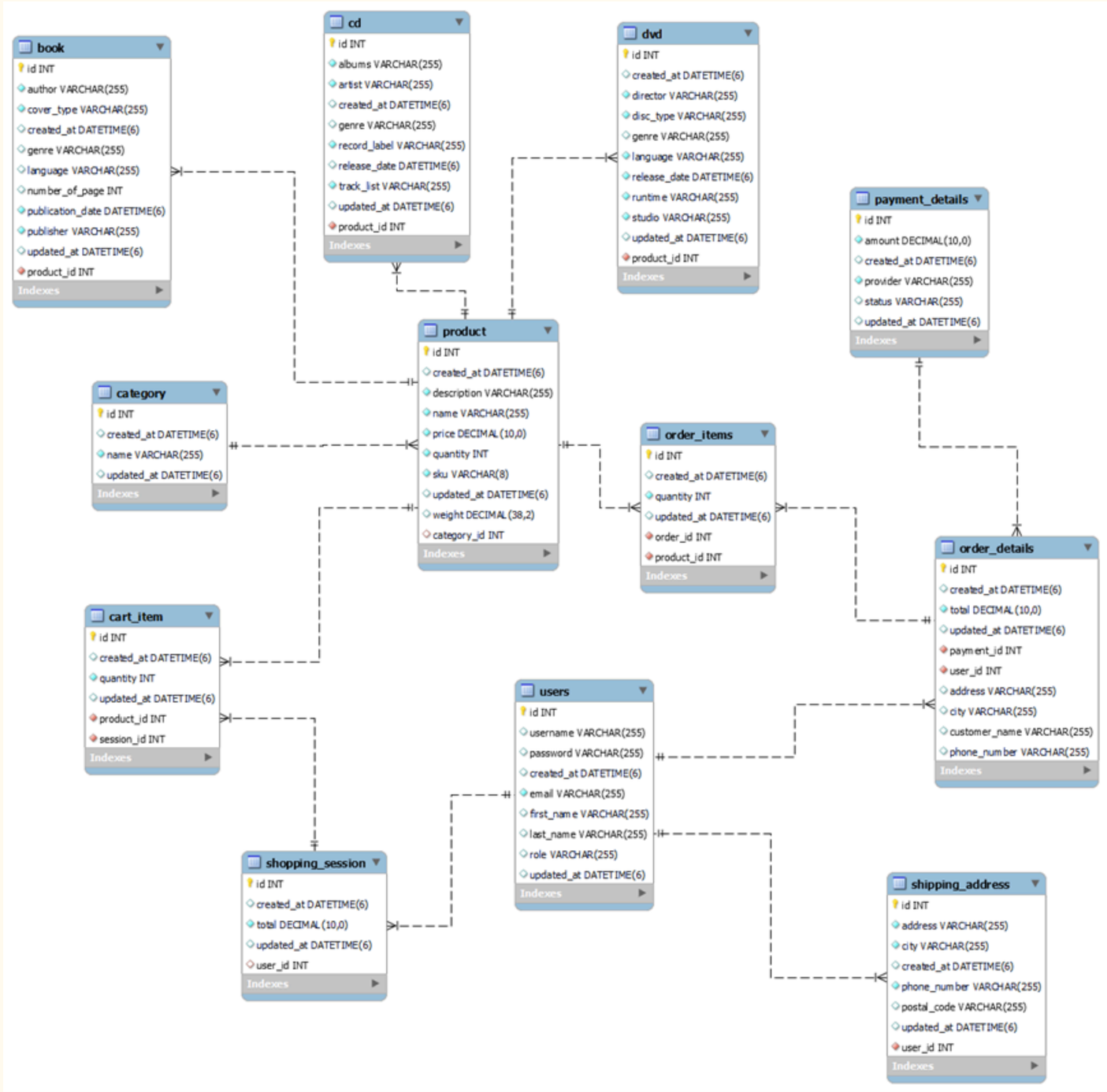
Relational database management system: MySQL

Perks: High reliability, scalable, robust, secured.

Entity Relation Diagram



DATABASE DESIGN



3. CLASS DESIGN

There are 7 main packages

Model

Represents the data structure that map to the database tables

Repository

Interacting with the data source and providing an abstraction layer for the model

Service

The core business logic of AIMS resides. It acts as a mediator between the controller and the repository

Controller

Handling incoming HTTP requests and mapping them to appropriate methods in the service layer

utils

Provide general-purpose functionality

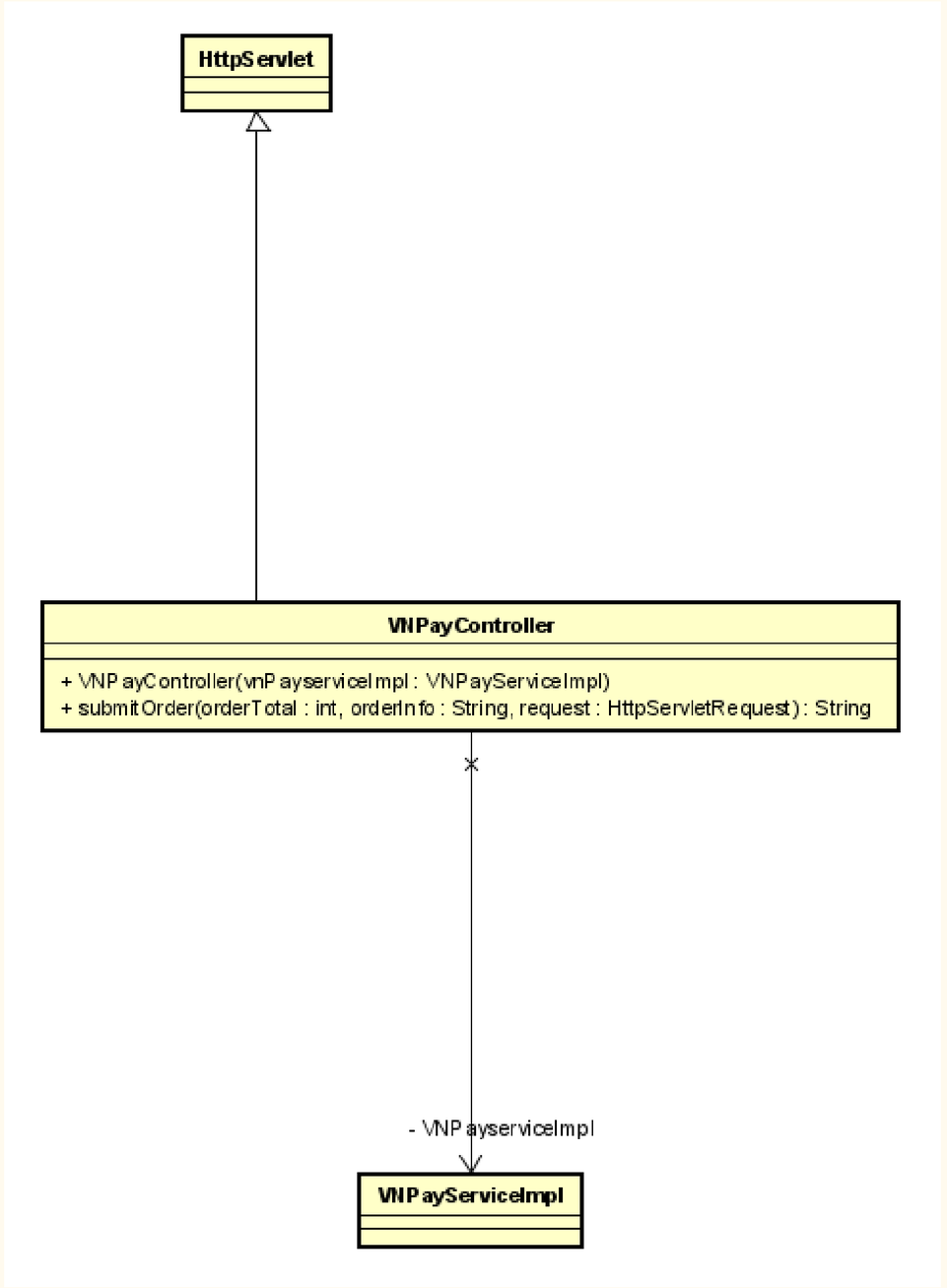
DTO

Data Transfer Object – transfer data between different layers

Request

It contains information about HTTPrequest

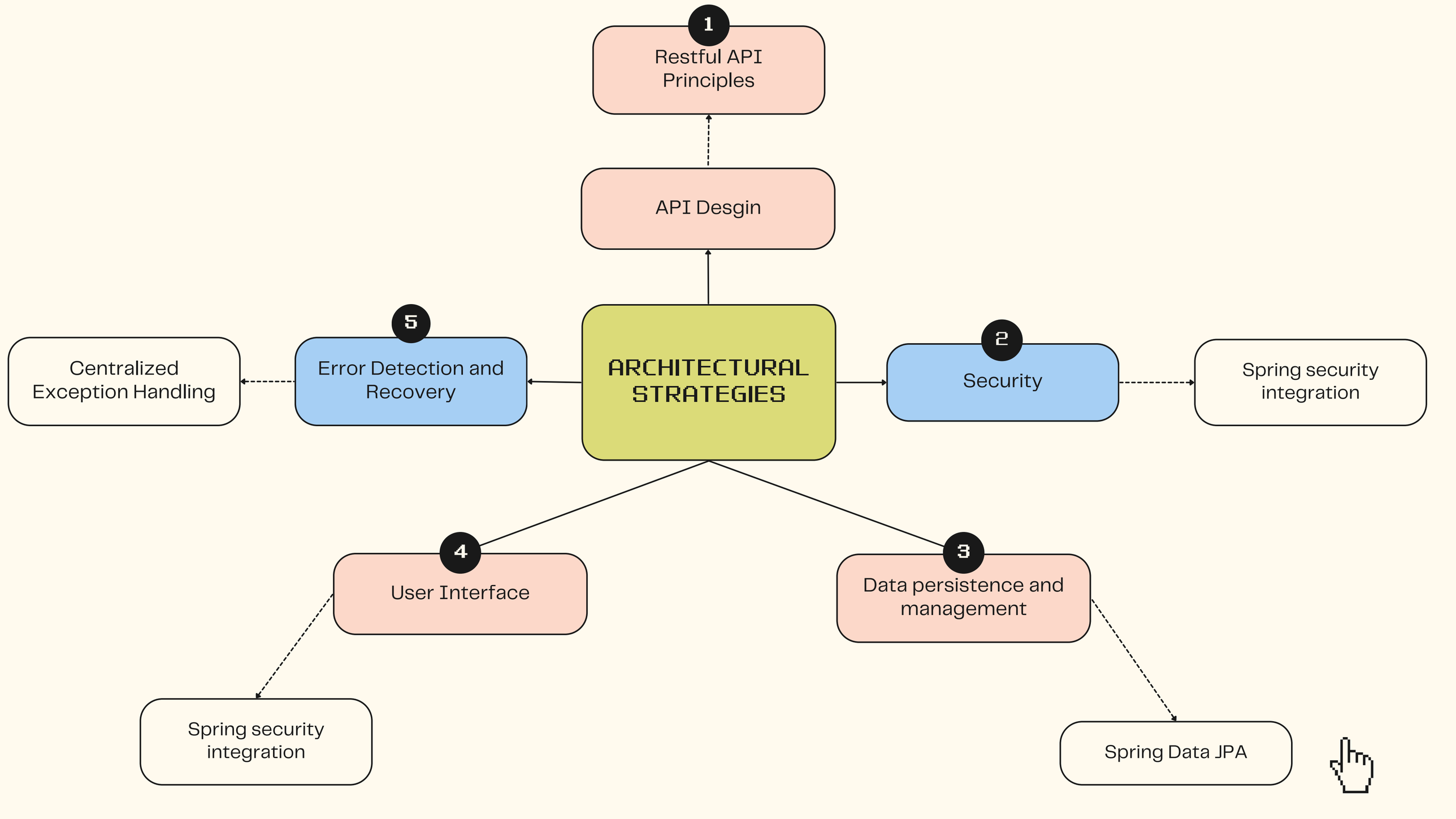
4. VNPAY SUBSYSTEM DESIGN



PART 5:

DESIGN CONSIDERATION

GROUP 18


1
Restful API Principles
API Desgin
ARCHITECTURAL STRATEGIES
5
Error Detection and Recovery
Centralized Exception Handling
2
Security
Spring security integration
4
User Interface
Spring security integration
3
Data persistence and management
Spring Data JPA

COUPLING & COHESION

Search

Coupling

- Communication: Frontend and backend communicate through RESTful APIs, enabling loose coupling.
- Requests and Responses: Frontend sends HTTP requests, backend responds with JSON data.
- Flexibility: Changes to frontend or backend do not affect the other, provided API contract is maintained.
- API Testing: Used Postman to test and explore APIs, validate responses, and generate code snippets, ensuring correct API functionality and effective communication between frontend and backend.

Cohesion

- Service Responsibilities: Each service has a single, well-defined responsibility.
- Frontend: Handles user interactions and displays data.
- Backend: Processes data and provides it to the frontend via API.
- Maintainability: High level of cohesion allows for easier maintenance and modification of each service independently.
- Consistency: Used clear and consistent API contracts and kept service responsibilities well-defined and separate to maintain high cohesion.

DESIGN PATTERNS

1. SINGLETON PATTERN

The Singleton pattern ensures a class has only one instance and provides a global point of access to it. Spring Boot beans are singletons by default, promoting resource sharing and consistency.

=> **Spring Boot Singleton Beans**

DESIGN PATTERNS

2. DEPENDENCY INJECTION (DI) PATTERN

- Dependency Injection reduces the coupling between classes and promotes code reuse, testability, and flexibility. SpringBoot uses DI extensively to manage the lifecycle of beans and their dependencies. => **SpringBoot DI**

- React's Context API can be used for dependency injection, providing a way to pass data through the component tree without having to pass props down manually at every level. => **React Context API**

DESIGN PATTERNS

3. REPOSITORY PATTERN

- The Repository pattern abstracts the data layer, providing a clean API for data access and manipulation. It decouples the business logic from data access logic, making the system more modular and easier to test.

=> **Spring Data JPA Repositories**

DESIGN PATTERNS

4. STRATEGY PATTERN

- The Strategy pattern allows the definition of a family of algorithms, encapsulates each one, and makes them interchangeable. It promotes the Open/Closed Principle by enabling new strategies to be added without modifying existing code.

- React Custom Hooks: Custom hooks encapsulate reusable logic for state management and side effects. This makes the logic interchangeable and composable.

DESIGN PATTERNS

5. FACTORY PATTERN

- The Template Method pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. It allows subclasses to redefine certain steps of an algorithm without changing its structure.

- Abstract Service Methods: Base service classes can provide default implementations of certain methods, while allowing subclasses to override specific steps.

DESIGN PATTERNS

6. OBSERVER PATTERN

- The Observer pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. This is essential for managing state and rendering updates in user interfaces.

- Redux in React: Redux's store acts as the subject, and components subscribe to state changes. When actions are dispatched, the store notifies all subscribed components about the state updates.

DESIGN PATTERNS

7. COMMAND PATTERN

- The Command pattern encapsulates a request as an object, thereby allowing for parameterization of clients with queues, requests, and operations. This is particularly useful for implementing undoable operations and handling actions as objects.

- Redux Actions: Actions in Redux encapsulate all the information needed to perform a certain operation, and these actions are dispatched to modify the state.

DESIGN PATTERNS

8. TEMPLATE PATTERN

- The Template Method pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. It allows subclasses to redefine certain steps of an algorithm without changing its structure.

- Abstract Service Methods: Base service classes can provide default implementations of certain methods, while allowing subclasses to override specific steps.

Thank You

For Listening!