EGAY

EGAY.com/anh-ne-nong non

E-Auction-Site

Present by: 99% Gambler

Group member

Lê Hưng - ITCSIU22271

Lê Nhật Anh - ITCSIU22254

Nguyễn Quốc Trung-ITITIU22171

Đàm Nguyễn Trọng Lễ-ITITIU22240

OUTLINE

01

Overview

02

Data

03

Auction+Demo

04

Summary

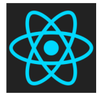
01

Overview

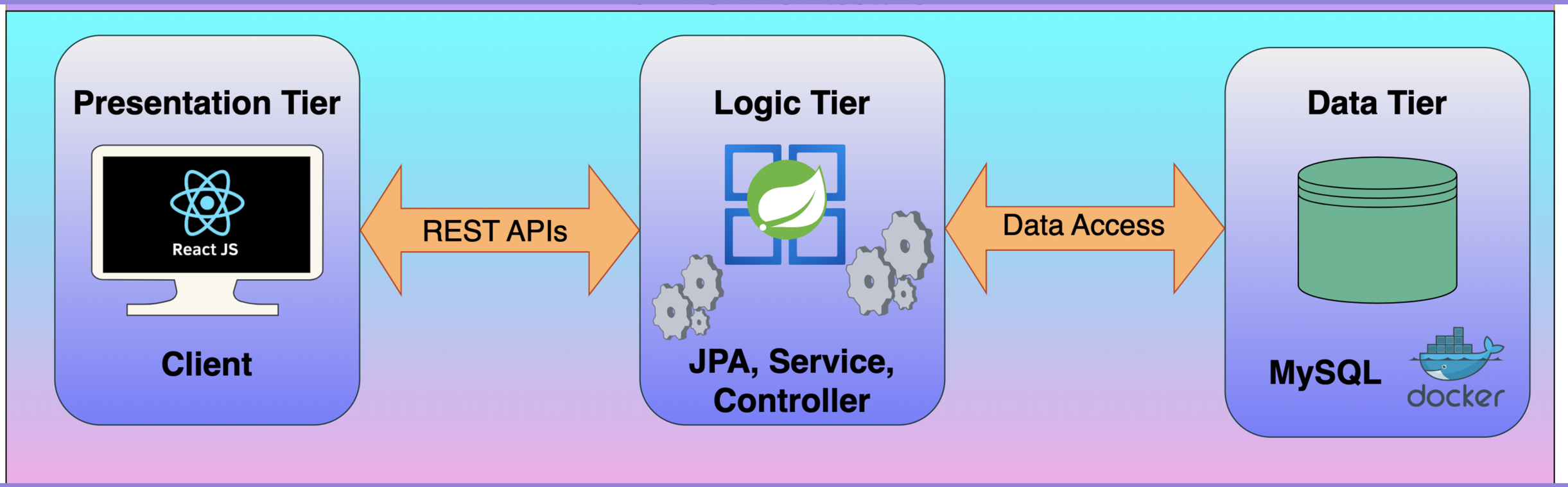
EGAY marketplace

- An e-commerce platform with both direct-buy and auction features, inspired by eBay.

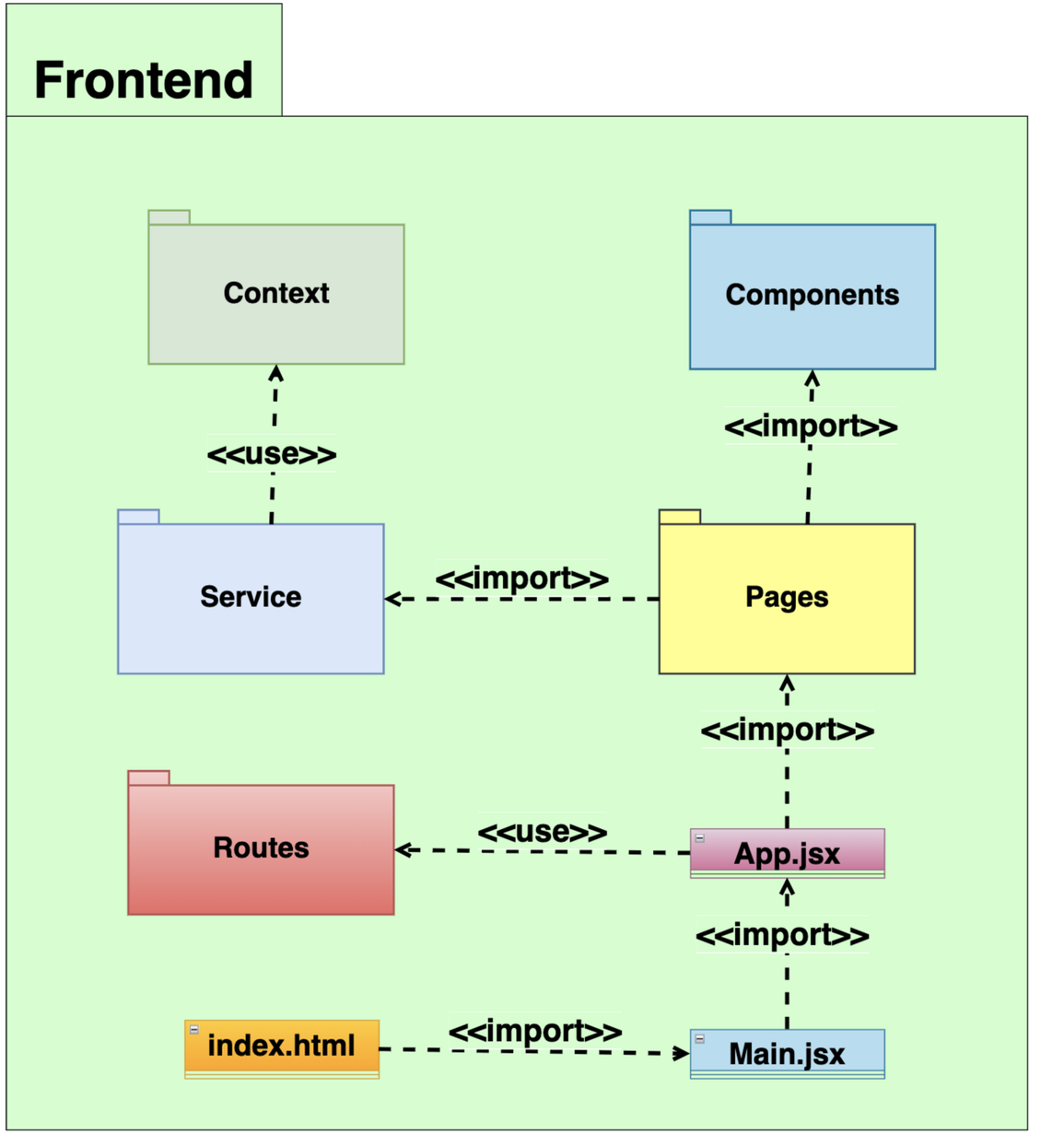
- Tech Stack

- Frontend:  React +  Vite +  Tailwind CSS
- Backend:  Spring Boot
- Real-time:  WebSocket
- Database:  MySQL
- Testing:  Postman

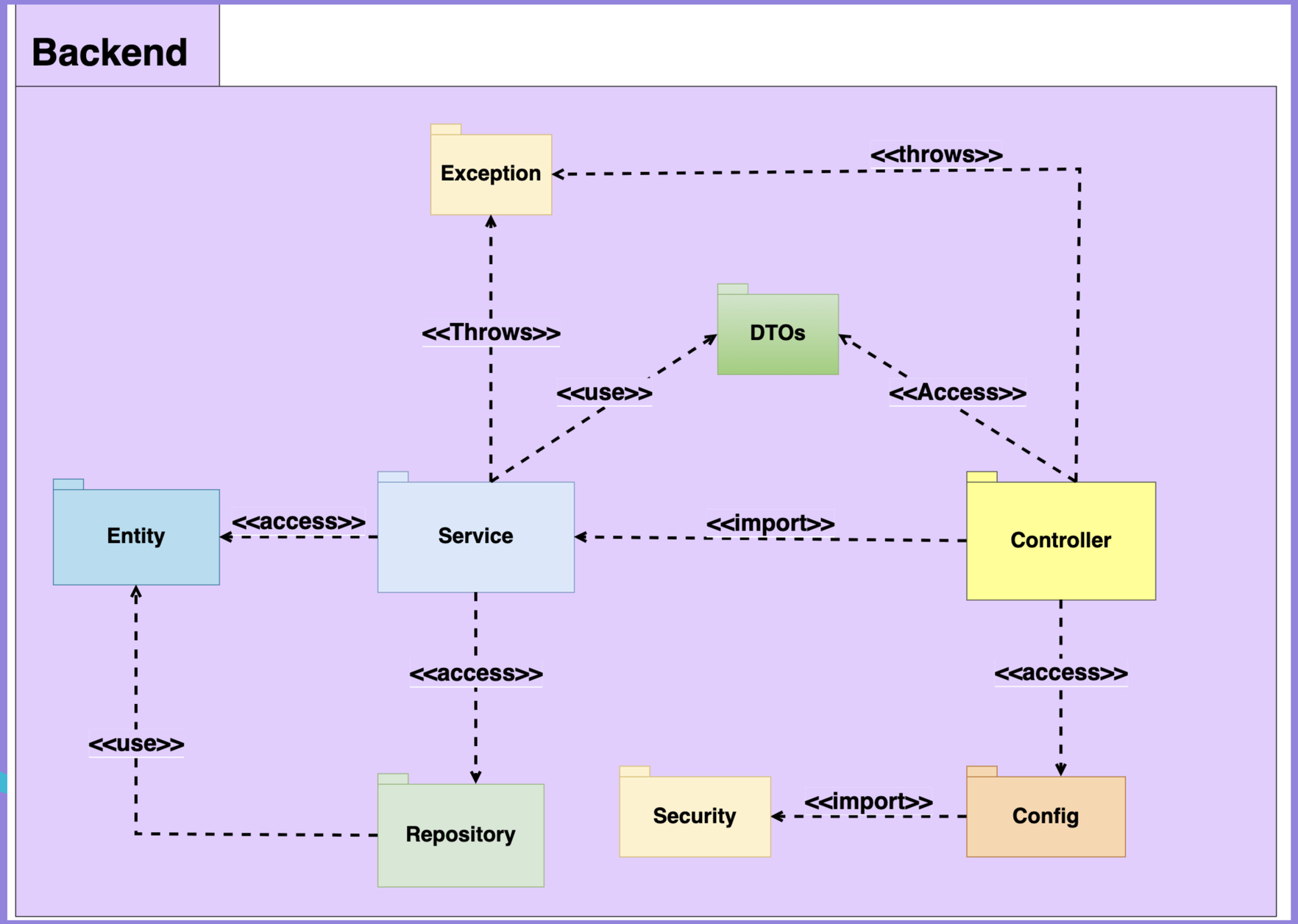
3-tier Architecture



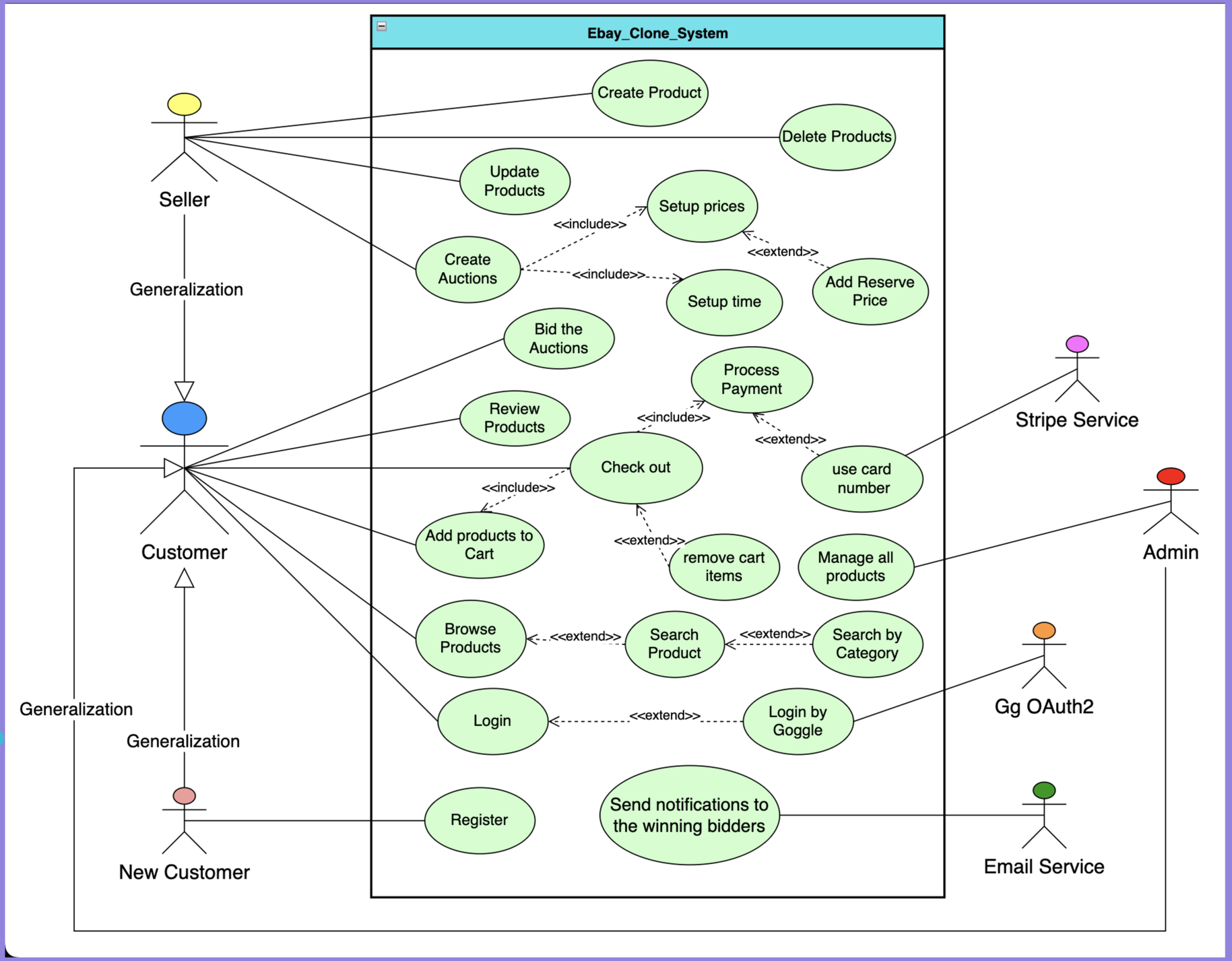
Frontend Package Diagram



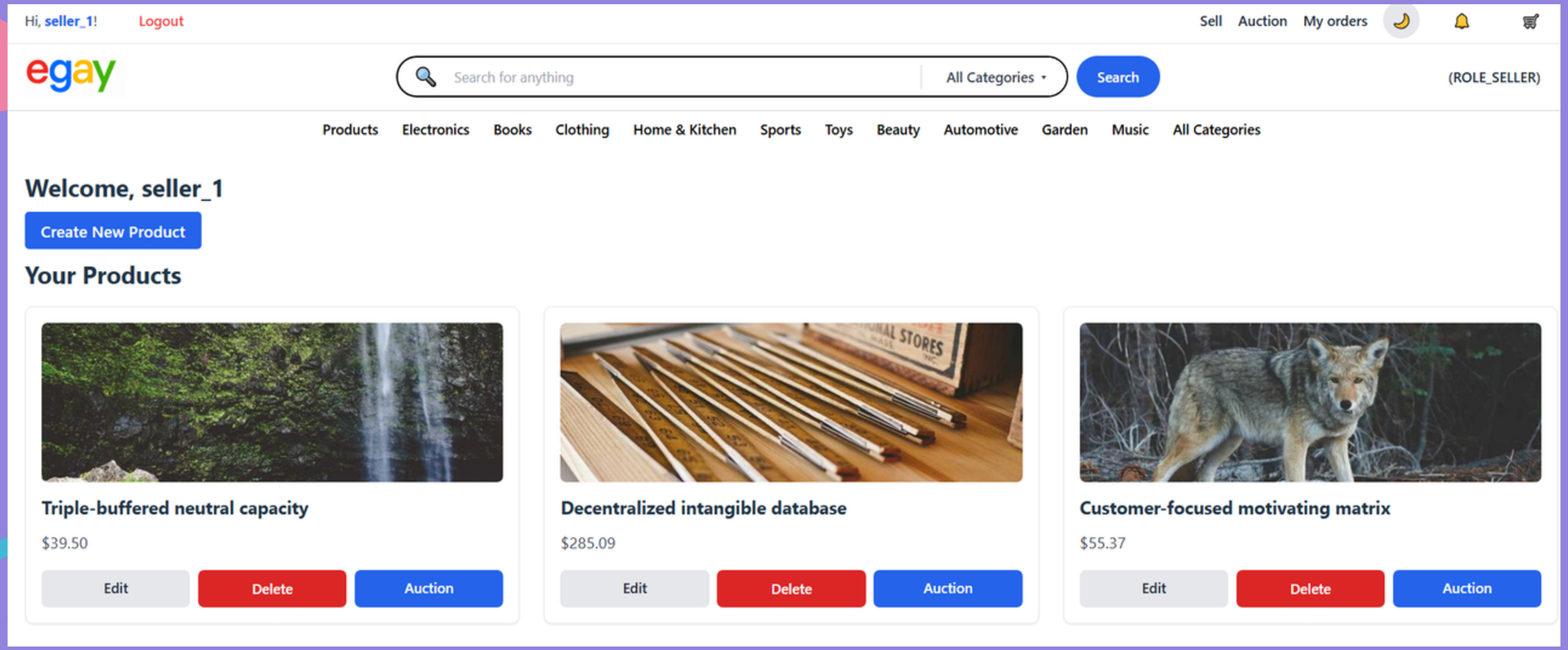
Backend-Package Diagram



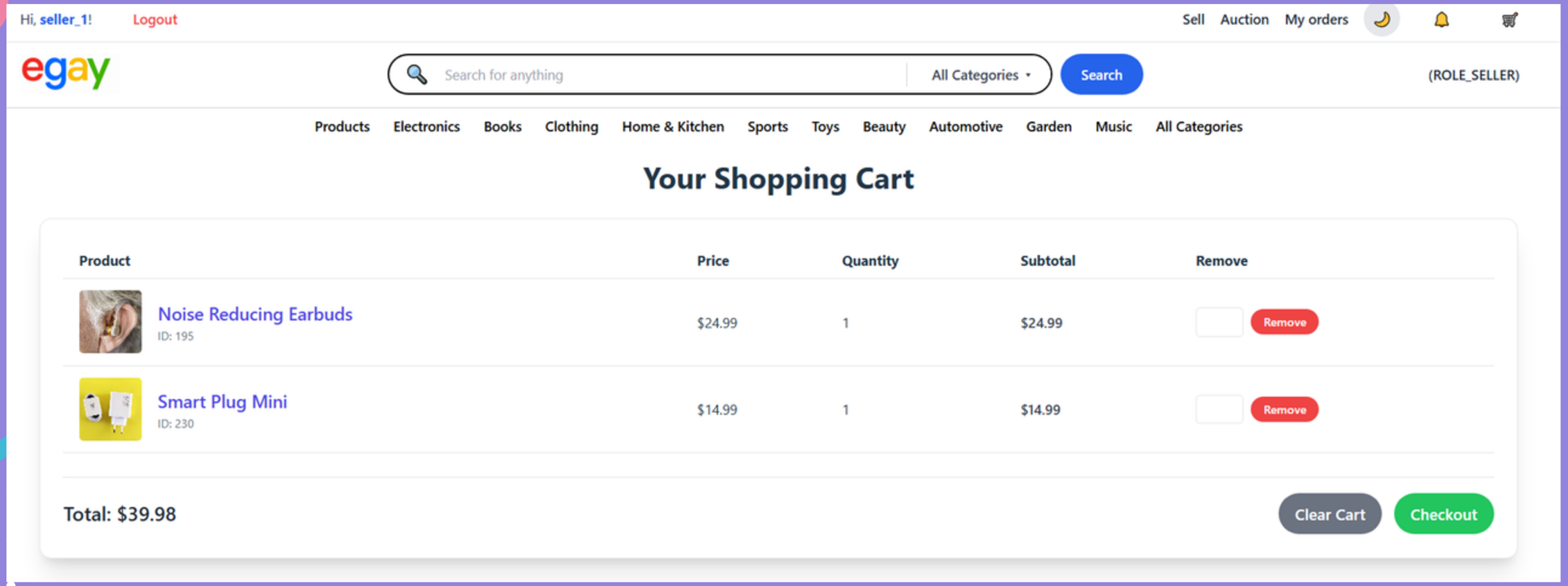
Use case Diagram



Seller Capabilities



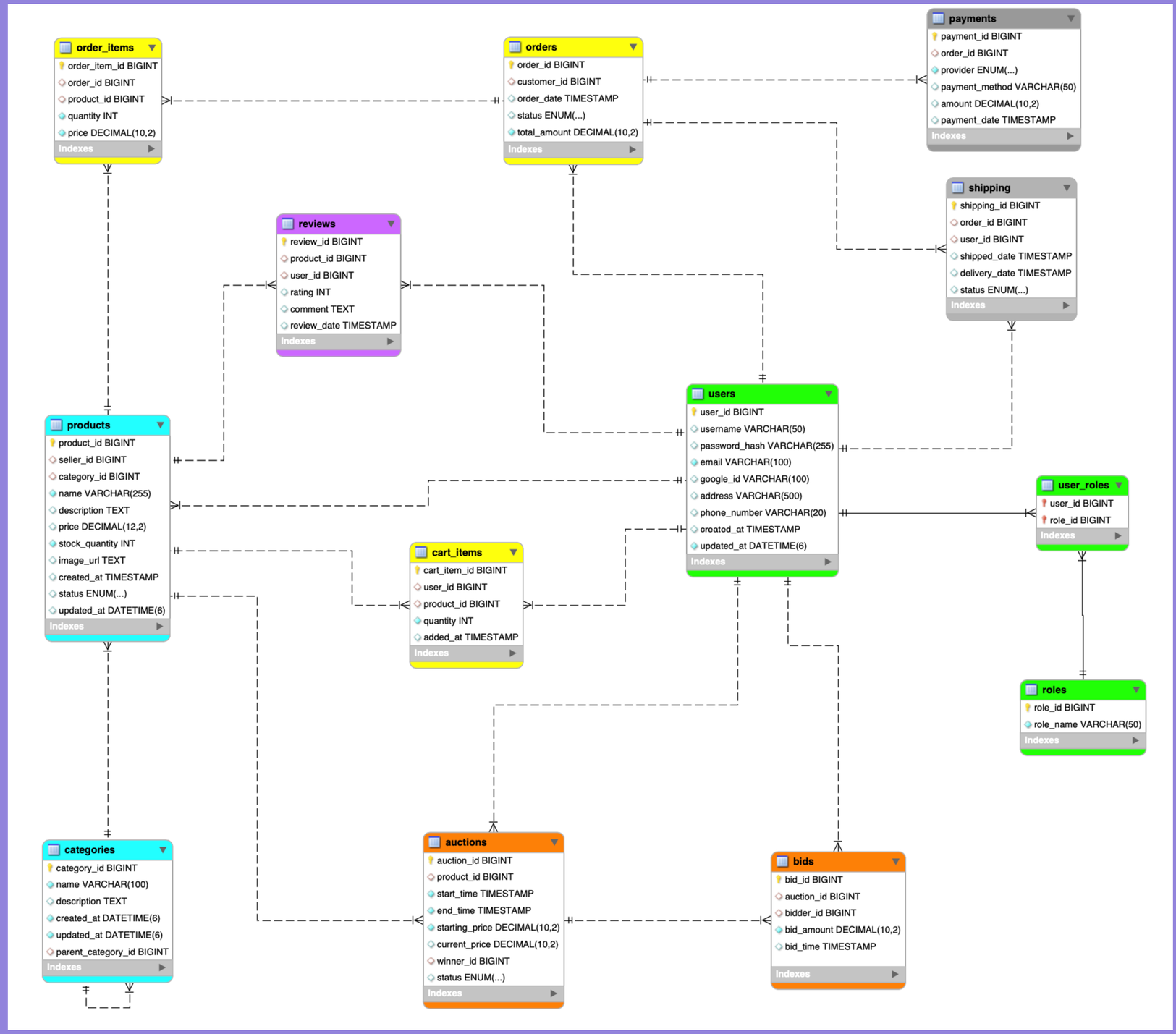
Buyer Experience



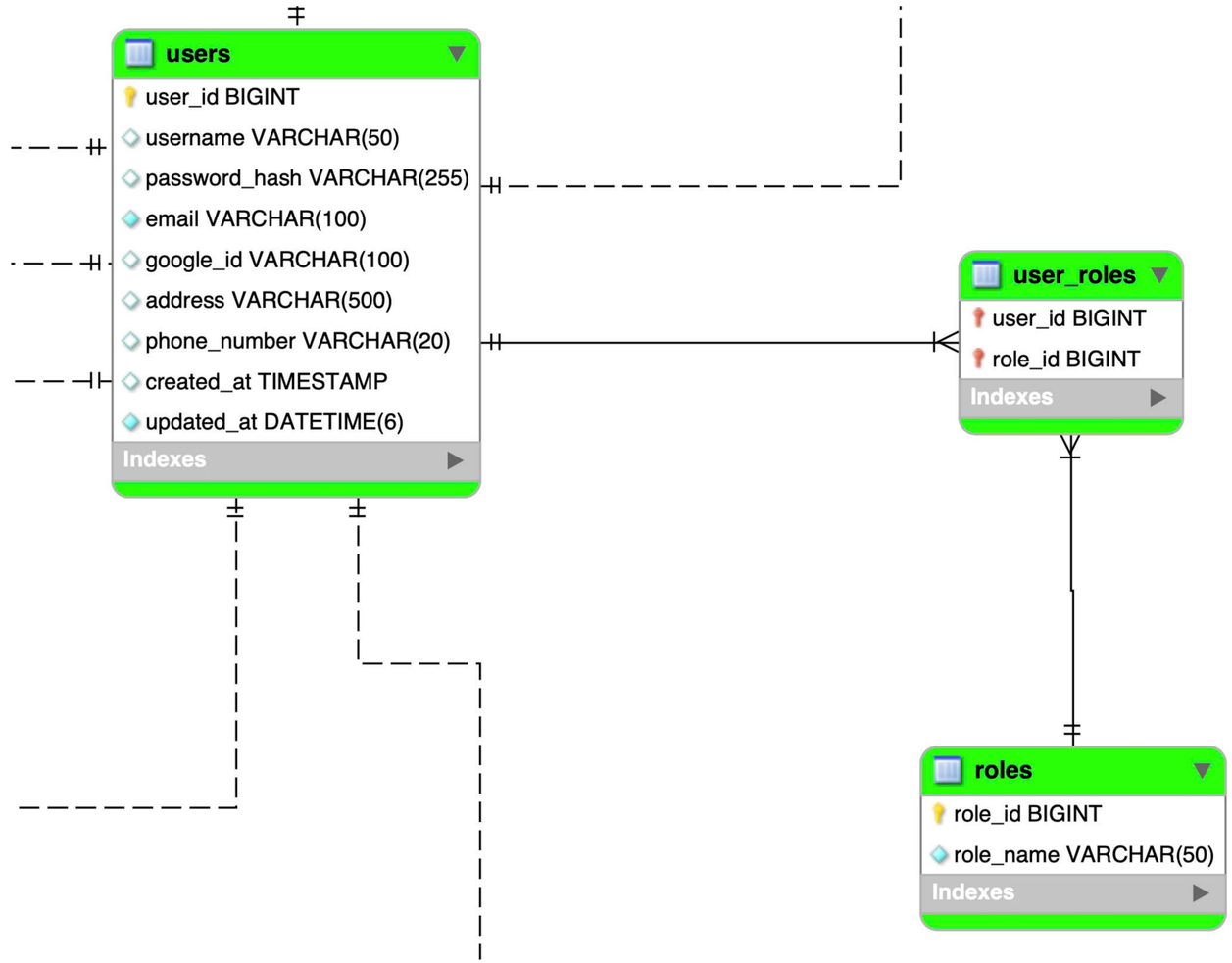
03

Database

Database Overview

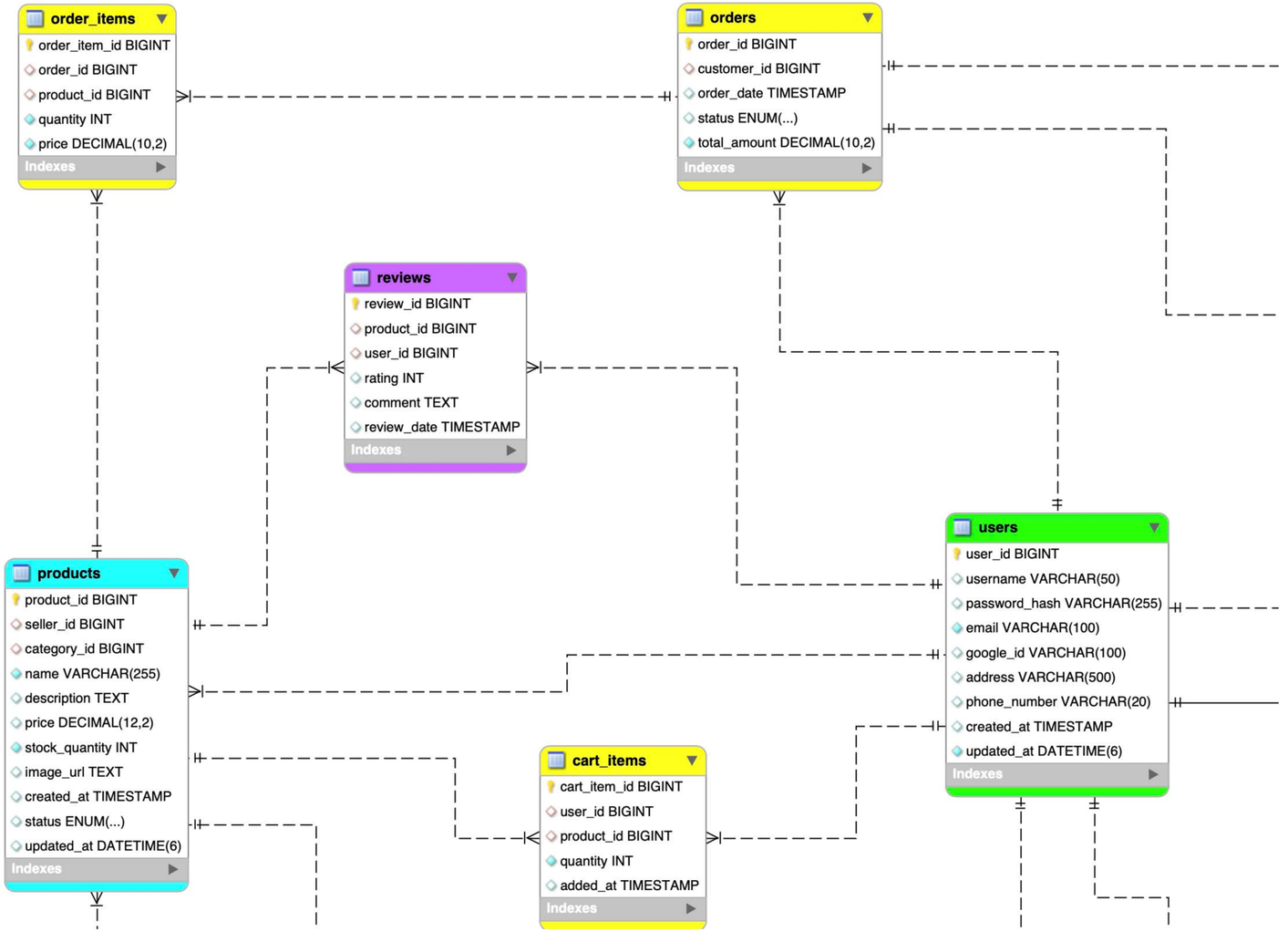


User Level Design:



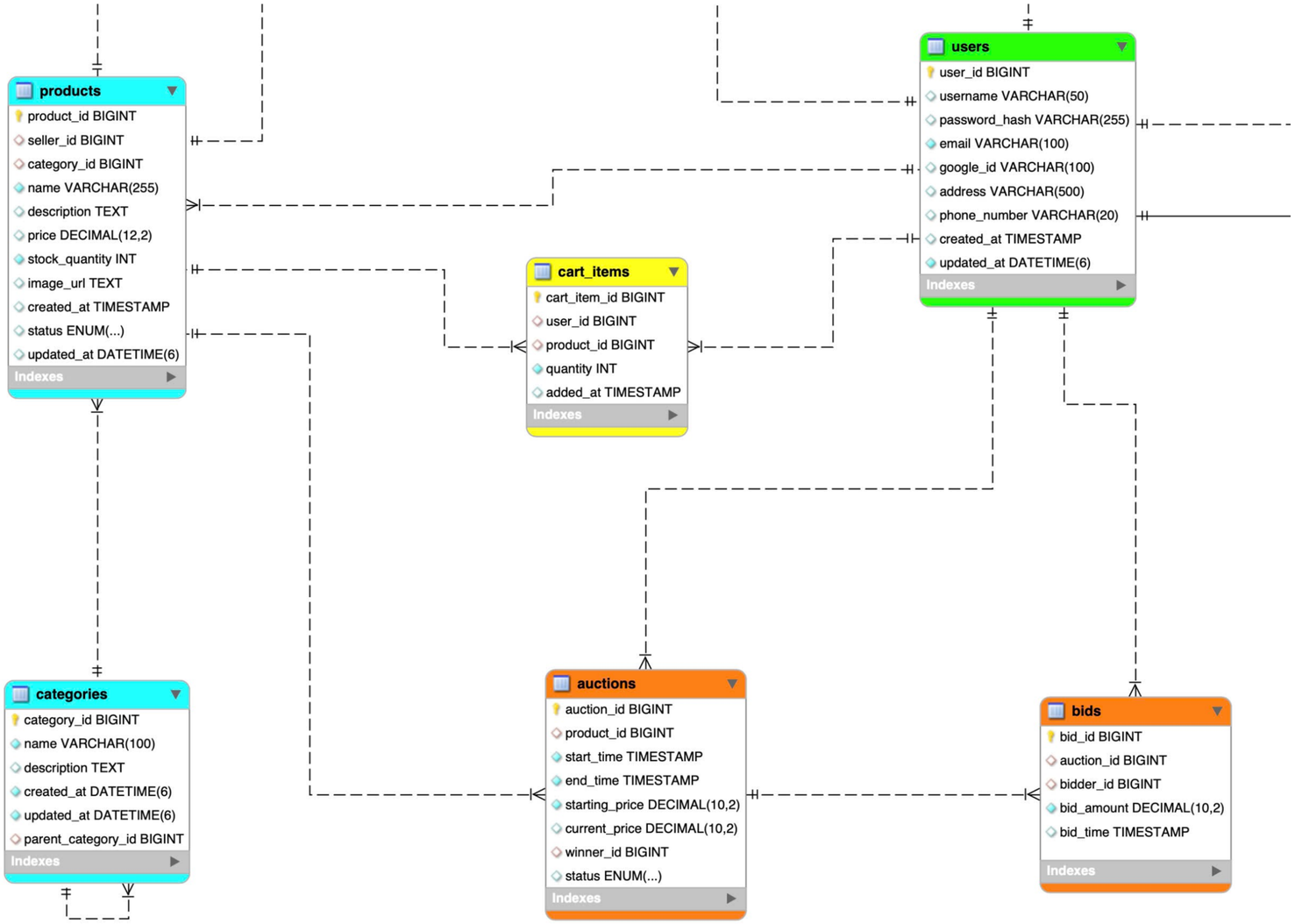
Separate tables user and role --> each user can has multiple roles

Cart and Order design

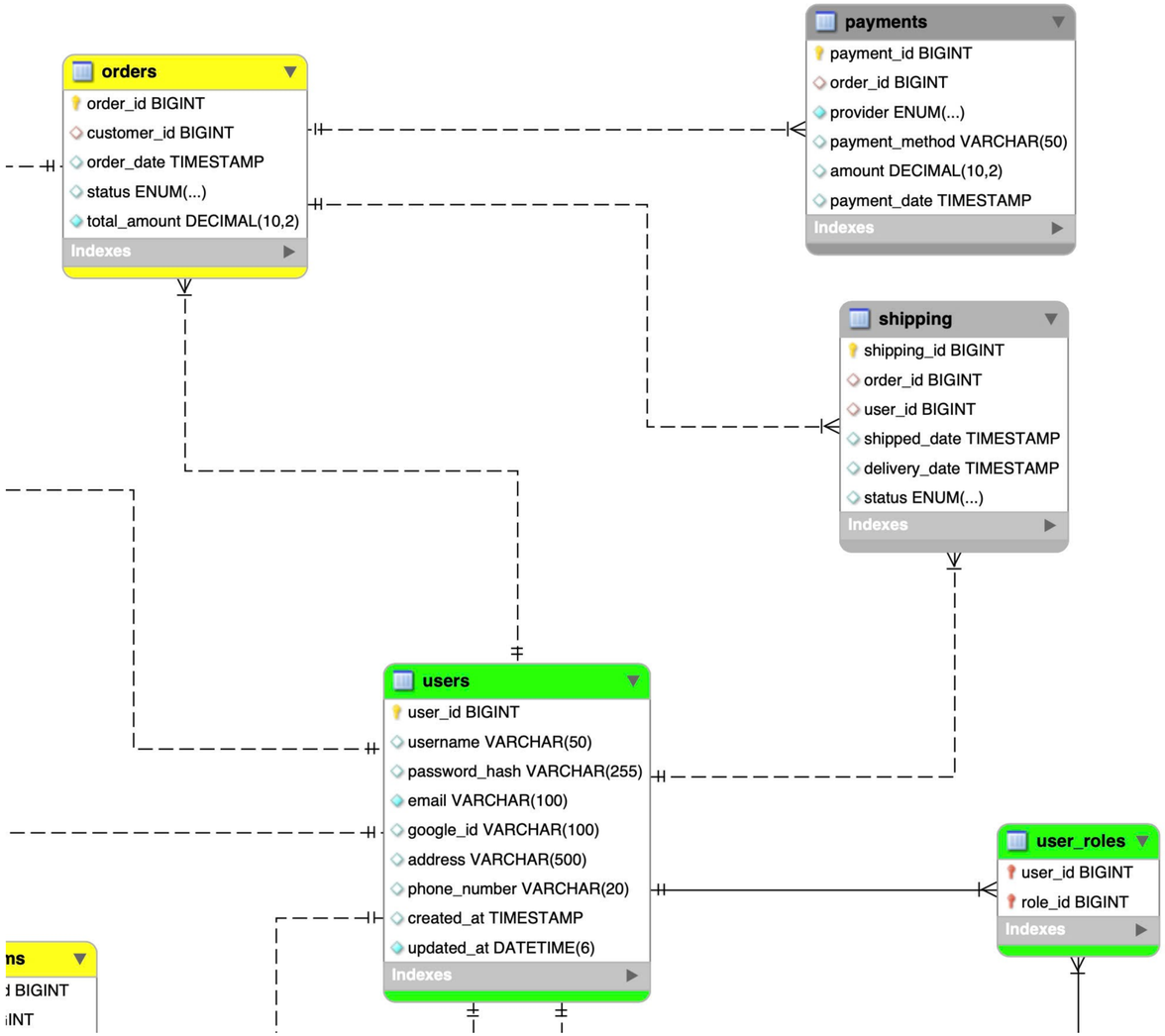


After a customer confirmed, the cart item will be changed into order item

Product and Auction Design



Payment and Shipping Design



04

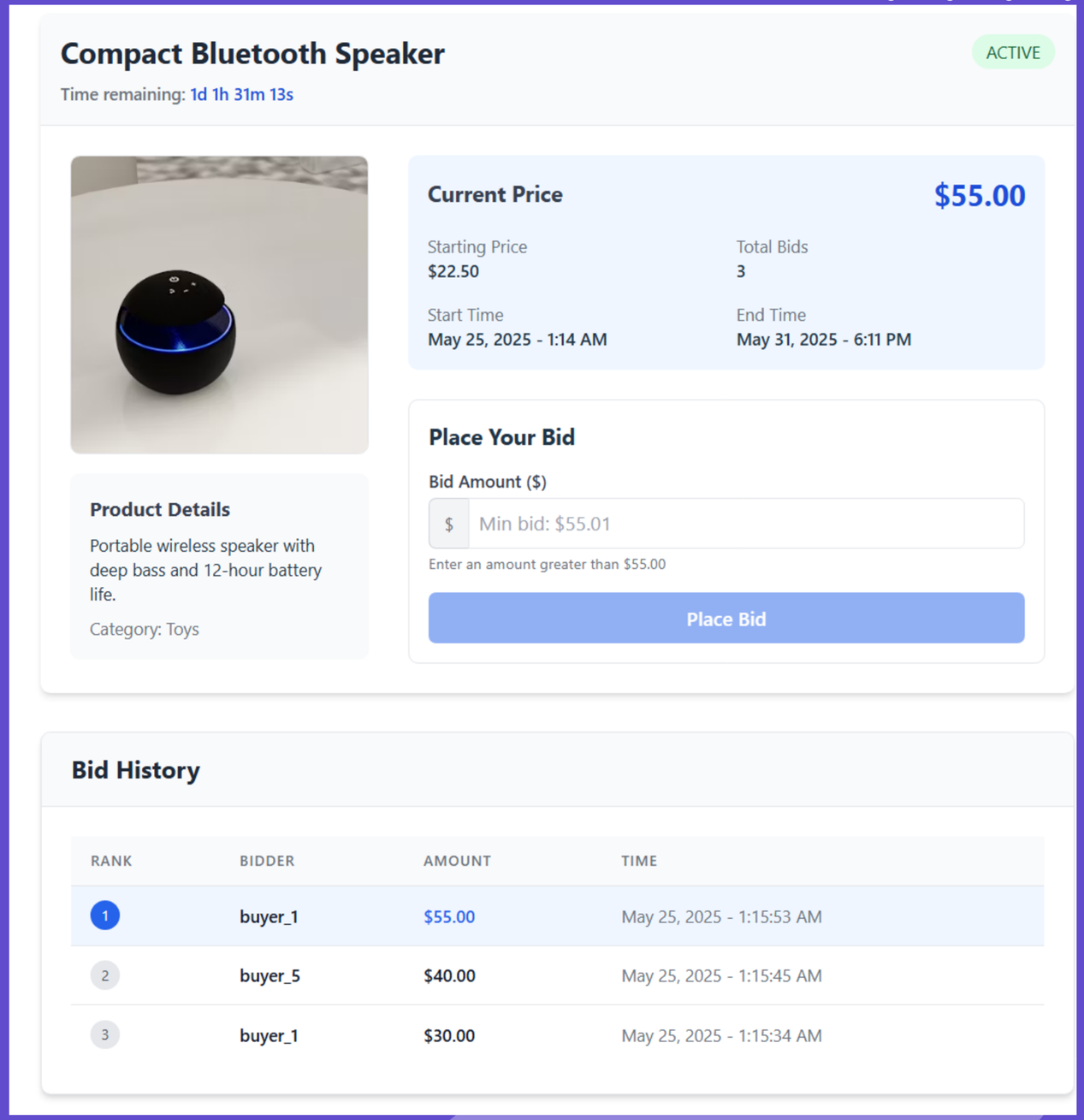
Auction + Demo

AUCTION

- Real-time bidding via WebSocket

- All users see updates instantly

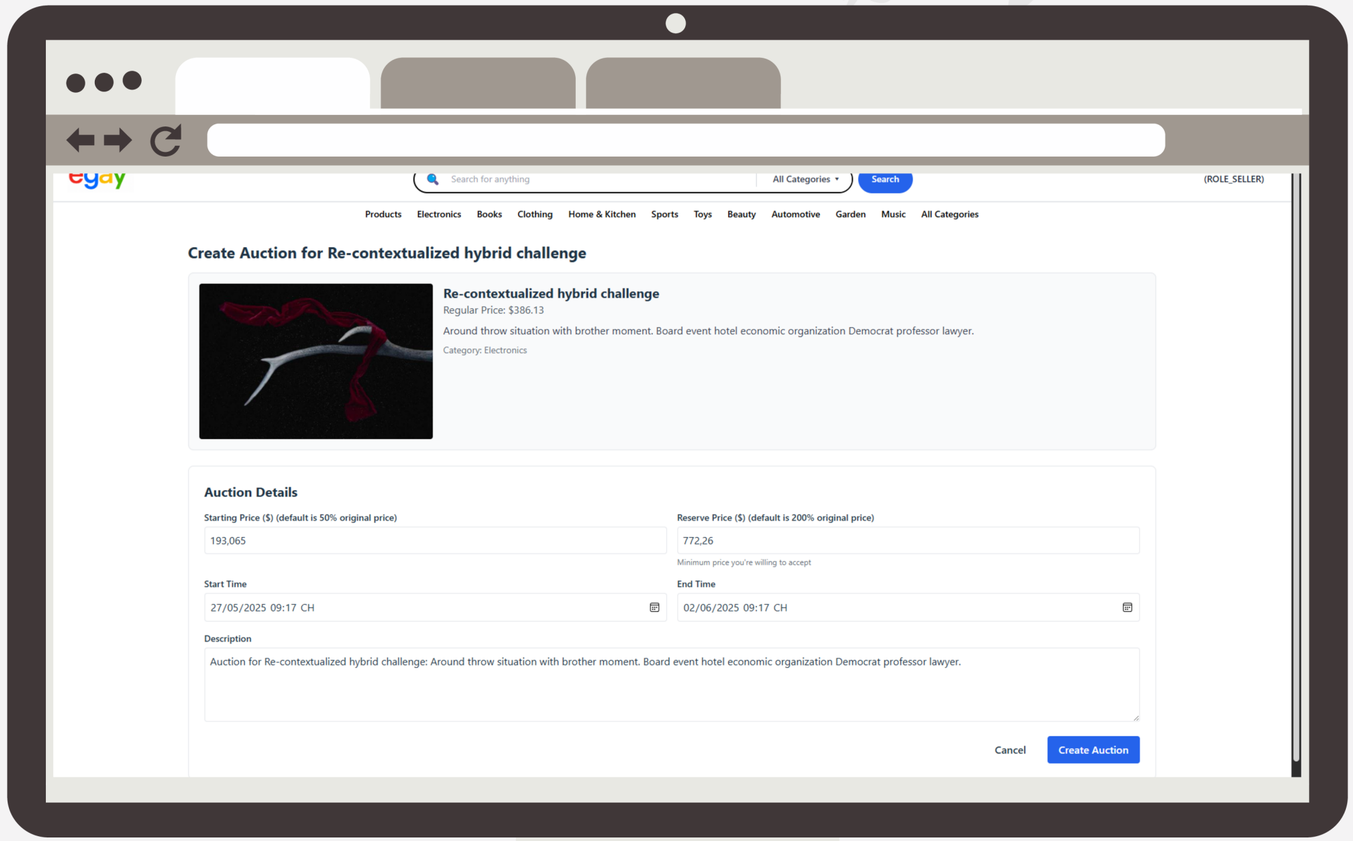
- Makes shopping competitive and exciting

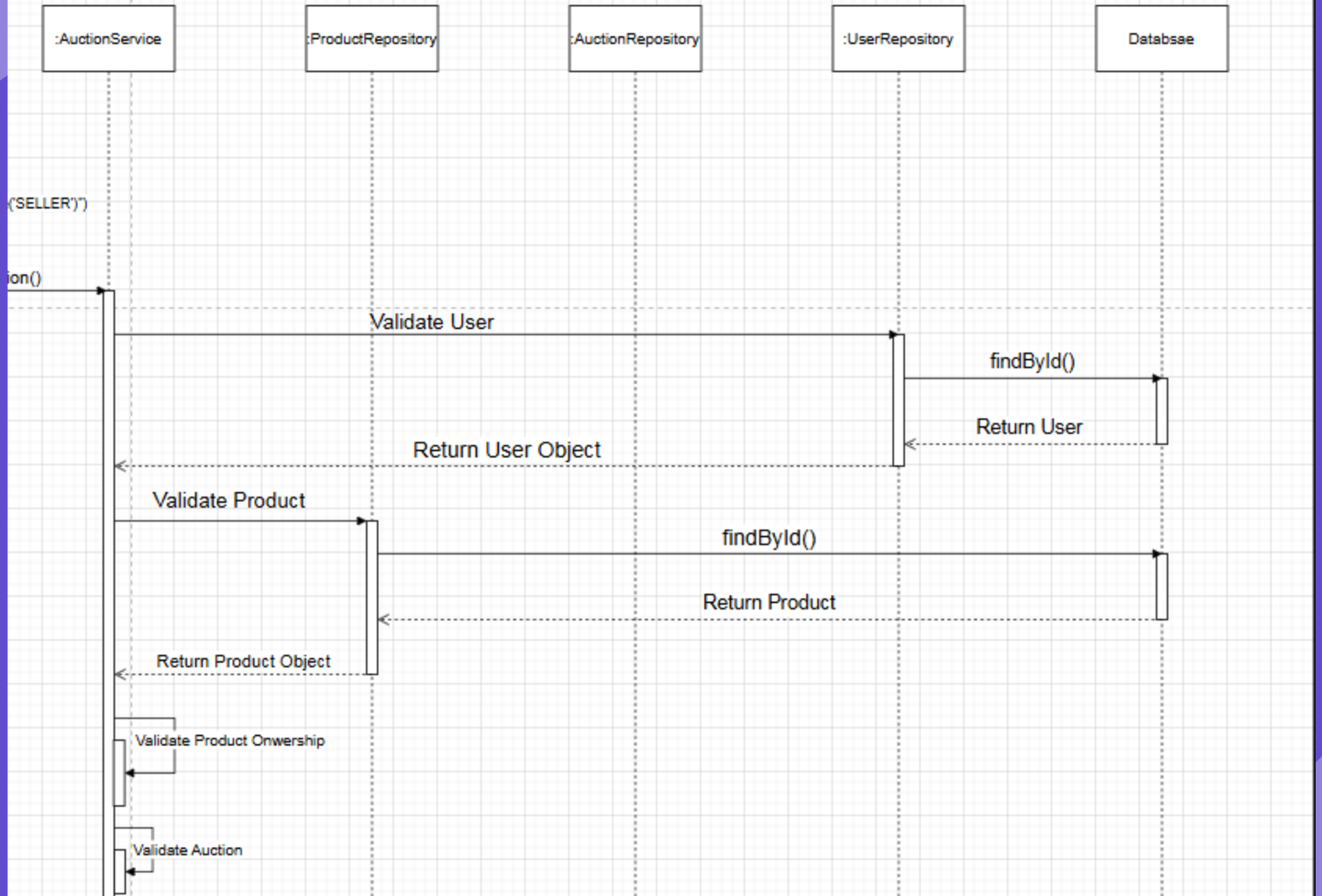


View + Create Auction

01 VALIDATION LOGIC

02 PESSIMISTIC LOCKING




:AuctionService
:ProductRepository
:AuctionRepository
:UserRepository
Databsae
('SELLER')")
ion()
Validate User
findById()
Return User
Return User Object
Validate Product
findById()
Return Product
Return Product Object
Validate Product Onwership
Validate Auction

```java
// Create an auction
@Transactional  1 usage  AnhNe *
public AuctionResponseDTO createAuction
(CreateAuctionRequestDTO dto, UserDetailsImpl sellerDetails)
        throws BadRequestException {...}
```

```java
public record CreateAuctionRequestDTO(...) {
}
```

```java
// Find the seller by userId
User seller = userRepository.findById(sellerDetails.getUserId())
        .orElseThrow(()
        -> new RuntimeException("Seller not found with ID: " + sellerDetails.getUserId()));

// find the product by productId
Product product = productRepository.findById(dto.productId())
        .orElseThrow(()
        -> new RuntimeException("Product not found with ID: " + dto.productId()));
```

```java
// create the auction
Auction auction = Auction.builder()
        .product(product)
        .startTime(dto.startTime())
        .endTime(dto.endTime())
        .startingPrice(dto.startPrice())
        .currentPrice(dto.startPrice())
        .build();
```

```java
// Save auction
Auction savedAuction = auctionRepository.save(auction);
```

```java
// Map to AuctionResponseDTO and return
return mapToAuctionResponseDTO(savedAuction);
```

```java
public record AuctionResponseDTO(...) {
}
```

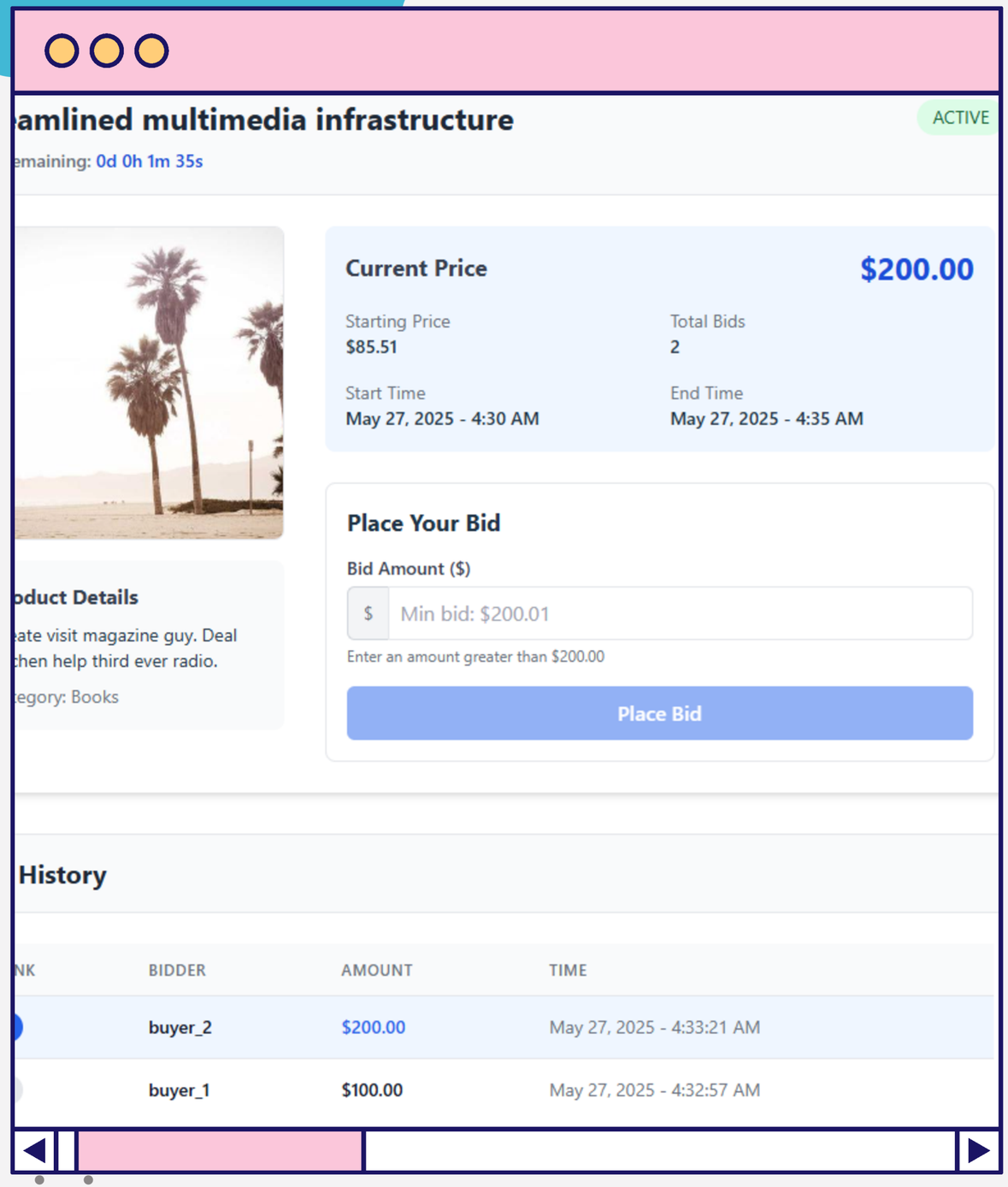
PESSIMISTIC LOCKING

```
@Lock(LockModeType.PESSIMISTIC_WRITE)   3 usages   new *
List<Auction> findByStatusAndEndTimeBefore
(Auction.AuctionStatus auctionStatus, LocalDateTime endTime
```

```
@Scheduled(fixedRate = 5000)   AnhNe +1 *
@Transactional
public void closeEndedAuctions() {
    LocalDateTime now = LocalDateTime.now();
    // Find auctions ready to be closed
    List<Auction> auctionsToClose = auctionRepository.
            findByStatusAndEndTimeBefore(Auction.AuctionStatus.ACTIVE, now);

    // Logic to set auction status and notify winners
    for (Auction auction : auctionsToClose) {...}
}
```

Bidding Process



Implement Websocket in FE

```
import { Client } from '@stomp/stompjs'; // Import STOMP Client
import SockJS from 'sockjs-client'; // Import SockJS
```

```
const client = new Client({
  webSocketFactory: () :SockJS => new SockJS(SOCKET_URL), // Use SockJS for transport
```

```
// Subscribe to the specific auction's bid topic
const subscription = client.subscribe(`/topic/auctions/${id}/bids`, (message :IMessage ) :void => {
```

Implement Websocket in BE

```
@Configuration  AnhNe
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override  no usages  AnhNe
    public void configureMessageBroker(MessageBrokerRegistry config) {...}

    @Bean  AnhNe
    public TaskScheduler heartbeatScheduler() {...}

    @Override  no usages  AnhNe
    public void registerStompEndpoints(StompEndpointRegistry registry) {...}

    @Bean  AnhNe
    public ServletServerContainerFactoryBean createWebSocketContainer() {...}
```

```java
@Override  no usages  AnhNe
public void configureMessageBroker(MessageBrokerRegistry config) {
    // Enable simple broker with heartbeat
    config.enableSimpleBroker( ...destinationPrefixes: "/topic")
            .setHeartbeatValue(new long[] {10000, 10000}) // Client and server heartbeat
            .setTaskScheduler(heartbeatScheduler()); // Task scheduler for heartbeats
```

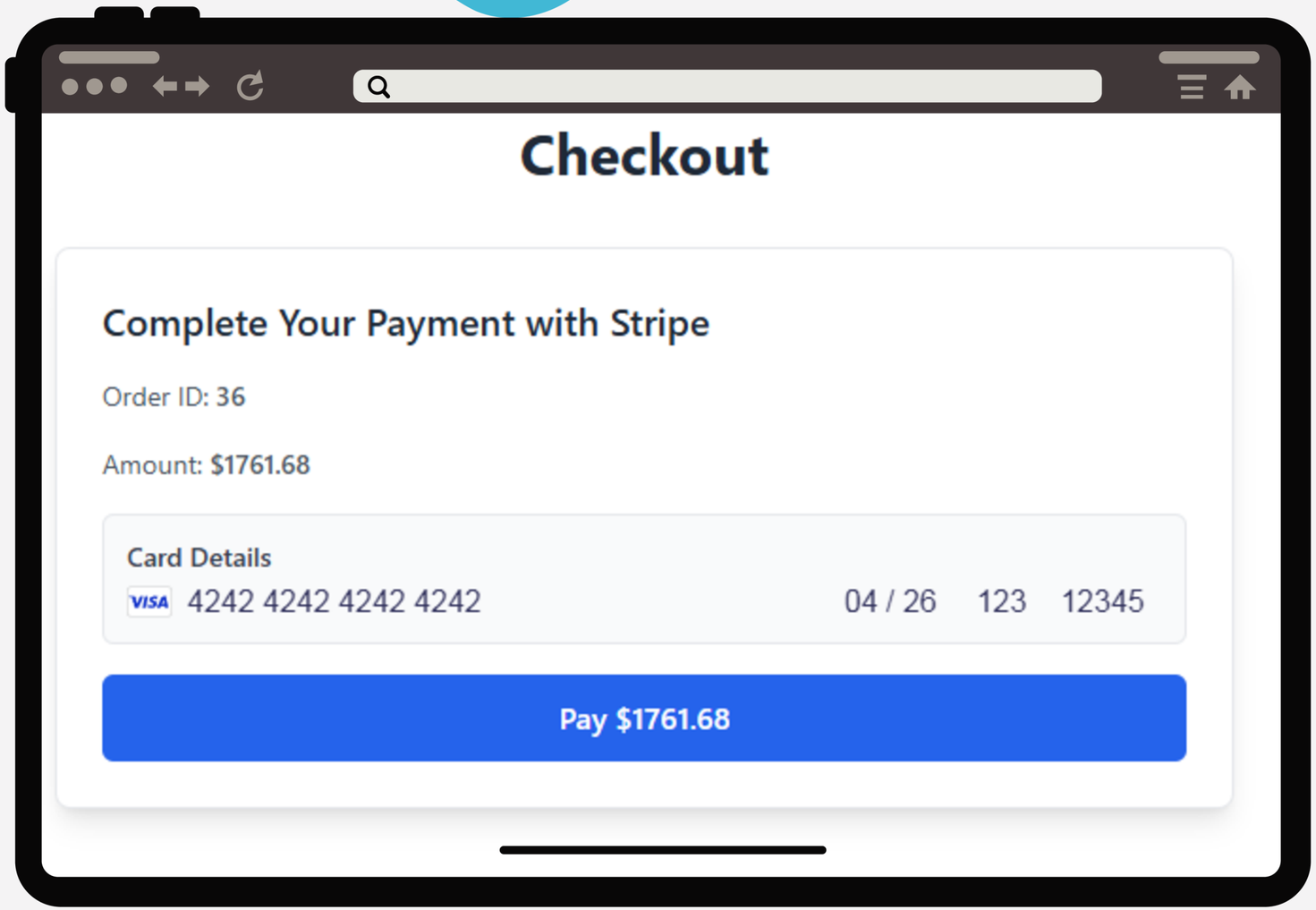
```java
@Override  no usages  AnhNe
public void registerStompEndpoints(StompEndpointRegistry registry) {
    // Register endpoint with SockJS support
    registry.addEndpoint( ...paths: "/ws")
            .setAllowedOriginPatterns("*")  // Allow all origins for development
            .withSockJS(); // Enable SockJS fallback options
```

```java
public record PlaceBidRequestDTO(
)
```

```java
public record BidResponseDTO(...) {
}
```

```java
// place a bid
@Transactional  4 usages  AnhNe *
public BidResponseDTO placeBid
(long auctionId, PlaceBidRequestDTO dto, UserDetailsImpl bidderDetails) {...}
```

```java
// 11. Broadcast bid via websocket
try {
    String destination = "/topic/auctions/" + auctionId + "/bids";
    log.info("Broadcasting bid to WebSocket destination: {}", destination);
    messagingTemplate.convertAndSend(destination, bidResponseDTO);
} catch (Exception e) {
    // Log WebSocket errors but don't fail the transaction
    log.error("Failed to broadcast bid via WebSocket for auction ID {}: {}"
            , auctionId, e.getMessage(), e);
    // Depending on requirements, we can to handle this differently (e.g., queueing)
}
```



Payment Process

Back

Next

05

Summary

Project Summary

E-commerce web application that supports both allowing buying items at a fixed prices and auction system

Challenging

Design database

Learn Websocket.

Future Improvement

Mobile responsiveness

Different payment method

Reference

ReactJS + VITE Framework : https://react.dev/ || https://vite.dev/

SpringBoot Framework :Tutorial from Youtube || https://spring.io/ || https://docs.spring.io/spring-boot/index.html || https://start.spring.io/ || https://docs.spring.io/spring-framework/reference/web/websocket.html

JSON Web Token (JWT) : https://jwt.io/

Stripe Payment Gateway API : https://stripe.com/

UI Reference and the main idea to make this project: https://www.ebay.com/

Testing APIs Tool : https://www.postman.com/

Database Hosting : https://hub.docker.com/_/mysql

Google Oauth2 : https://console.cloud.google.com/

Google Application Password (email service): https://myaccount.google.com/apppasswords

Fake product images source : https://images.unsplash.com/

Q&A

Thank You

For Your Attention!