Auction Platform System

Prepared by: **Leo Mark Castro** || **Full Stack Web Developer**

Introduction

The purpose of the application is to be Car Auction Platform with a minor moderation dashboard. The application will have to primary users:

1. User
    a. They can view ongoing bids, create bids according to the system rules and get the product being auctioned at if they won the bidding transaction
    b. The user can also create products that they would like others to auction for.
2. Admin
    a. The admin should be able to moderate the auction listings

Use Cases

1. **Registration**
    a. New users are required to register and log in before being able to participate in the auction. Required data: full name, email address, phone number.
2. **Login**
    a. Registered users have to log in before being able to use any feature
    b. For this test, only two types of user authorization are required, normal user, and admin.
3. **Logout**
4. **Listing**
    a. All registered users can see the list of auctions.
    b. Upon choosing a listing, the details of the auction will be displayed.
    c. All registered users can create a new auction, which consists of car brand, year, type, opening price, price increment, and expiry date of the auction.
    d. Users can bid on any open auction, except for auctions belonging to their own. The bidder cannot set just any bid price. The next bid price is always the last bid price + bid price increment.
    e. If there are 10 people at the same time trying to bid same item with the same price, there can only be one owner for that bid, the rest 9 must be rejected (i.e. "Sorry, somebody has bid quicker with this price. Please check the new price") and they can still bid with a new, higher price later on after the price update.
    f. Expired listings that have no bidder will remain open until closed by the owner. If there is a new bid after it expires, that bid will become the winner.
5. **Admin**
    a. Admin type users can close a listing and can permanently delete a listing.

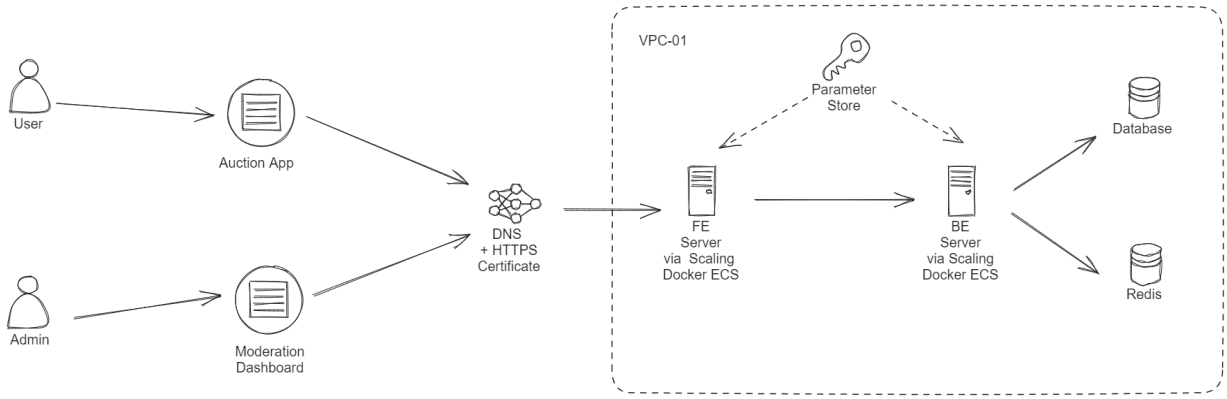
b. For this exercise, the admin-type user can only be set directly in the database row.

Considerations

- Security aspect for FE and BE
- Performance and scalability
- Bid reliability. I.e. many users bidding on the same item at the same time
- Good design on user roles and permissions, even if in this exercise only two types of users are required, please prepare for the future where there can be more permission types.

Design Specifications

Architecture Diagram



The Diagram above should show the general architectural approach on the setup and configuration of the Web Servers for the application should it include the front-end and back-end servers hosted on AWS.

To further provide details: It shows the 2 primary users of the application namely the ‘User’ and the ‘Admin’ accessing their own applications that are accessed via a URL that is provided by a DNS with HTTP Certificate (via Route53 and AWS Certificate Manager). Ideally, the two dashboards were served by a single FE Server (NextJS) that fetches its data on a backend server (NestJS). Both of them are dockerized applications running on Elastic Container Service with auto scaling feature enabled via the Elastic Load Balancer and ECS Auto Scaler. Their configurations are loaded via the Parameter Store dashboard. Notably, it contains the URL link for the AWS Relational Database instance and the S3 Instance. The Servers should be inside a

non-public Virtual Private Cloud network so that non-aws instances cannot access the servers unless explicitly defined in the Security Group.

Tech Stack

For the tech stack, the developer would recommend the usage of the following technologies

- NextJS - Frontend Framework
- NestJS - Backend
- PostgreSQL via RDS - Database
- Redis / Redis Queue - Queue
- ECS + ELB - For Auto Scalable and Fast Deployable Server Docker Instances
- Route 53 + ACM - URL and HTTPS Certificate
- VPC Security Group - Prevent External Access to Backend (Configurable) and Database
- Parameter Store - Global Storage of Sensitive Information

The general approach is to use a dedicated server and codebase for NextJS so that it is detached from backend architecture and open for future revisions. For the backend server, the current approach uses a monolith server. It is easy, fast and efficient to implement. It is also scalable via multi-instance deployment via Elastic Load Balancing and ECS Auto scaling.

PostgreSQL were also chosen to enforce unified data schema.

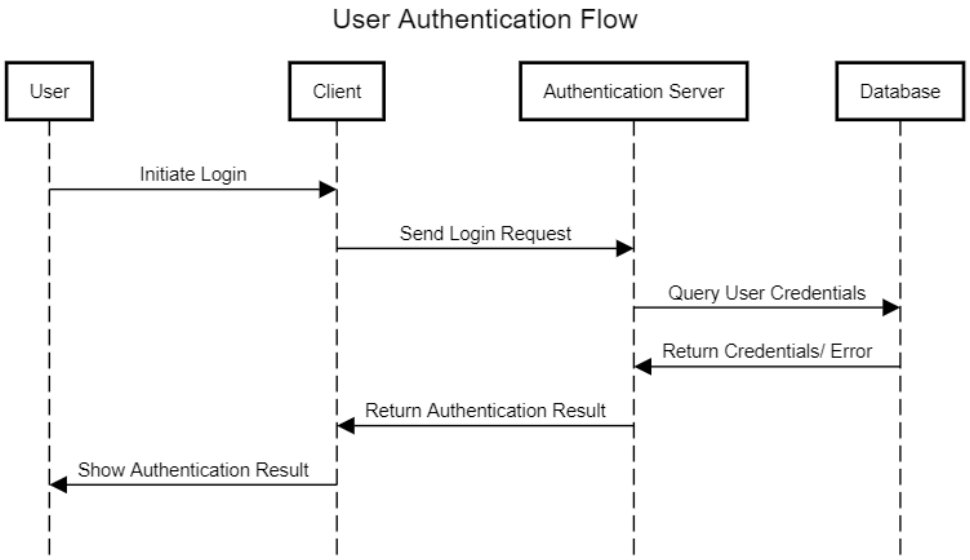
Sequence Diagrams

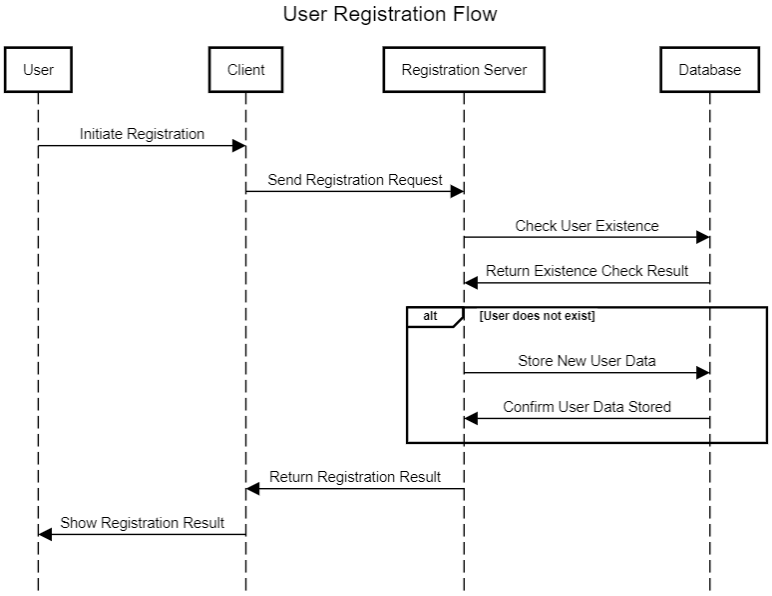
The following flows focuses on the high-level End User-focused sequence diagram with emphasis on User Journey as they visit the website.

Authentication Flows

For authentication, it involves the User Login flow and User Registration. Provided that email and password would be used as credentials. On register, the user should also provide intermediate profile information such as full name and phone number.

The authentication method would use HTTP-only cookies to facilitate authentication tagging and avoid JS snipping.

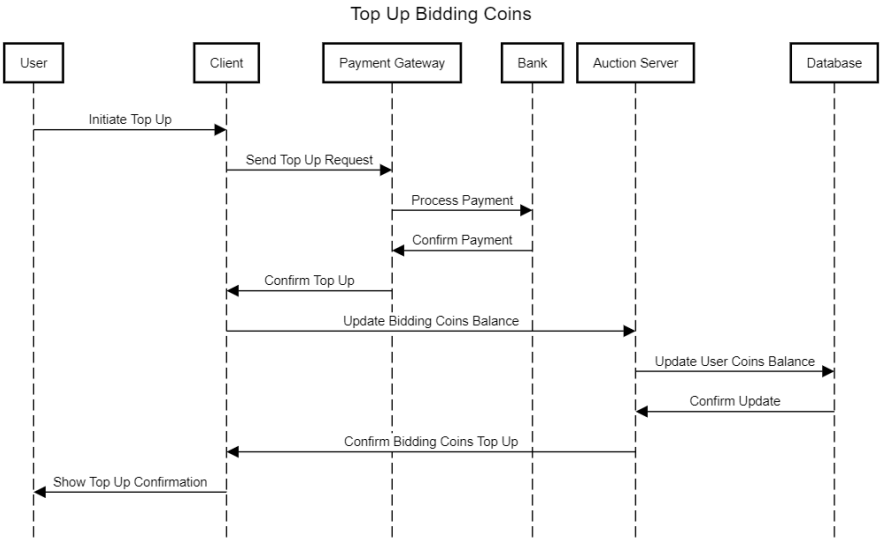




Bidding Balance Top Up

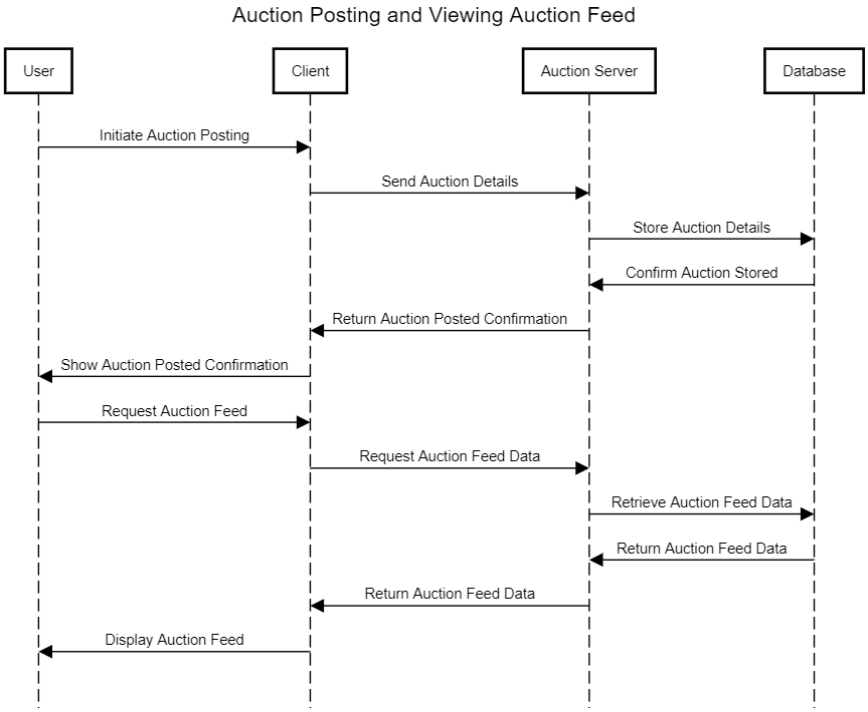
To speed up the bidding speed for auctions, the user should top-up on their account first to get ‘bidding coins’. These bidding coins would be used to put bids on products and be refunded back to the ‘bidding wallet’ if someone else won the bidding raise. The user should be able to withdraw these coins later if unused on any biddings.

Top-ups could also be implicitly created if the user pays for the auction raise directly.



Auction Listing

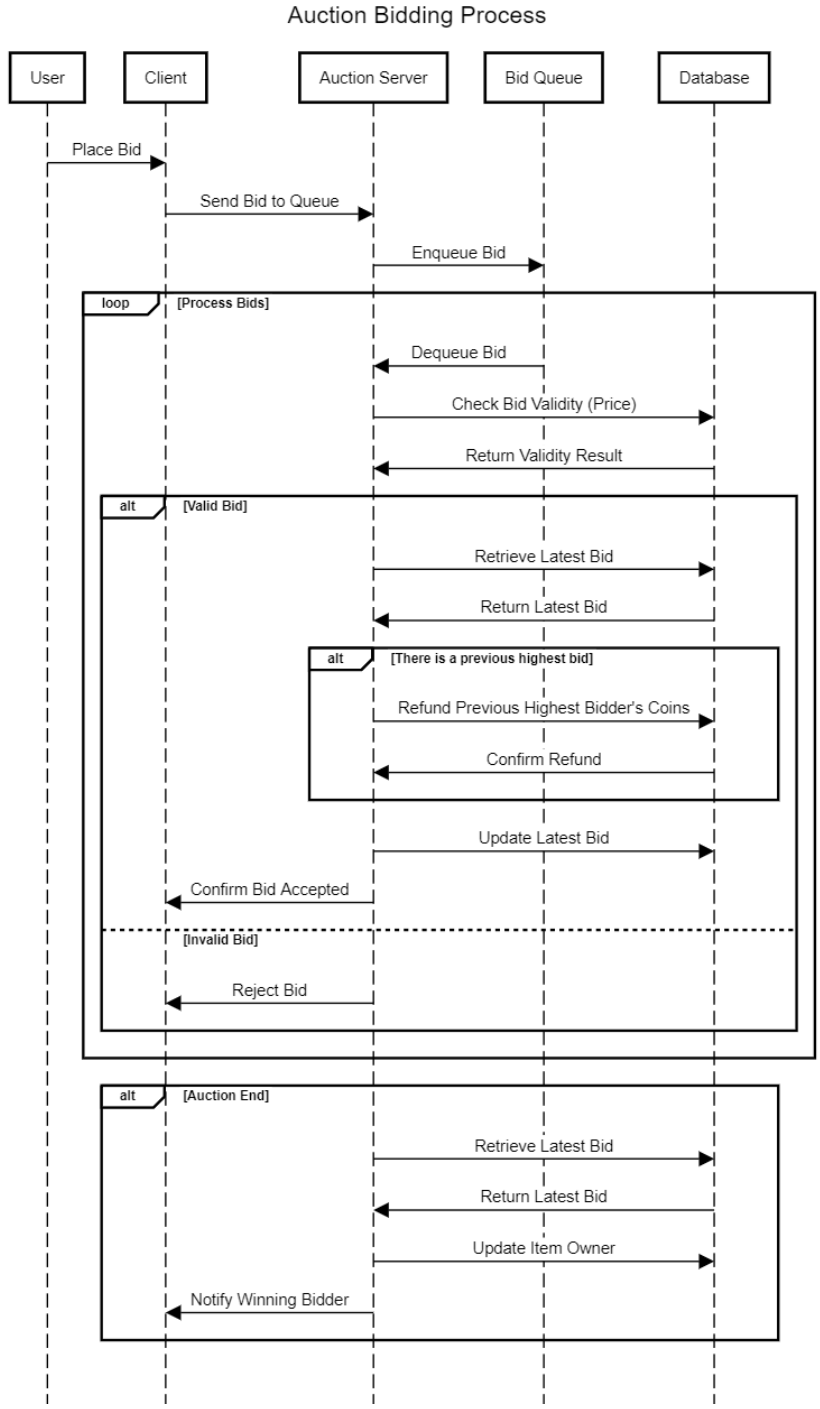
Sequence Flow detailing the Auction Creation and Auction Fetching Mechanism.



Auction Bidding

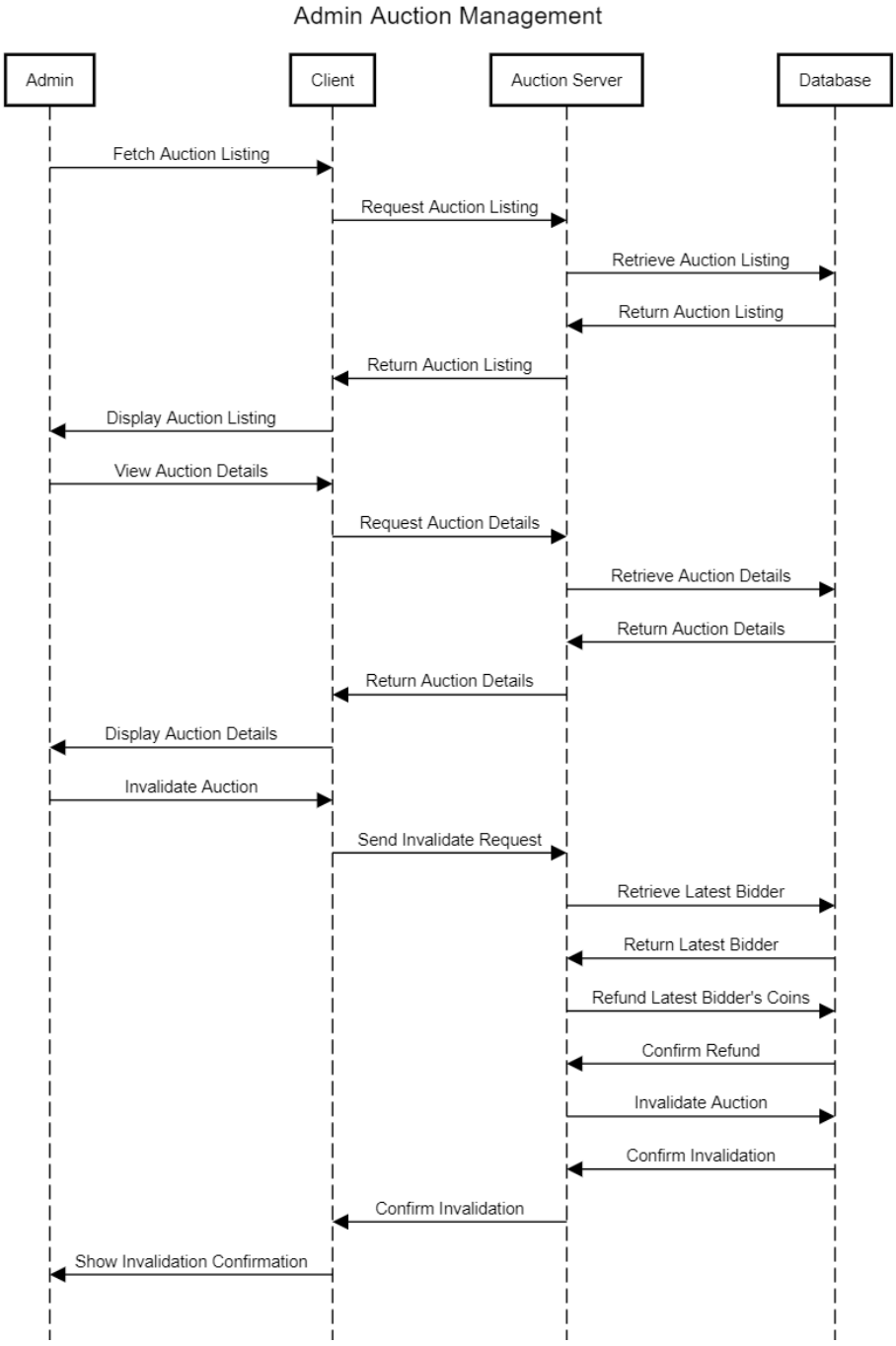
In this process, it was detailed that a Bid Queue was implemented to process multiple incoming bids. Each bid will be processed sequentially.

Once the auction finishes, the auction bidding closes for that specific listing and the item would officially be acquired by the latest bid winner.



Listing Moderation

In this sequence, the admin should be able to view available auction listing, view information and send an invalidate auction to trigger an auction invalidation and automatic refund for the latest winner.



Endpoint List

Authentication and Registration:

- `POST /api/auth/login` - User login.
- `POST /api/auth/register` - User registration.

Auction Posting and Viewing:

- `POST /api/auctions` - Create a new auction.
- `GET /api/auctions` - Fetch all auctions.
- `GET /api/auctions/:auctionId` - Fetch details of a specific auction.

Auction Bidding:

- `POST /api/bids` - Place a bid on an auction.
- `GET /api/auctions/:auctionId/bids` - Fetch all bids for a specific auction.

Admin Auction Management:

- `GET /api/admin/auctions` - Fetch all auctions for admin view.
- `GET /api/admin/auctions/:auctionId` - Fetch details of a specific auction for admin view.
- `POST /api/admin/auctions/:auctionId/invalidate` - Invalidate an auction.

User Top-Up for Bidding Coins:

- `POST /api/topup` - Process top-up payment and update user coins.
- `PUT /api/stripe/webhook` - Webhook endpoint to receive payment success triggers and update user's balance.

Database Diagram