Backend Intern Task

Work from Home

Create a comprehensive RESTful API for a real-time bidding platform using Node.js, Express, Socket.io, and a SQL database (PostgreSQL or MySQL). The API should support advanced CRUD operations, user authentication, role-based access control, real-time bidding, and notifications.

Requirements:

Environment Setup:

Use Node.js and Express to create the API.

Use Socket.io for real-time communication.

Use PostgreSQL or MySQL for the database.

Use environment variables for database and other configuration settings.

Database Schema:

Create a users table with the following fields:

id (Primary Key)

username (String, unique, not null)

password (String, not null)

email (String, unique, not null)

role (String, default to 'user') // roles: 'user', 'admin'

created_at (Timestamp, default to current time)

Create an items table for auction items with the following fields:

id (Primary Key)

name (String, not null)

description (Text, not null)

starting_price (Decimal, not null)

current_price (Decimal, default to starting_price)

image_url (String, nullable) // for storing image paths

end_time (Timestamp, not null) // auction end time

created_at (Timestamp, default to current time)

Create a bids table for storing bids with the following fields:

id (Primary Key)

item_id (Foreign Key referencing items.id)

user_id (Foreign Key referencing users.id)

bid_amount (Decimal, not null)

created_at (Timestamp, default to current time)

Create a notifications table to store notifications for users:

id (Primary Key)

user_id (Foreign Key referencing users.id)

message (String, not null)

is_read (Boolean, default to false)

created_at (Timestamp, default to current time)

API Endpoints:

Users:

POST /users/register - Register a new user.

POST /users/login - Authenticate a user and return a token.

GET /users/profile - Get the profile of the logged-in user.

Items:

GET /items - Retrieve all auction items (with pagination).

GET /items/:id - Retrieve a single auction item by ID.

POST /items - Create a new auction item. (Authenticated users, image upload)

PUT /items/:id - Update an auction item by ID. (Authenticated users, only item owners or admins)

DELETE /items/:id - Delete an auction item by ID. (Authenticated users, only item owners or admins)

Bids:

GET /items/:itemId/bids - Retrieve all bids for a specific item.

POST /items/:itemId/bids - Place a new bid on a specific item. (Authenticated users)

Notifications:

GET /notifications - Retrieve notifications for the logged-in user.

POST /notifications/mark-read - Mark notifications as read.

WebSocket Events:

Bidding:

connection - Establish a new WebSocket connection.

bid - Place a new bid on an item.

update - Notify all connected clients about a new bid on an item.

Notifications:

notify - Send notifications to users in real-time.

Authentication and Authorization:

Use JWT (JSON Web Tokens) for authentication.

Implement role-based access control to restrict access to certain endpoints based on user roles.

Protect the POST, PUT, and DELETE endpoints appropriately.

Validation and Error Handling:

Validate incoming data for required fields.

Handle and return appropriate HTTP status codes and messages for errors (e.g., 400 for bad requests, 401 for unauthorized, 403 for forbidden, 404 for not found).

Image Upload:

Implement image upload functionality for auction items using a library like multer.

Store image URLs in the database.

Search and Filtering:

Implement search functionality for auction items.

Allow filtering items by status (e.g., active, ended).

Pagination:

Implement pagination for the GET /items endpoint.

Notifications:

Implement a notification system to notify users about bids on their items and when they are outbid.

Project Structure:

Organize your project with a clear structure, separating concerns (e.g., routes, controllers, models, and services).

Testing:

Add unit and integration tests for the API using a testing framework like Mocha, Chai, or Jest.

Documentation:

Provide a README file with instructions on how to set up and run the project.

Document the API endpoints and their expected inputs and outputs.

Bonus:

Implement a rate limiting middleware to prevent abuse of the API.

Use a linter like ESLint to ensure code quality.

Implement logging for API requests and errors.

Add a feature for users to reset their passwords.

Use Docker for containerization.

Deliverables:

Source code hosted on a Git repository (e.g., GitHub).

README file with setup and usage instructions.

SQL scripts or migrations for setting up the database schema.

Tests covering the main functionality.