Estate Vista - A Real Estate MERN Stack project

The Estate Vista project is a dynamic and feature-packed platform designed to redefine how users interact with property listings. Leveraging the MERN (MongoDB, Express.js, React, Node.js) stack, this project seamlessly integrates modern technologies to create a robust and user-friendly experience. The Real Estate website project aims to set new standards in the real estate industry by offering a modern, intuitive, and secure platform for users to explore, inquire, and engage with property listings. Whether you are a potential buyer, seller, or administrator, this project caters to diverse users, providing a seamless and enjoyable real estate experience.

Design Document

Technology Stack

Frontend

- **React.js**: The front end of the real estate website will be built using React.js, leveraging its component-based architecture for creating dynamic user interfaces.
- **React Router**: React Router will handle client-side routing, enabling navigation between different pages of the website without full page reloads.
- **Redux:** Redux will be used for state management, providing a predictable and centralized state container for the application. This will facilitate data sharing and communication between React components.
- **Tailwind CSS**: Tailwind CSS will be used for styling the website. Its utility-first approach allows for rapid development and customization of the user interface.
- **Axios**: Axios will handle HTTP requests from the frontend to the backend API, facilitating data fetching and interaction with the server.

Backend

- **Node.js**: The backend of the website will be powered by Node.js, allowing for the development of scalable and efficient server-side applications using JavaScript.
- **Express.js**: Express.js will handle routing, middleware, and HTTP requests on the server side, providing a robust foundation for building RESTful APIs.
- **MongoDB**: MongoDB will store data for the real estate website. Its flexibility and scalability make it suitable for storing various types of data, including user profiles, property listings, and related information.
- **Mongoose**: Mongoose will simplify interactions with the MongoDB database, providing schema validation and modeling capabilities.

Authentication and Authorization

- **Google OAuth**: User authentication will be implemented using Google OAuth for secure login and registration.
- **JWT (JSON Web Tokens)**: JSON Web Tokens will be used for authentication and authorization purposes, enabling access to protected routes and resources.

Image Storage

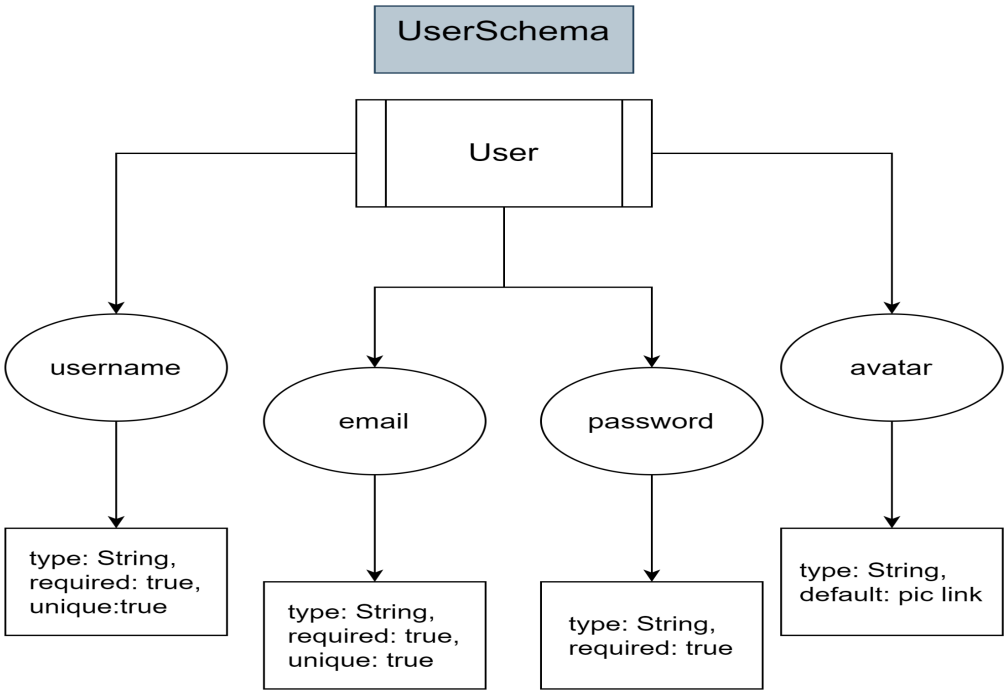
- **Cloudinary:** Cloudinary will be used for storing and serving property images. Its cloud-based infrastructure provides scalability and performance benefits for handling image uploads and storage.

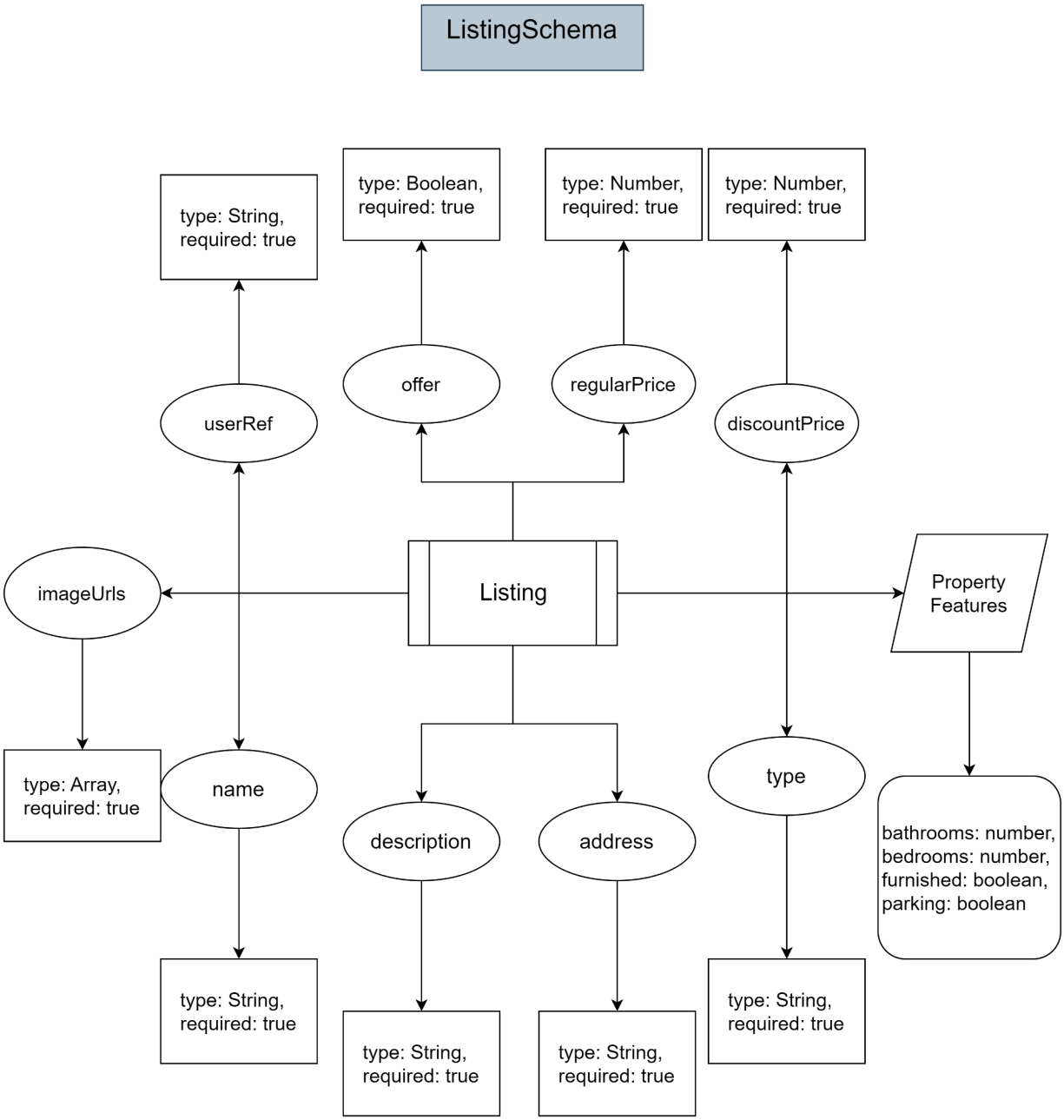
Deployment

- **Vercel**: The real estate website will be deployed on Vercel, leveraging its seamless integration with the MERN stack, continuous deployment capabilities, and serverless architecture.
- **GitHub**: The website's source code will be hosted on GitHub for version control and collaboration, enabling automated deployments to Vercel.

Database Schemas

Using MongoDB for storing data on the website.





API Design

1. Auth
    - post/signup → to signup a new account
    - post/signin → to signin a existing account
    - post/google → to signin/signup with Google
    - get/signout → to signout from the account

2. Listing
    - post/create → to create a new listing of property
    - delete/delete/:id → to delete an existing property
    - post/update/:id → to update an existing property
    - get/get/:id → to get 1 property from listing according to ID
    - get/get → to get all properties from listing

3. User
    - post/update/:id → to update user info
    - delete/delete/:id → to delete user
    - get/listing/:id → to get user's listing of property
    - get/:id → to get user

Filtering Options

1. **Search**

- Users can search for properties using a search bar, allowing them to input keywords, locations, or other relevant search terms.
- The search functionality will filter properties based on the entered query and display matching results in real time.

2. **Rent or Sell**

- Users can filter properties based on whether they are available for rent, for sale, or both.
- Checkboxes or dropdown menus will allow users to select their preferred option(s) for filtering.

3. **Price Sorting**

- Users can sort properties based on price, allowing them to view listings from high to low or low to high.
- Radio buttons or dropdown menus will enable users to choose their preferred sorting option.

Implementation Details

- The frontend will send filter criteria to the backend API, which will query the MongoDB database accordingly.
- Express.js routes will handle the filtering logic on the server side, fetching filtered results from the database.
- Axios will facilitate communication between the front end and backend, sending filter parameters and receiving filtered data.

- React components will dynamically render filtered property listings based on the received data.
- Redux will manage the application state, including the current filter settings and retrieved property data.

Example Workflow

- The user selects filtering options (e.g., rent, price high to low).
- Frontend sends filter criteria to the backend API.
- Express.js route processes the filter criteria and queries the MongoDB database for matching properties.
- Filtered property data is sent back to the frontend via Axios.
- React components dynamically render the filtered property listings based on the received data.
- The user views and interacts with the filtered property listings on the frontend, with Redux managing the application state.