TextZ - A Real Time Chating MERN Stack project

the Real-time Chat Application is a platform designed to facilitate instant communication between users. Leveraging technologies such as Socket.IO for real-time messaging, Node.js for the backend, MongoDB for data storage, and JWT along with Google Auth for user authentication, this application offers a seamless and secure chatting experience.

Design Document

Technology Stack

Frontend

- **React.js**: The front end of the chat app 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.

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

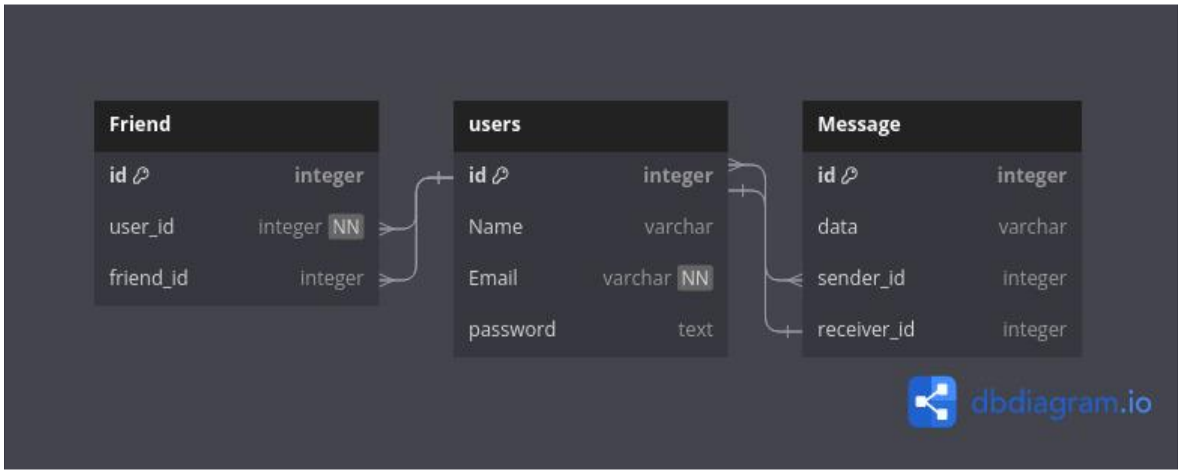
- **Cloudinary:** Cloudinary will be used for storing and serving property images. Its cloud-based infrastructure provides scalability and performance benefits for handling image uploads and storage.

Deployment

- **Netlify**: The real estate website will be deployed on Netlify, 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 Netlify.

Database Schemas

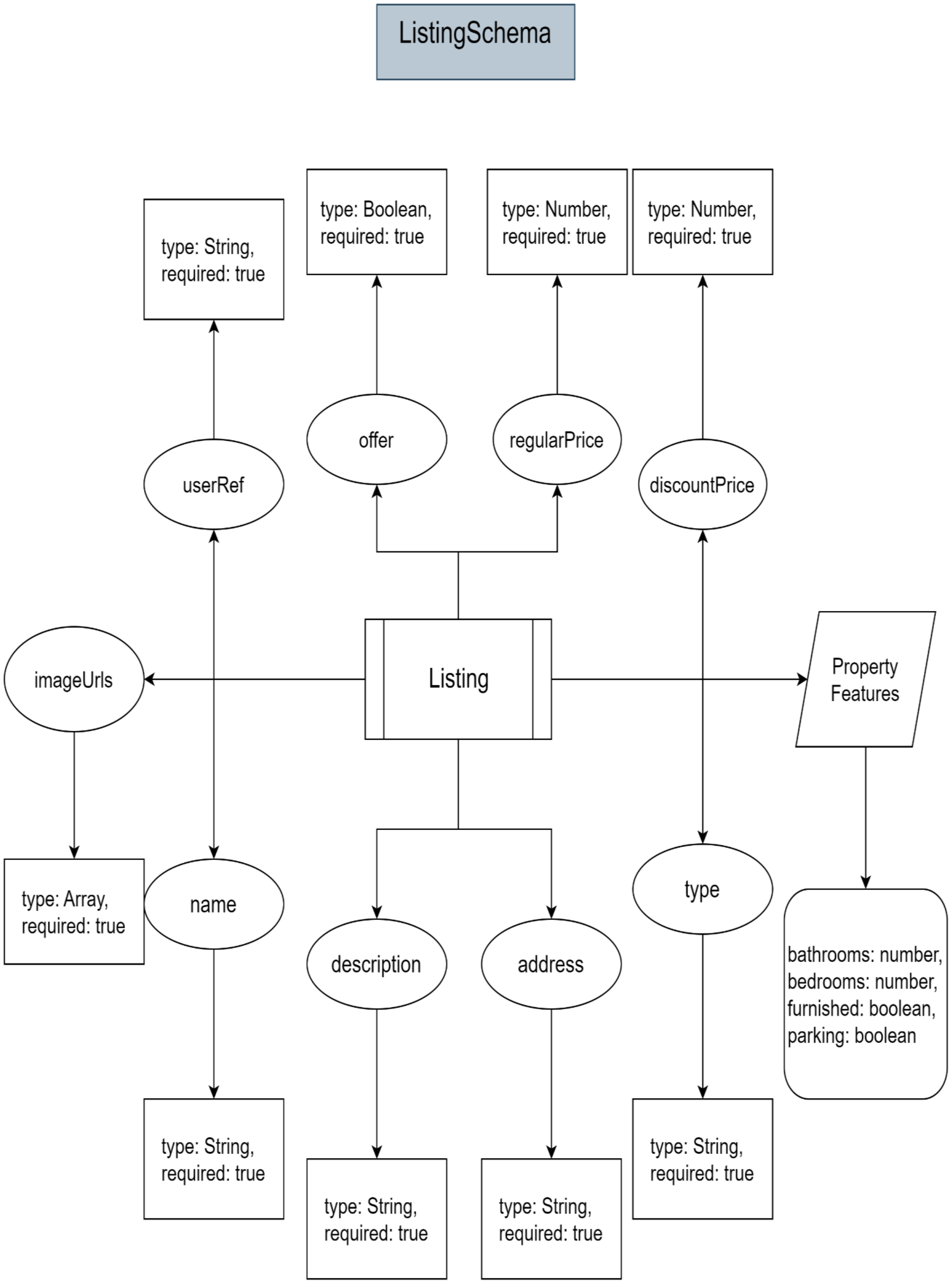
Using MongoDB for storing data on the website.



Friend	
id	integer
user_id	integer NN
friend_id	integer

users	
id	integer
Name	varchar
Email	varchar NN
password	text

Message	
id	integer
data	varchar
sender_id	integer
receiver_id	integer


ListingSchema
Listing
userRef
type: String, required: true
offer
type: Boolean, required: true
regularPrice
type: Number, required: true
discountPrice
type: Number, required: true
imageUrls
type: Array, required: true
Property Features
bathrooms: number, bedrooms: number, furnished: boolean, parking: boolean
name
type: String, required: true
description
type: String, required: true
address
type: String, required: true
type
type: String, required: true

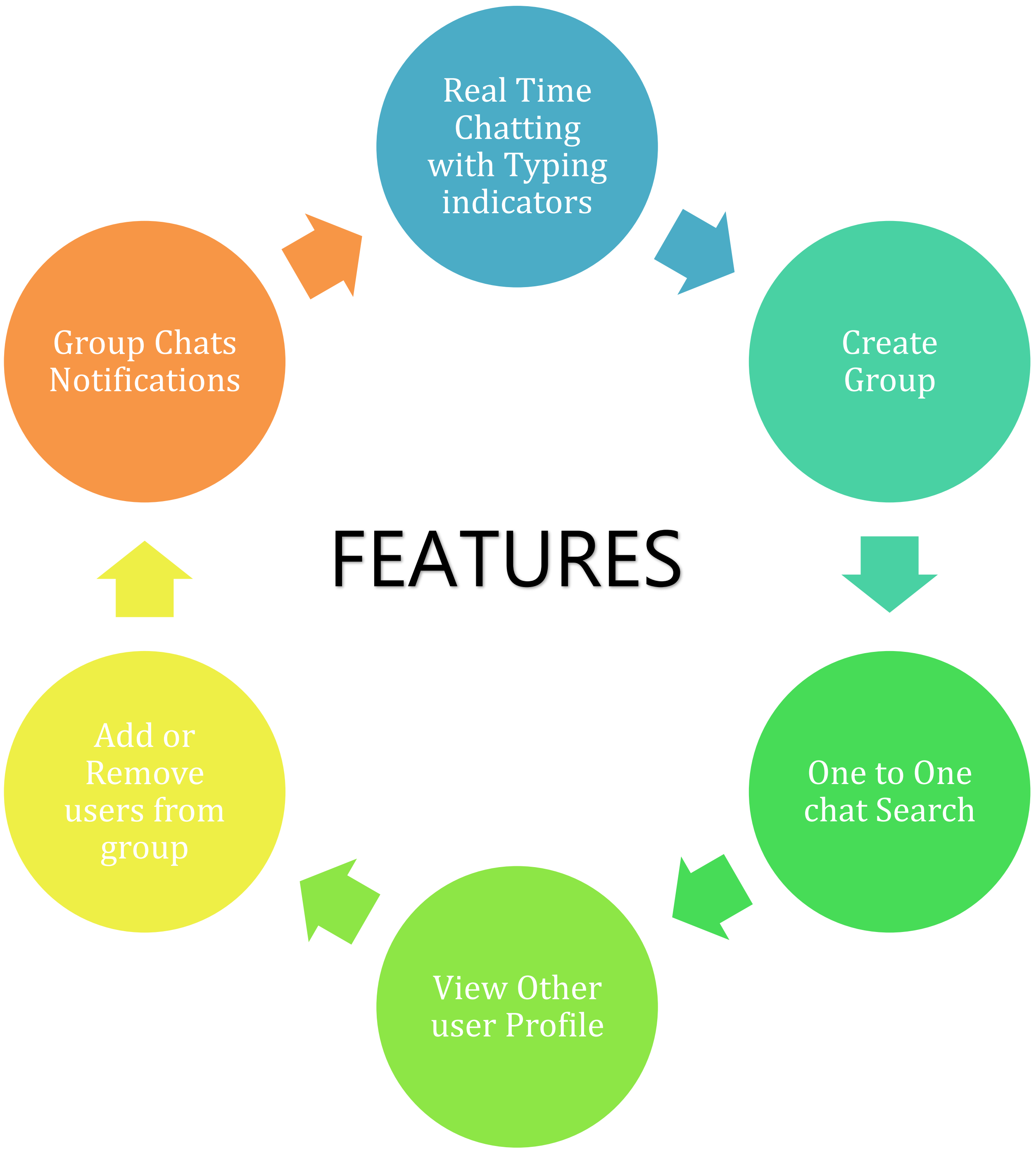
API Design

1Auth

- `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 accoun

1. Listing
    - `post/create` → to create a new listing of chat group
    - `delete/delete/:id` → to delete an existing chat group
    - `post/update/:id` → to update an existing chat group
    - `get/get/:id` → to get chat from listing according to ID
    - `get/get` → to get all chats from listing

2. User
    - `post/update/:id` → to update user info
    - `delete/delete/:id` → to delete user
    - `get/listing/:id` → to get the user's listing of chats
    - `get/:id` → to get user


Real Time Chatting with Typing indicators
Create Group
FEATURES
One to One chat Search
View Other user Profile
Add or Remove users from group
Group Chats Notifications