Software Design Documentation

Companion-Connect

Introduction

Companion Connect is a pet adoption website designed to bridge the gap between potential pet adopters and shelters or individual pet owners seeking to rehome their pets. The platform offers a seamless user experience, enabling adopters to browse available pets, learn about their characteristics, and contact pet owners or shelters. Additionally, it empowers shelters and individuals to easily list pets for adoption and manage inquiries. By focusing on user convenience and promoting responsible pet adoption, Companion Connect aims to reduce the number of pets in shelters and create lasting connections between animals and their new families.

System Overview

Companion Connect is a web-based application that provides the following functionalities:

1. User Registration and Profiles:

- Adopters can create profiles to save searches, bookmark pets, and contact owners.
- Shelters or individual pet owners can create profiles to list pets for adoption.

2. Pet Listings and Search:

- Users can browse a wide range of pets categorized by species, breed, age, size, and other attributes.
- Advanced search filters enable users to find pets that match their preferences.

3. Communication and Inquiry Management:

- Adopters can directly send inquiries to pet owners or shelters through an integrated messaging system.

4. Admin Features:

- Admins can monitor listings, ensure adherence to adoption guidelines, and manage platform operations.

Backend Design

Technologies Used:

- **Node.js**: JavaScript runtime environment.
- **Express.js**: Web framework for building APIs.
- **MongoDB**: NoSQL database for storage.
- **Mongoose**: ODM (Object Data Modelling) library for MongoDB.
- **JWT**: JSON Web Tokens for authentication.
- **bcrypt**: Library for encrypting passwords.

APIs and Database Schema

API Design:

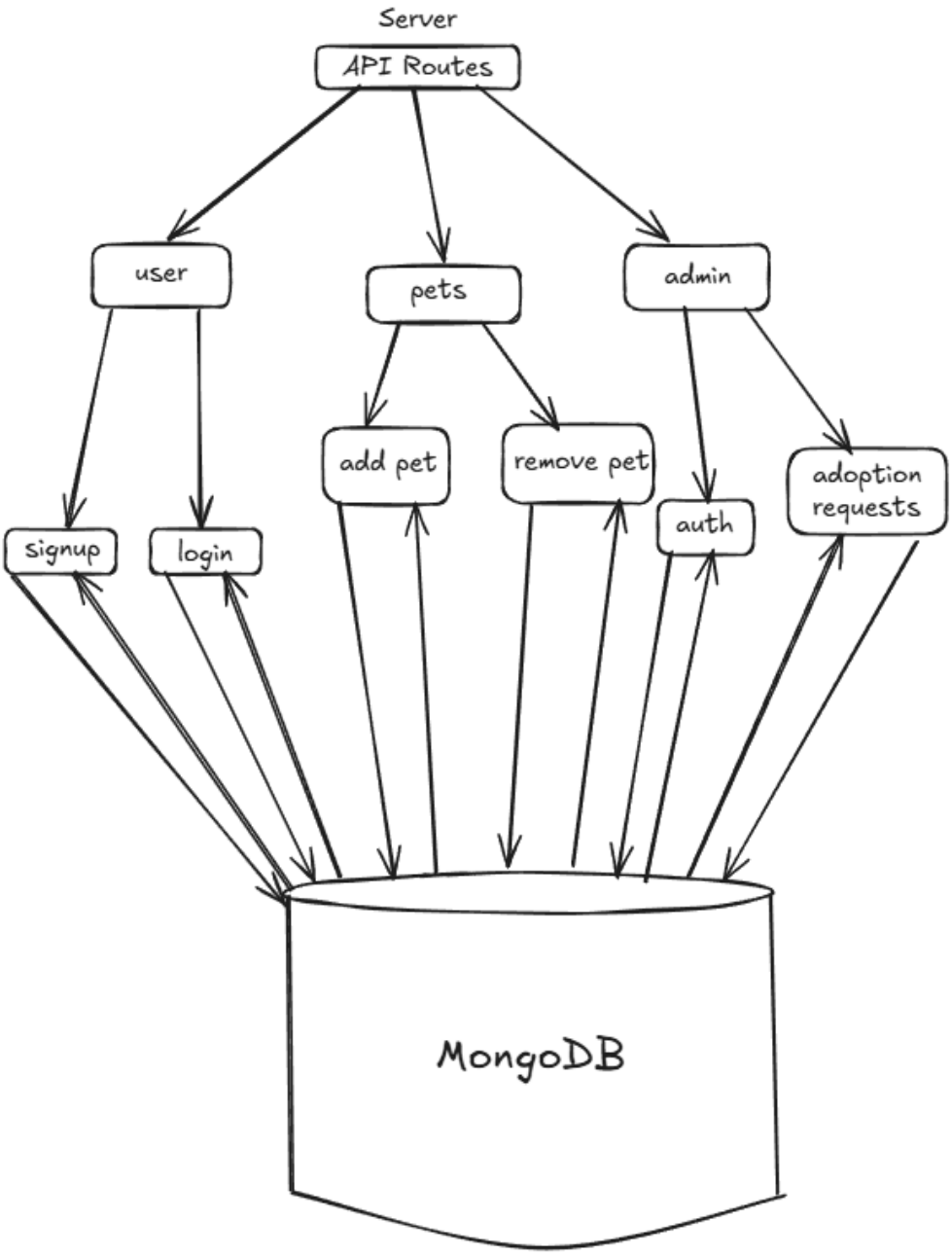
- **Authentication**:
  - `POST /login`: User login and JWT token issuance.
- **Users**:
  - `POST /user`: User registration.
  - `GET /user/:email`: Retrieve user information.
- **Pets**:
  - `POST /pet/upload-pet`: Upload the pet details.
- **Admin**:
  - `GET /admin/auth/:email`: Verifies admin status.
  - `GET /admin/pets`: Get the list of unapproved pets.
  - `POST /admin/pets/approve/:id`: Approve a pet for adoption availability.
  - `POST /admin/pets/reject/:id`: Reject and remove a pet from the database.

Database Schema:

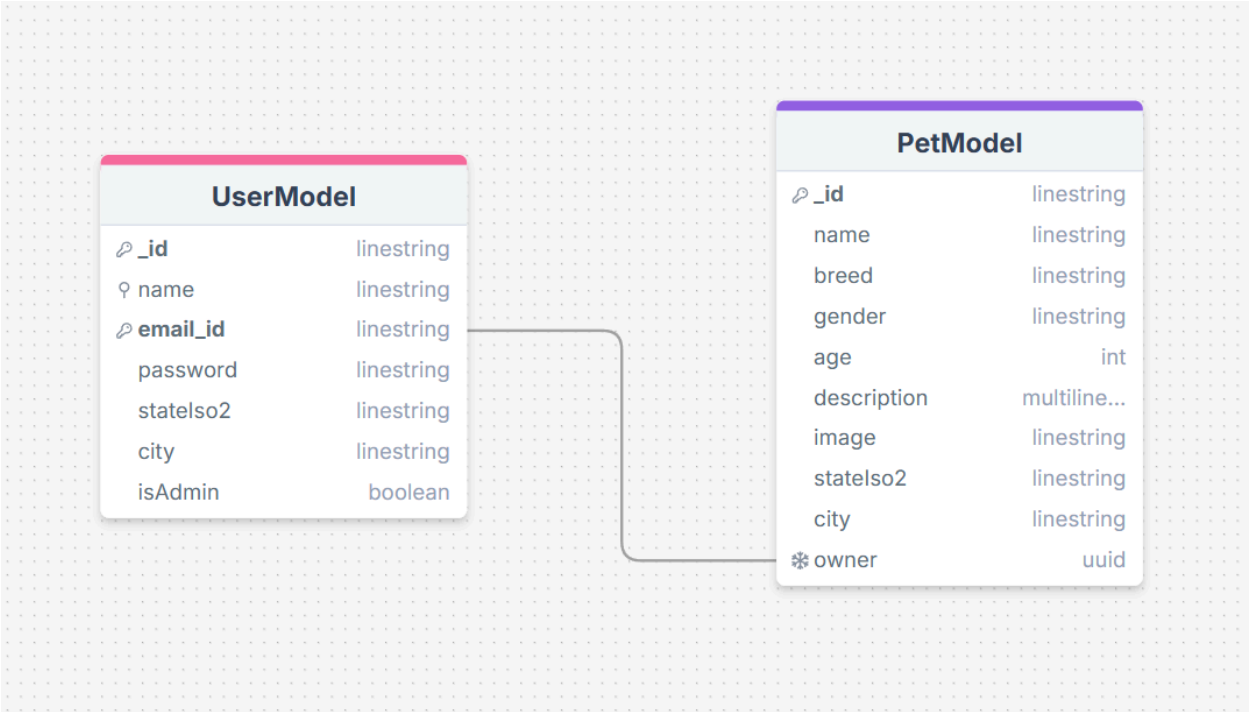
- **User Model**:
  - `Name (String)`: Full name of user.
  - `Email ID (String)`: Should be unique.
  - `Password (String)`: Stored after encryption.
  - `StateIso2 (String)`: ISO code of the state.
  - `City (String)`: City of the user.
  - `isAdmin (Boolean)`: Admin status.
- **Pet Model**:
  - `Name (String)`: Name of pet.
  - `Breed (String)`: Breed of pet.

- `Gender (String)`: Gender of pet.
- `Age (Number)`: Age of pet.
- `Description (String)`: Necessary information about the pet.
- `Image (String)`: Provided by Multer.
- `StateIso2 (String)`: ISO code of the state.
- `City (String)`: Current city of the pet.
- `Approved (Boolean)`: Approval status.
- `Owner (Schema.ObjectId)`: User ID of the pet owner.

API Routes Diagram



Database Diagram:



Frontend Design

Technologies Used:

- **ReactJS**: JavaScript library for building UI.
- **React-Router-DOM**: For managing frontend routing.
- **Tailwind CSS**: CSS framework for styling.
- **Vite**: Build tool for faster development.
- **ESLint**: Linting utility used to maintain code quality.

Client-Side Routes:

Using React-Router-DOM Library:

- `/`: Landing Page
- `/signup`: Signup/Registration Page
- `/login`: Login Page
- `/home`: Home Page
- `/adoption`: Give for Adoption Page
- `/admin`: Admin Page (only accessible by admins)

State Management:

- **Local State**: Managed using `useState` and `useEffect` hooks provided by ReactJS.
- **Authentication State**: Managed using `LocalStorage`.
- **Data Fetching and API Calls**: Utilizes the `FetchAPI`.

Project Structure:

- **main.jsx**: Entry point of the React application.
- **App.jsx**: Main application component
- **components/**: Reusable UI components.
- **router.jsx**: Defines client-side routing
- **index.css**: Global CSS and Tailwind directives.

Security Considerations

- **Authentication:** Implemented using JSON Web Tokens (JWT).
- **Password-Security:** Encrypted using **bcrypt.**
- **CORS:** Configured to allow requests only from trusted origins.

Deployment Plan

- **Backend Environment Variables:**
  - PORT
  - MONGOOSE_URI
  - DB_NAME
  - JWT_SECRET_KEY

- **Frontend Environment Variables:**
  - VITE_API_URL

Deployment Steps

- **Backend Deployment:**
  - Host on platforms like AWS, Heroku, Render
  - Use PM2 for process management

- **Frontend Deployment:**
    - Host on platforms like Vercel, Netflify, Github Pages etc.

- **Database:**
    - Use managed MongoDB services like Atlas
    - Enable IP whitelisting and backups

- **Domain and SSL:**
    - Configure custom domain and SSL certificates.

- **CI / CD Pipelines:**
    - Automate deployments using GitHub Actions or Jenkins.

Possible Future Enhancements

- **Technical Enhancements:**
    - Switch to TypeScript for better type safety.
    - Integrate Web-Sockets for Real-Time Updates (eg. available pets data And latest adoption requests).

- **Feature Enhancements:**
    - Enable direct messaging between users for inquiry.
    - Integrate a chatbot for 24x7 user support.
    - Add pet preferences option.