MERN Application Development Progress

Michaela Kemp 231001

Open Window, School of Fundamentals

DV200

Lecturer(s): Tsungai Katsuro

02 November 2024

Functional Requirements Checklist

User Authentication

- ☑ **User Registration and Login**: JWT-based authentication implemented with bcrypt for password hashing. This provides secure login and token-based access control across protected routes.

CRUD Operations for Requests

- ☑ **Create Requests**: Users can create travel requests with details like start location, end location, meeting time, and type.
- ☑ **Update Requests**: Users can update their requests, and if a canceled request is updated, it reopens automatically.
- ☑ **Reopen Requests**: Users can reopen previously closed requests.
- ☑ **Delete Requests**: Users can delete requests along with associated comments and likes.
- ☑ **Fetch User Requests**: Users can view a list of their own requests, including status (open, closed, or canceled).

API Integration (Google Maps API)

- ☐ **Google API Integration**: Users can choose start and end locations for requests and see the distance from their current location to the request's starting point.

Frontend and Backend Functionality

- ☑ **Backend**: Backend functionality includes all core API routes and integrates Express for routing, MySQL for data storage, JWT for authentication, and Multer for image uploads.
- ☑ **Frontend**: The frontend allows users to register, log in, create and manage requests, like and comment on requests, and update their profiles.
- ☑ **Protected Routes**: The backend has protected routes for profile updates, request actions, and request interactions, requiring valid authentication tokens.

Responsiveness across Different Devices

- ☐ **Responsive Design**: Frontend is responsive, supporting mobile, tablet, and desktop layouts.

Additional Features Specific to Application Purpose

- ☑ **Like Functionality**: Users can like requests with a duplicate-like prevention mechanism.
- ☑ **Comment Functionality**: Users can comment on requests, with initials displayed alongside each comment for better identification.
- ☑ **User Profile Management**: Users can update profile information, including uploading a profile picture.
- ☑ **File Upload (Profile Image)**: Multer is implemented to handle profile image uploads.
- ☑ **Fetch Open Requests**: Endpoint available for fetching all open requests, including user details associated with each request.

- ☐ **Request Acceptance**: Users can accept others' requests, and the request creator can approve or decline acceptances.
- ☑ **Error Handling and Validation**: Basic validation and error handling across routes, ensuring required fields and authenticated access.

System Documentation

Technical Architecture:

Overview of Components:

- **Backend**: server.js with Express handles server-side logic and API routes.
- **Authentication**: JWT (JSON Web Token) for secure, stateless user authentication.
- **Database**: MySQL for data storage.
- **File Upload**: multer for managing profile image uploads.

Technology Stack:

- **Database**: MySQL (Structured relational database).
- **Backend Framework**: Node.js with Express for server and API.
- **Authentication**: JWT for secure, stateless authentication.
- **Environment Management**: dotenv for managing sensitive configuration values like database credentials and JWT secrets.

Component Interaction:

- The **frontend** interacts with the backend by making HTTP requests to the Express API, using JWT tokens to authenticate sensitive endpoints.
- **MySQL** stores persistent data, with tables for users, requests, likes, and comments.
- **multer** handles image uploads, allowing profile images to be stored on the server and referenced in the database.

Database Schema

Tables:

Users:

Stores user information and is the primary table for user-related data.

- **id**: INT(11) — Primary Key, Auto Increment, Unique ID for each user.
- **name**: VARCHAR(100) — First name of the user.
- **surname**: VARCHAR(255) — Last name or surname of the user.
- **email**: VARCHAR(100) — User's email address, unique to each user.
- **password**: VARCHAR(255) — User's password, stored as a hash for security.
- **profile_image**: VARCHAR(255) — Path or filename of the user's profile image.
- **bio**: TEXT — A short biography or description provided by the user.
- **created_at**: TIMESTAMP — The date and time when the user account was created.

Requests:

Holds travel request details created by users. This table has a foreign key reference to users.

- **id**: INT, Primary Key, Auto Increment, Unique ID for each request.
- **user_id**: INT, Foreign Key referencing users(user_id), identifies the request creator.
- **request_status**: ENUM ('open', 'closed', 'canceled'), Current status of the request.
- **created_at**: TIMESTAMP, Automatically set when the request is created.
- **meeting_time**: DATETIME, Scheduled meeting time for the request.
- **start_location**: placeholder.
- **end_location**: placeholder.
- **request_type**: ENUM, Type of request (e.g., 'Walk', 'Trip', 'Other').
- **start_lat**: FLOAT, Latitude for the starting location.
- **start_lng**: FLOAT, Longitude for the starting location.
- **end_lat**: FLOAT, Latitude for the ending location.

- **End_lng**: FLOAT, Longitude for the ending location.

Likes:

Tracks the likes given to requests by users, forming a many-to-many relationship between users and requests.

- **id**: INT(11) — Primary Key, Auto Increment, Unique ID for each like entry.
- **request_id**: INT(11) — Foreign Key referencing requests(id), indicates the liked request.
- **user_id**: INT(11) — Foreign Key referencing users(id), indicates the user who liked the request.

Comments:

Stores comments left by users on requests, allowing users to discuss or respond to requests.

- **id**: INT(11) — Primary Key, Auto Increment, Unique ID for each comment.
- **request_id**: INT(11) — Foreign Key referencing requests(id), links the comment to a specific request.
- **user_id**: INT(11) — Foreign Key referencing users(id), identifies the user who made the comment.
- **comment**: TEXT — The content of the comment.
- **created_at**: TIMESTAMP — The date and time when the comment was created.

User Interface (UI) Design

Screens:

1. **Home Page**
    a. **Hero Section**: A welcoming hero section that displays a background image, a greeting message, a brief tagline, and a prominent "Get Started" button.
    b. **Features Section**: Highlights the app's main features, each represented by an icon and short description.
    c. **How It Works Section**: Shows the app's core steps with a visual flow, including arrows between each step.
    d. **Footer**: Contains the app's copyright information.
    e. **Navigation and Icon**: The navbar shows a user icon next to a greeting with the user's name and links to the profile page for easy access.

2. **Profile Page**
    a. **User Information**: Displays editable profile details, including name, surname, bio, and profile image. Users can upload a profile picture, change their name or bio, and view their email.

3. **Create a Request Page**
    a. **Input Fields**: A form where users can input details to create a new request, including:
        i. **Start Location**: Text input or location picker for the starting point of the trip.
        ii. **End Location**: Text input or location picker for the destination.
        iii. **Meeting Date & Time**: Date and time picker for when the trip will start.
        iv. **Trip Type**: Dropdown to select the type of trip (e.g., Walk, Trip, Errand).

v. **Submit Button**: A button to save the new request.

4. View Requests Page

1. **Detailed View**: Provides a detailed view of each request, including start and end points on Google Maps, meeting time, request type, and current status.
2. **Interactions**: Users can like and comment on requests from other users. Comments are displayed with the commenter's name and profile details for easy recognition.

API Endpoints:

User Management:

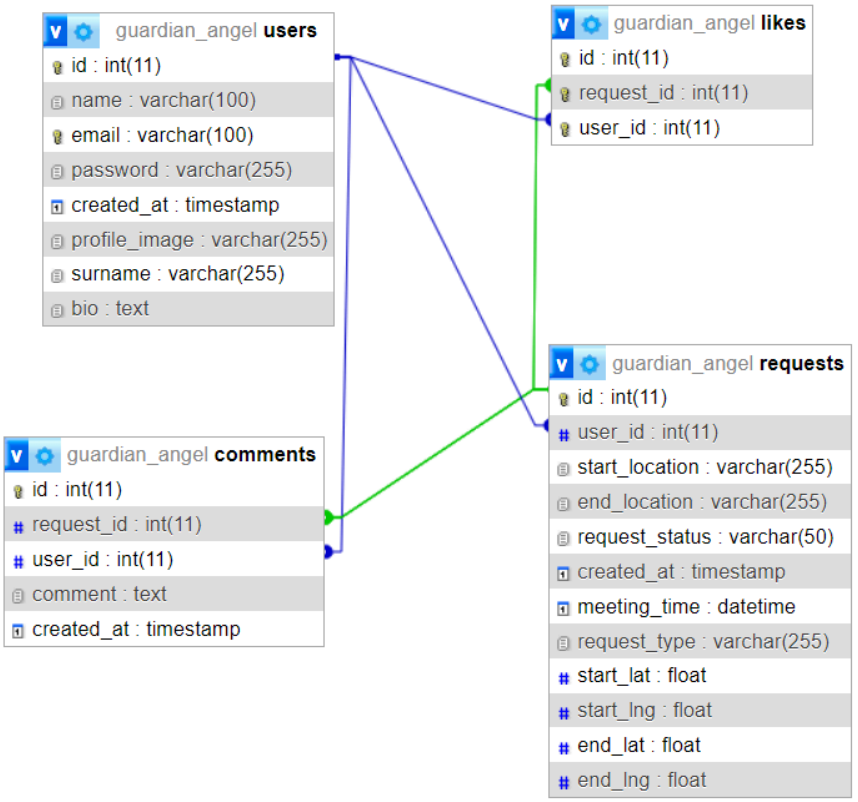
- **POST /register**: Registers a new user with credentials.
- **POST /login**: Authenticates a user, providing a JWT upon success.
- **GET /user/profile**: Fetches the logged-in user's profile.
- **POST /user/profile/update**: Updates profile information (name, surname, bio, and image).

Request Management:

- **POST /request**: Creates a new travel request.
- **POST /user/request/update**: Updates an existing request. Automatically reopens it if it was previously canceled.
- **POST /user/request/reopen**: Reopens a closed request.
- **POST /user/request/cancel**: Cancels a request.
- **DELETE /user/request/**: Deletes a request and associated comments/likes.
- **GET /user/requests**: Retrieves user's requests, with expired ones updated to "closed".

Request Interactions:

- **POST /requests//like**: Likes a request.
- **POST /requests//comment**: Adds a comment to a request.
- **GET /requests**: Lists all open requests.


guardian_angel users
id : int(11)
name : varchar(100)
email : varchar(100)
password : varchar(255)
created_at : timestamp
profile_image : varchar(255)
surname : varchar(255)
bio : text
guardian_angel likes
id : int(11)
request_id : int(11)
user_id : int(11)
guardian_angel requests
id : int(11)
user_id : int(11)
start_location : varchar(255)
end_location : varchar(255)
request_status : varchar(50)
created_at : timestamp
meeting_time : datetime
request_type : varchar(255)
start_lat : float
start_lng : float
end_lat : float
end_lng : float
guardian_angel comments
id : int(11)
request_id : int(11)
user_id : int(11)
comment : text
created_at : timestamp

Problem Statement

Many women face safety concerns when traveling alone or in unfamiliar areas. **Guardian Angel** addresses this societal issue by providing a platform where women can create travel requests for other women. Women in the community can respond by liking, commenting, or offering company. This system encourages support, which helps alleviate safety concerns, empowering women to feel more secure in their travels.