Food Blog Project Documentation

Project Overview

The Food Blog project is a web application that allows users to create, share, and explore food-related blogs. Users can register as either Clients or Vendors, where Clients can browse and interact with blogs, and Vendors can manage orders for food-related services. The application provides features such as blogging, commenting, following users, searching, and order management, all integrated with a backend API.

The project consists of a frontend built with HTML, CSS, and JavaScript, and a backend API running on `http://localhost:5000/api`. The frontend communicates with the backend to perform actions like user authentication, blog management, and notifications.

Project Structure

The project is organized into the following frontend files, each serving a specific purpose:

- **index.html**: The homepage displaying a list of recent blogs.
- **blog.html**: A detailed view of a single blog post, including comments and like functionality.
- **search.html**: A search page to find blogs by tags or keywords.
- **feed.html**: A personalized feed showing blogs from followed users.
- **profile.html**: A user profile page displaying the user's blogs, notifications, and orders (for Vendors).
- **login.html**: A login page for users to access their accounts.
- **register.html**: A registration page for new users to create an account.

Backend API

The backend API (running on `http://localhost:5000/api`) provides the following endpoints:

- **Users**:
  - `POST /api/users/register`: Register a new user.
  - `POST /api/users/login`: Authenticate a user and return a JWT token.
  - `GET /api/users/:id`: Get user details by ID.
- **Blogs**:
  - `GET /api/blogs`: Retrieve a list of all blogs.
  - `GET /api/blogs/:id`: Retrieve a specific blog by ID.
  - `POST /api/blogs`: Create a new blog (authenticated).
- **Comments**:
  - `GET /api/comments/:blogId`: Retrieve comments for a specific blog.

    - `POST /api/comments`: Add a comment to a blog (authenticated).
- **Follows**:
    - `POST /api/follows/:userId`: Follow a user (authenticated).
    - `DELETE /api/follows/:userId`: Unfollow a user (authenticated).
    - `GET /api/follows/feed`: Retrieve blogs from followed users (authenticated).
- **Notifications**:
    - `GET /api/notifications`: Retrieve notifications for the authenticated user.
    - `PUT /api/notifications/:id/read`: Mark a notification as read (authenticated).
- **Orders** (for Vendors):
    - `GET /api/orders`: Retrieve orders for the authenticated Vendor.
    - `PUT /api/orders/:id`: Update the status of an order (authenticated).

Features

The Food Blog application includes the following features:

1. **User Authentication**:
    - Users can register as either a Client or Vendor (`register.html`).
    - Users can log in to access personalized features (`login.html`).
    - JWT tokens are stored in `localStorage` for session management.
2. **Blog Management**:
    - Users can view a list of recent blogs (`index.html`).
    - Users can view a detailed blog post, including comments and likes (`blog.html`).
    - Authenticated users can create blogs (functionality to be implemented in a future `create-blog.html` page).
3. **Social Features**:
    - Users can follow/unfollow other users (`profile.html`).
    - Users can view a personalized feed of blogs from followed users (`feed.html`).
    - Users can like and comment on blogs (`blog.html`).
4. **Search Functionality**:
    - Users can search for blogs by tags or keywords (`search.html`).
5. **Profile Management**:
    - Users can view their own or other users' profiles (`profile.html`).
    - The profile page displays the user's blogs, notifications, and orders (for Vendors).
6. **Notifications**:
    - Users receive notifications for actions like new comments or follows (`profile.html`).
    - Notifications can be marked as read.
7. **Order Management (Vendors Only)**:
    - Vendors can view and update the status of orders placed by Clients (`profile.html`).

File Descriptions

1. `index.html`

- **Purpose**: The homepage of the application.
- **Features**:
  - Displays a list of recent blogs with titles, excerpts, tags, and author information.
  - Includes navigation links to other pages.
- **API Integration**:
  - Fetches blogs from `GET /api/blogs`.

2. `blog.html`

- **Purpose**: Displays a single blog post with its details.
- **Features**:
  - Shows the blog title, body, tags, and author.
  - Allows users to like the blog and add comments (authenticated users only).
  - Displays a list of comments.
- **API Integration**:
  - Fetches blog details from `GET /api/blogs/:id`.
  - Fetches comments from `GET /api/comments/:blogId`.
  - Posts comments via `POST /api/comments`.

3. `search.html`

- **Purpose**: Allows users to search for blogs.
- **Features**:
  - Provides a search bar to filter blogs by tags or keywords.
  - Displays search results as a list of blog cards.
- **API Integration**:
  - Fetches blogs from `GET /api/blogs` and filters them client-side.

4. `feed.html`

- **Purpose**: Displays a personalized feed for authenticated users.
- **Features**:
  - Shows blogs from users the authenticated user follows.
- **API Integration**:
  - Fetches followed users' blogs from `GET /api/follows/feed`.

5. `profile.html`

- **Purpose**: Displays a user's profile.

- **Features**:
  - Shows the user's username, role, and blogs.
  - Displays notifications and allows marking them as read.
  - For Vendors, shows a list of orders with the ability to update their status.
  - Allows following/unfollowing other users (if viewing another user's profile).
- **API Integration**:
  - Fetches user details from `GET /api/users/:id`.
  - Fetches blogs from `GET /api/blogs`.
  - Fetches notifications from `GET /api/notifications`.
  - Fetches orders from `GET /api/orders` (for Vendors).
  - Updates order status via `PUT /api/orders/:id`.
  - Follows/unfollows users via `POST/DELETE /api/follows/:userId`.

6. `login.html`

- **Purpose**: Allows users to log in to their accounts.
- **Features**:
  - Provides a form for email and password input.
  - Displays error messages for invalid credentials.
- **API Integration**:
  - Authenticates users via `POST /api/users/login`.

7. `register.html`

- **Purpose**: Allows new users to create an account.
- **Features**:
  - Provides a form for username, email, password, and role (Client or Vendor).
  - Displays error messages for invalid input.
- **API Integration**:
  - Registers users via `POST /api/users/register`.

Technical Details

Frontend Technologies

- **HTML**: Structure of the web pages.
- **CSS**: Embedded CSS for styling (Tailwind CSS styles were emulated due to CSP restrictions).
- **JavaScript**: Handles interactivity, API requests, and DOM manipulation.
- **LocalStorage**: Stores JWT tokens for user authentication.

Security

- **Content Security Policy (CSP)**:
  - Restricts resources to `'self'` for most directives.
  - Allows inline styles and scripts (`'unsafe-inline'`).
  - Permits API requests to `http://localhost:5000`.
  - Images are restricted to `'self'` and `data:` URLs.
- **Authentication**:
  - Uses JWT tokens for session management.
  - Tokens are validated on each authenticated request.

Limitations

- External resources (e.g., Tailwind CSS, Font Awesome) were removed due to CSP restrictions.
- Images are replaced with placeholders (e.g., Emojis) due to CSP limitations on external image sources.
- Blog creation functionality is not implemented in the current frontend (requires a `create-blog.html` page).

Usage Instructions

Prerequisites

1. **Backend Setup**:
   - Ensure the backend API is running on `http://localhost:5000/api`.
   - The backend should support the endpoints listed above.
2. **Frontend Setup**:
   - Place all HTML files (`index.html`, `blog.html`, etc.) in a directory.
   - Serve the files using a local server (e.g., `python -m http.server 8000`).

Running the Application

1. Start the backend API server.
2. Open a terminal in the directory containing the HTML files.
3. Run a local server:
4. `python -m http.server 8000`
5. Open a browser and navigate to `http://localhost:8000/index.html`.

User Workflow

1. **Register**:
   - Go to `register.html` to create a new account.
   - Choose a role (Client or Vendor).
2. **Login**:

    - Go to `login.html` to log in with your credentials.
3. **Explore Blogs**:
    - Visit `index.html` to see recent blogs.
    - Use `search.html` to find specific blogs.
    - View a blog's details on `blog.html`.
4. **Interact**:
    - Comment on or like blogs (authenticated users only).
    - Follow users from their profile (`profile.html`).
    - View your feed on `feed.html`.
5. **Manage Profile**:
    - Check your blogs, notifications, and orders (if a Vendor) on `profile.html`.
6. **Logout**:
    - Click the "Logout" button in the navigation bar.

Future Improvements

1. **Blog Creation Page**:
    - Add a `create-blog.html` page to allow users to create new blogs.
2. **Image Upload**:
    - Implement image upload functionality for blogs (requires backend support and CSP adjustments).
3. **Enhanced Search**:
    - Implement server-side search to improve performance.
4. **Responsive Design**:
    - Improve responsiveness for mobile devices.
5. **Real-time Notifications**:
    - Add WebSocket support for real-time notification updates.

Troubleshooting

- **API Errors**:
    - Ensure the backend API is running on `http://localhost:5000`.
    - Check the browser console for error messages.
- **Authentication Issues**:
    - Verify that the JWT token is correctly stored in `localStorage`.
    - Ensure the token is sent in the `Authorization` header for authenticated requests.
- **CSP Violations**:
    - If resources fail to load, check the CSP settings in the `<meta>` tag.
    - Ensure no external resources are used.

Conclusion

The Food Blog project provides a functional platform for food enthusiasts to share and explore recipes and food-related content. With its user-friendly interface and integration with a backend API, it supports core features like blogging, social interaction, and order management. Future enhancements can further improve the user experience and scalability of the application.