BLOG VERSE DOCUMENTATION

Pulkit Mangla

Table Of Contents

Introduction

Blog Verse is a blogging platform that allows users to explore a diverse range of blogs, with personalized recommendations for related reads. Users can also start their own blogs and publish content seamlessly. Furthermore, they can subscribe to their favourite authors to receive timely notifications about newly published posts.

Website Link: https://bloggverse.netlify.app

Architecture

Overview

Blog Verse is made using MERN (MongoDB, Express.js, React, Node.js) stack. The architecture follows a standard client-server model.

Technologies Used

MongoDB: NoSQL database for storing blog posts, user information (with sensitive information like passwords hashed), JSON web tokens, sessions data, notifications etc.

Express.js: Web application framework for Node.js , used to build the backend server.

React: JavaScript library for building user interface.

Node.js: JavaScript runtime for executing server-side code.

System Design

Frontend

- **React:** Used for building user interface.
- **React Router:** For client-side routing.
- **Axios:** For making HTTP requests to the backend.

Backend

- **Node.js and Express.js:** For building the RESTful API.
- **JWT (JSON Web Tokens):** For issuing tokens to authenticated users.
- **Passport.js:** For handling user authentication with local and Google users.
- **Mongoose:** ODM(Object Data Modelling) library for MongoDB.

Database

- **MongoDB Atlas:** Cloud-based Mongo-DB service.
- **Schema Design:**

- **User:** Contains user information such as username, password (hashed), email, and phone number.

- **Profiles:** Other user information like his/her subscriptions, subscribers, profile picture, cover picture, no. of blogs posted etc.

- **Post:** Contains blog posts details such as title, content, author , timestamps, likes and comments.

- **Notifications:** User notifications with their redirect links.

- **Tokens:** JWT tokens for authenticating users.

- **Sessions:** Contains session information, including session ID, cookie details, and expiration information.

API Design

Endpoints

User

- POST /signup: Register a new user locally.
- POST /login: Authenticate a user using local strategy.
- POST /logout: Log out a user by deleting tokens and session data.
- POST /refresh-token: Refresh JWT token.

- GET /google: Initiate Google OAuth authentication.
- GET /google/callback: Handle Google OAuth callback and authenticate user.
- GET /profile/:email: Retrieve user profile by email.
- GET /userPosts/:email: Retrieve posts by a specific user email.

- PUT /updateProfile/:id: Update user profile.

Blog Post

- POST /createBlog: Create a new blog post.

- GET /getPosts: Retrieve all blog posts.
- GET /postById/:id: Retrieve a single blog post by post id.
- GET /getRelatedPosts/:id: Retrieve posts related to a specific post.

- PUT /updatePosts: Update a blog post.
- PUT /putComment: Add a comment to a blog post.
- PUT /like: Like a blog post.

- DELETE /deletePost/:id: Delete a blog post.

Notifications

- POST /sendNotif: Send notification about newly added blog to all the user subscribers.
- GET /getNotifs/:email: Retrieve notifications for a specific user.

- DELETE /deleteNotifs/:email: Delete notifications for a specific user.

User Authentication

- **Passport.js**: Used for user authentication with both local and Google strategies.
  - **Local Strategy:** Users authenticate using username and password.
  - **Google Strategy:** Users authenticate using their Google account.

- **JWT Authentication:** After successful authentication with Passport.js, a JWT is issued to the client, which is then included in the Authorization header of subsequent requests.

- **Session Management:** Sessions are stored in the database and managed using cookies.

- **Middleware:** Authentication middleware protects routes that require user authentication.

Deployment

Frontend:

Deployed on Netlify.

Backend:

Deployed on Render.

Environment Variables:

Securely manage environment variables using .env files for local development and platform-specific configuration for production.

Future Improvements

- **Scalability**: Implement load balancing and horizontal scaling for handling increased traffic.
- **SEO Optimization**: Enhance the website for better search engine visibility.
- **Content Management**: Add an admin panel for easier management of blog posts and comments.
- **Real-time Features**: Integrate WebSockets for real-time updates and notifications.