Design Report: Code Connect (Channel-Based Tool for Programming Issues)

Introduction

Project Overview

Code Connect is a modern, channel-based discussion platform specifically designed for programming-related questions and technical discussions. The platform aims to provide developers with a structured and intuitive environment to share knowledge, ask questions and solve programming queries. By organizing discussions into dedicated channels, Code Connect creates a focused space for technical discussions, making it easier for developers to find relevant information and contribute to meaningful solutions.

Purpose and Objective

The primary objective of Code Connect is to help developers interact and help each other out.

The platform seeks to:

- Enable knowledge sharing and problem-solving
- Facilitate organized discussions
- Foster collaboration among developers
- Provide a User-Friendly interface for technical discussions

System Architecture

Code Connect follows a modern three-tier architecture pattern, consisting of :

- Backend Layer (Node.js/Express)
- Frontend Layer (React.js)
- Database Layer (MySQL)

Backend Architecture

The backend is structured as a RESTful API using Node.js and Express:

- **API Layer**
    - WebSocket support for real-time messaging
    - Authentication middleware using JWT
    - Restful endpoints for CRUD operations

- **Service Layer**
    - Data Validation
    - Error handling

- **Data Access Layer**
    - Database interactions
    - Query Optimization

Frontend Architecture

The frontend was built using Recat.js

- **State Management**
    - React Context API for Global State
    - Local React Component using useState and react hooks

- **Routing System**
    - Channel-based routing
    - Protected routes for authenticated users
    - Usage of reacter-router-dom

- **User Interface Components**
    - Authentication pages (Login, Register)
    - Channel Management page to manage channels
    - ChannelDetail page to manage the messages, replies and channel management
    - Navigation Bar for navigating between pages

Database Architecture

The System uses MySQL for the database.

- User Authentication data
- Channel Data, Message Data, and Reply Data are all stored in individual tables
- Implementation of proper data types for each field
- Key entries are represented by distinctively
- Minimizes data redundancy and ensures consistency across tables

The platform uses Docker for the containarization of the system including two different Dockerfiles one for frontend and one for the Backend and a docker-compose.yml for composing the containers.

TECHNICAL DESIGN

Frontend Design

The src/ directory contains all the frontend's source code, including .jsx files, CSS files for styling, and api.js for API calls.

The src directory contains:

- Components: A directory containing the authentication pages and protected routes
    - Login.jsx and Logout.jsx : Contains the source code for login and logout pages
    - Register.jsx: contains the source code for the Register page
    - ProtectedRoute.jsx: Ensures and checks whether the user is authenticated or not
- Pages: A directory which contains all the pages which are displayed to the user
    - AdminDashboard.jsx : Contains the source code for the admin dashboard
    - Bookmarks.jsx: Contains the source code for the bookmark page
    - ChannelDetail.jsx: Contains the source code for managing individual channels
    - Channels.jsx: It contains the source code for the Channels dashboard which allows user to see and join channels
    - SearchPage.jsx: It contains the source code for the search page and search bar

    - LadningPage.jsx: It is the initial landing page where the user navigates when the app opens
    - UserAnalytics.jsx: It contains the source code for displaying user analytics

- Services
    - **Api.js**
    - It is the file responsible for serving as the API communication layer, and helping the frontend make HTTPS requests to the backend.
    - It uses Axios to interact while creating axios instance with a base URL and default JSON header.

- Styles**:** This Directory contains all the styles files for the pages
- App.jsx: This is the main component of the app, which sets up routing, navigation and authentication for Code Connect
- DockerFile: This serves as dockerfile for the frontend

The front-end uses the react-router-dom to manage navigation between different pages without full page reloads.

The navigation component in App.jsx handles the navigation between different pages. And includes links to login, registration, logout, channels, admin dashboard, user analytics, bookmarks and search.

It also has conditional Rendering, so the navigation bar changes according to the current page it is on.

Backend Design

The backend is built using node.js and Express, following RESTful principles. Each endpoint follows REST conventions and includes proper error handling, input validation and response formatting.

The endpoints are organized into logical groups:

- Authentication endpoints (/aoi/v1/auth/)
- Channel management endpoints (/api/v1/channels)
- Message handling endpoints (/api/v1/messages)
- User Management endpoints (/api/v1/users)

The Endpoints:

- POST /api/v1/auth/login: To authenticate and generate authentication token
- POST /api/v1/auth/refresh: To generate a new access token
- GET /api/v1/users/profile: Gets the current user profile
- POST /api/v1/channels: Creates a Channel
- GET /api/v1/channels: Lists all the available Channels
- GET /api/v1/channels/:channelId: Gets a particular channel
- PUT /api/v1/channels/:channelId: Updates a channel information
- DELETE /api/v1/channels/:channelId: Delets a channel
- POST /api/v1/channels/:channelId/messages: Submits a message in a channel
- GET /api/v1/channels/:channelId/messages: Gets a message in a channel
- PUT /api/v1/messages/:messageId: Updates a message in a channel
- DELETE /api/v1/messages/:messageId: Deletes a message in a channel
- GET /api/v1/search/messages: Searches a particular message
- GET /api/v1/search/channels: Searches for a particular channel

The backend also follows a layered approach where it has the following layers:

- **Route Layer:** Handles Https requests and routes them to appropriate controllers
- **Controller Layer:** Processes requests and manages logic
- **Service Layer:** Implements logic and data processing
- **Repository Layer:** Handles data access and data base operations

Authentication System:

The authentication system implements a secure, token-based approach using JSON Web Tokens(JWT). It includes:

1. **User Registration**
2. **User Login**
3. **Token Management**

Database Design

Database System: MySQL 8.0

Type: Relational Database

Selection: Provides high performance for structured data, excellent Node.js integration, proven reliability in production environments and ACID compliance for data integrity.

The database is structured, has relationship management and Index optimization.

Database Model

USERS : Manages user accounts and authentication

Key Attributes:

- Id: Unique identifier (Primary Key)
- Username: unique name for each user
- Password: secure password storage
- Role: User permission Level
- displayName: displayname for a user
- isAdmin: used to check if a user is admin or not
- Created_at: timestamp when a account was created

CHANNELS: Organises channels

Key Attributes:

- Id: Unique identifier (Primary Key)
- Name: Channel title
- Description: Channel purpose
- Created_at: timestamp when the channel was created
- Created_by: who created the channel

MESSAGES: Stores communication content

Key Attributes:

- Id: Unique identifier (Primary Key)
- Content: Message Text

- Channel_id: id for each channel
- User_id: id of the user who sends the message
- Titile: titile of the message
- Image_url: url for the attached image
- Created_at: timestamp the message was created
- Upvotes: no.of upvotes
- Downvotes: no.of downvotes

REPLIES: Stores replies to the messages

Key Attributes:

- Id: Unique Identifier(Primary Key)
- Message_id: the message to which the reply got posted
- User_id: id of the user who send the reply
- Content: content of the reply
- Image_url: the URL of image in the reply
- Parent_reply_id: reply id of parent id
- Created_id: timestamp for creation of reply
- Upvotes: The number of upvotes of on a reply
- Downvotes: the number of downvotes of an reply

Bookmarks: Stores bookmarks for a user

Key Attributes:

- **Id:** identifier(Primary Key)
- User_id: the id of user
- Message_id: id of the message
- Created_at: timestamp for the bookmark

Votes: Stores the upvotes and Downvotes

Key Attribute:

- Id: identifier(Primary key)
- User_id: user id of the user
- Message_id: message id of the message
- Reply_id: id of reply
- Created_at: timestamp of the vote
- Vote_type: upvote and downvote

DATA INTEGRITY

- Primary Key constraints on all tables
- Foreign Key constraints for referential integrity
- Unique constraints on username and bookmark combinations
- NOT NULL constraints on required fields
- Default values for timestamp and vote counts

DATA PROTECTION

- Password hashing using bcrypt
- Input validation and sanitization
- Prepared statements for all queries
- Role-based access control