# **Part 1: MERN Stack Development**

## 1.1 MySQL

### MySQL Schema Design :

● **Design a MySQL schema for a blog application with the following**

**entities: User, Post, and Comment. Include relevant fields and**

**relationships.**

**1.User Table**

CREATE TABLE User (

user\_id INT PRIMARY KEY AUTO\_INCREMENT,

username VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL,

password VARCHAR(255) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

2.Post Table

CREATE TABLE Post (

post\_id INT PRIMARY KEY AUTO\_INCREMENT,

title VARCHAR(255) NOT NULL,

content TEXT NOT NULL,

author\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (author\_id) REFERENCES User(user\_id)

);

3.Comment Table

CREATE TABLE Comment (

comment\_id INT PRIMARY KEY AUTO\_INCREMENT,

text TEXT NOT NULL,

author\_id INT,

post\_id INT,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (author\_id) REFERENCES User(user\_id),

FOREIGN KEY (post\_id) REFERENCES Post(post\_id)

);

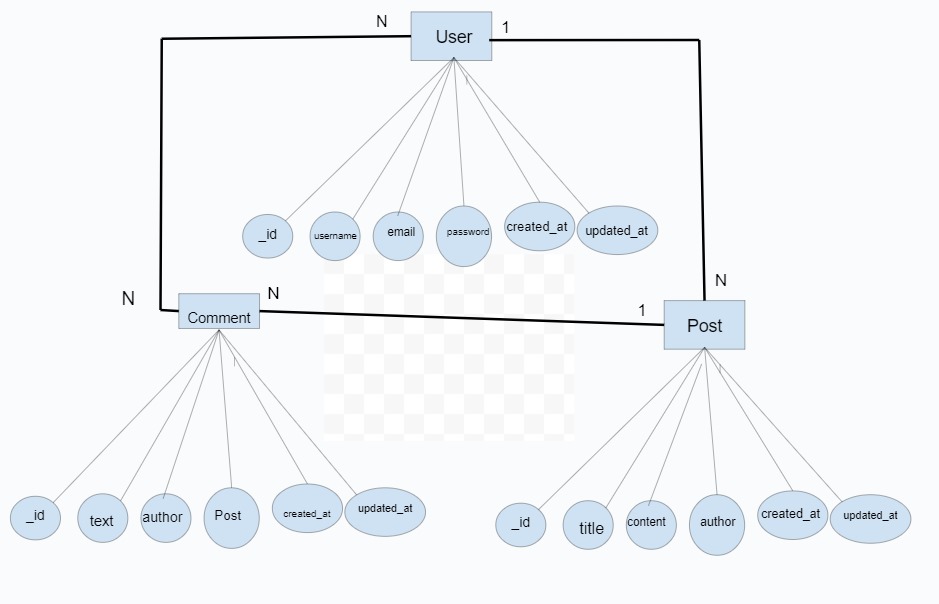
This schema establishes the following relationships:

One-to-Many relationship between User and Post (one user can have multiple posts).

One-to-Many relationship between User and Comment (one user can have multiple comments).

One-to-Many relationship between Post and Comment (one post can have multiple comments).

ER Diagram:



MySQL Query:

● Write a MySQL query to find all posts created by a specific user,

including the user details.

**Query:**

SELECT

Post.post\_id,Post.title,Post.content,

Post.created\_at AS post\_created\_at,

Post.updated\_at AS post\_updated\_at,

User.user\_id,User.username,

User.email,User.created\_at AS user\_created\_at,

User.updated\_at AS user\_updated\_at

FROM

Post JOIN User ON Post.author\_id = User.user\_id

WHERE

User.user\_id = 1;

## 1.2 Express.js

### Express.js Middleware :

● **Create an Express.js middleware that logs the timestamp and the**

**requested URL for every incoming request.**

**Program:**

const logRequest = (req, res, next) => {

  const timestamp = new Date().toISOString();

  console.log(`${timestamp}`);

  next();

};

module.exports = logRequest;

const express = require("express");

const logRequest = require('./path/to/logRequestMiddleware');

const app = express();

app.use(logRequest);

app.listen(3000, async () => {

     console.log("Server is running at port 3000");

   });

const logRequest = (req, res, next) => {

  const timestamp = new Date().toISOString();

  console.log(`${timestamp}`);

  next();

};

module.exports = logRequest;

const express = require("express");

const logRequest = require('./path/to/logRequestMiddleware');

const app = express();

app.use(logRequest);

app.listen(3000, async () => {

     console.log("Server is running at port 3000");

   });

### Express.js Route :

**● Implement an Express.js route that retrieves a list of posts from the**

**database and returns it as JSON.**

Program:

const express = require("express");

const app = express();

app.use(express.json());

const { open } = require("sqlite");

const sqlite3 = require("sqlite3");

const path = require("path");

const dbPath = path.join(\_\_dirname, "blogs.db");

let db = null;

const initializeDBAndServer = async () => {

  try {

    db = await open({

      filename: dbPath,

      driver: sqlite3.Database,

    });

    app.listen(3000, async () => {

      console.log("Server is running at  3000");

    });

  } catch (e) {

    console.log(e.message);

  }

};

initializeDBAndServer();

app.get("/posts", async(request,response) => {

    const sqlQuery = `

        SELECT

             \*

        FROM

           Posts;

    `;

    const result = db.all(sqlQuery);

response.json(result);

});

Run:

node filename.js

## 1.3 React.js

### React.js Component:

**● Create a React.js functional component that displays a list of blog**

**posts. Each post should show the title, author, and a button to view the**

**full post.**

### React.js State Management:

**● Implement state management in React to handle the loading state**

**while fetching the blog posts. Show a loading indicator when posts are**

**being fetched.**

**Program:**

import {Component} from 'react'

import Loader from 'react-loader-spinner'

import Post from './post'

class Posts extends Component {

  state = {postsList: [], show: false,isLoading: true}

  componentDidMount() {

    this.getPostsDetails()

  }

  getFormattedData = data => ({

    postId: data.post\_id,

    title: data.title,

    content: data.content,

    authorId: data.author\_id,

    createdAt: data.created\_at,

  })

  getPosts = async () => {

    this.setState({isLoading: true})

    const apiUrl = `https://apis.ccbp.in/posts`

    const options = {

      headers: {

        Authorization: `Bearer ${'jwtToken'}`,

      },

      method: 'GET',

    }

    const response = await fetch(apiUrl, options)

    const fetchedData = response.json()

    const updatedPostsData = fetchedData.map(eachPost =>

      this.getFormattedData(eachPost),

    )

    this.setState({

      postsList: updatedPostsData,

      isLoading: true,

    })

  }

   renderLoadingView = () => (

     <div className="loader-container">

         <Loader type="ThreeDots" color="#0b69ff" height="50" width="50" />

    </div>

      )

  render() {

    const {postsList,isLoading} = this.state

    return (

      <div className="container">

           {isLoading && this.renderLoadingView() }

          {!isLoading && <ul>

              {postsList.map(eachPost => <Post key={eachPost.postId} post={eachPost} />}

            </u>

          }

      </div>

    )

  }

}

export default Posts

./Post/

class Post extends Component{

    state = {show: false}

    showFullPost = () => this.setState(prevState => ({show: !prevState.show}))

    render(){

           const {eachPost} = this.props

           const {postId, title, content, authorId, createdAt } = eachPost

           return (

               <li>

                 <h1>{postId} </h1>

                 <p>{content} </p>

                {show && (

                     <>

                       <p>{authorId}</p>

                       <p>{createdAt}</p>

                     </>

                  )}

                 <button onClick={this.showFullPost}> Show Full Post </button>

               </li>

               )

       }

}

export default Post

## 1.4 Node.js

### Node.js File System:

**● Write a Node.js script that reads a JSON file containing user data,**

**manipulates the data to include the total number of posts each user**

**has created, and writes the modified data back to a new JSON file**.

**Program:**

const fs = require('fs');

const readFile = (filePath) => {

  try {

    const data = fs.readFileSync(filePath, 'utf8');

    return JSON.parse(data);

  } catch (error) {

    console.error('Error reading the file:', error.message);

    process.exit(1);

  }

};

const writeFile = (filePath, data) => {

  try {

    fs.writeFileSync(filePath, JSON.stringify(data, null, 2));

    console.log('Modified data has been written to', filePath);

  } catch (error) {

    console.error('Error writing to the file:', error.message);

    process.exit(1);

  }

};

const calculateTotalPosts = (userData) => {

  const modifiedData = userData.map((user) => {

    const totalPosts = user.posts.length;

    return { ...user, totalPosts };

  });

  return modifiedData;

};

const inputFilePath = 'users.json';

const outputFilePath = 'modified\_users.json';

const userData = readFile(inputFilePath);

const modifiedUserData = calculateTotalPosts(userData);

writeFile(outputFilePath, modifiedUserData);

Run:

node filename.js

## Node.js API Integration :

● **Create a Node.js function that fetches data from an external API (e.g.,**

**JSONPlaceholder) and returns a list of users along with their posts.**

const axios = require('axios');

const getUsersWithPosts = async () => {

  try {

   const usersResponse = await axios.get('https://jsonplaceholder.typicode.co m/users');

    const users = usersResponse.data;

    const usersWithPosts = await Promise.all(users.map(async (user) => {

   const postsResponse = await axios.get(`https://jsonplaceholder.typicode.com /posts?userId=${user.id}`);

      const posts = postsResponse.data;

      return { ...user, posts };

    }));

    return usersWithPosts;

  } catch (error) {

    console.error('Error fetching data:', error.message);

    throw error;

  }

};

getUsersWithPosts()

  .then((result) => {

    console.log('Users with their posts:', result);

  })

  .catch((error) => {

    console.error('Error:', error.message);

  });

Run:

node filename.js