INSPECTION CODE
TEAM NAME - WALL STREET DEVS

Front End

1) **Home Page ➖**

```
function Home() {
  return (
    <Container className="text-center">
      <div class="d-flex justify-content-center">
          <img src={homeImg} alt="" className="w-50 shadow-lg rounded mt-5" />
      </div>
      <p className="py-4">
        Lorem ipsum, dolor sit amet consectetur adipisicing elit. Non tenetur
        accusamus impedit architecto excepturi temporibus eos ducimus magnam
        quod dolores laborum magni, mollitia enim commodi beatae eius
        Laboriosam obcaecati in, sapiente impedit ipsum excepturi quisquam velit
        ad sequi ex nulla vitae alias est provident voluptatem, repudiandae,
        explicabo repellendus! Quos.
      </p>
    </Container>
  );
}

export default Home;
```

Explanation -
Here we are creating a home landing JS page which will contain the website name and information.

2) **Login Page -**

```
function Login() {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();

  //get user state from redux
  let { userObj, isError, isLoading, isSuccess, errMsg } = useSelector(
  (state) => state.user
);

//get dispatch function to call action creator functions
let dispatch = useDispatch();

//get navigate function to navigate programmatically
let navigate = useNavigate();

//when login form is submitted
const onFormSubmit = (userCredentialsObject) => {
   // console.log(userCredentialsObject)
  if (userCredentialsObject.userType === "user") {
    dispatch(userLogin(userCredentialsObject));
  }

  if (userCredentialsObject.userType === "admin") {
    alert("Admin devoloplment in progress...");
    // dispatch(userLogin(userCredentialsObject));
  }
};

//this to be executed when either isSuccess or isError changed
useEffect(() => {
  if (isSuccess) {
    navigate("/userdashboard");
  }
}, [isSuccess, isError]);

return (
  <div className="container">
    <p className="display-2 text-center text-primary">Login</p>

    {/* <img
      src={loginImg}
      width="300px"
      className="d-sm-block d-none mx-auto"
      alt=""
    /> */}
    <div className="row  ">
      <div className="col-12 col-sm-8 col-md-6  mx-auto">
        <Form onSubmit={handleSubmit(onFormSubmit)}>
          <Form.Group className="mb-3">
            <Form.Label>Select type of User</Form.Label> <br />
  {/* user type */}
  <Form.Check inline type="radio" id="user">
    <Form.Check.Input
      type="radio"
      value="user"
      {...register("userType", { required: true })}
    />
    <Form.Check.Label>User</Form.Check.Label>
  </Form.Check>
  <Form.Check inline type="radio" id="admin">
    <Form.Check.Input
      type="radio"
      value="admin"
      {...register("userType", { required: true })}
    />
    <Form.Check.Label>Admin</Form.Check.Label>
  </Form.Check>
</Form.Group>
{/* username */}
<Form.Group className="mb-3">
  <Form.Label>Username</Form.Label>
  <Form.Control
    type="text"
    placeholder="Enter Username"
    {...register("username", { required: true })}
  />
  {/* validation error message for username */}
  {errors.username && (
    <p className="text-danger">* Username is required</p>
  )}
</Form.Group>

{/* password */}
<Form.Group className="mb-3">
  <Form.Label>Password</Form.Label>
  <Form.Control
    type="password"
    placeholder="Enter Password"
    {...register("password", { required: true })}
  />
  {/* validation error message for password */}
  {errors.password && (
    <p className="text-danger">* Password is required</p>
  )}
```

```
          </Form.Group>

          <Button variant="secondary" type="submit">
            Login
          </Button>
        </Form>
      </div>
    </div>
  </div>
 );
}

export default Login;
```

Explanation -
Here, we are creating a login page containing three fields i.e., the first field asks if it is user or admin and the other fields ask for username and password.

We are getting userObj value from redux which will tell if someone is logged in or not.
-> We are using the Form tag to create a from here which contains the fields to enter the value such as username and password.
Then we have a login button which will submit the form and onFormSubmit function will be called.
->In the submit function we check if the user is admin or user and we call the appropriate dispatch function according to that.

3) **Register Functionality -**

```
function Signup() {
 const {
  register,
  handleSubmit,
  formState: { errors },
 } = useForm();

 // state for image
 const navigate = useNavigate();

 const onFormSubmit = async (userObj) => {
  try {
   // // create FormData object
   const formData = new FormData();
   // // append values to it
```

```
    formData.append("userObj", JSON.stringify(userObj));

    // http post req
    console.log(formData);
    const response = await axios.post("http://localhost:4000/user-api/create-user", formData, {
      headers: {
        "Content-Type": 'application/json',
      },
    });
    console.log(response);
    alert(response.data.message);

    if (response.data.message === "New User created") {
      // navigate to login
      navigate("/login");
    }
  } catch (error) {
    console.error(error);

    if (axios.isAxiosError(error)) {
      // Axios-specific error handling
      if (!error.response) {
        // Network error
        console.error("Network error:", error);
      } else {
        // Server responded with an error status
        console.error("Server error:", error.response.data);
      }
    } else {
      // Other types of errors
      console.error("Unexpected error:", error);
    }

    alert("Something went wrong in creating user");
  }
};

return (
  <Container>
    <div className="display-2 text-center text-info">Signup</div>
    <div className="row">
      <div className="col-12 col-sm-8 col-md-6 mx-auto">
        <Form onSubmit={handleSubmit(onFormSubmit)}>
          {/* username */}
```

```
<Form.Group className="mb-3">
  <Form.Label>Username</Form.Label>
  <Form.Control
    type="text"
    placeholder="Enter Username"
    {...register("username", { required: true })}
  />
  {/* validation error message for username */}
  {errors.username && (
    <p className="text-danger">* Username is required</p>
  )}
</Form.Group>

{/* password */}
<Form.Group className="mb-3">
  <Form.Label>Password</Form.Label>
  <Form.Control
    type="password"
    placeholder="Enter Password"
    {...register("password", { required: true })}
  />
  {/* validation error message for password */}
  {errors.password && (
    <p className="text-danger">* Password is required</p>
  )}
</Form.Group>

{/* email */}
<Form.Group className="mb-3">
  <Form.Label>Email</Form.Label>
  <Form.Control
    type="text"
    placeholder="Enter email"
    {...register("email", { required: true })}
  />
  {/* validation error message for password */}
  {errors.email && (
    <p className="text-danger">* Email is required</p>
  )}
</Form.Group>
<Form.Group className="mb-3">
  <Form.Label>City</Form.Label>
  <Form.Control
    type="text"
```

```
          placeholder="Enter city"
          {...register("city", { required: true })}
        />
        {/* validation error message for password */}
        {errors.city && (
          <p className="text-danger">* City is required</p>
        )}
      </Form.Group>

      <Button variant="primary" type="submit">
        Signup <MdLogin />
      </Button>
    </Form>
   </div>
  </div>
 </Container>
 );
}

export default Signup;
```

Explanation - Here we are creating a signup page that will enable users to create an account on the website. It contains 4 fields which are username, password, city, and email and a signup button.

4) **USER DASHBOARD FUNCTIONALITY -**

```
function Userdashboard() {
 let { userObj } = useSelector((state) => state.user);
 const navigate = useNavigate();
 const handleNavigation = (path) => {
  navigate(path);
 };

 return (
  <>
   {/* <img
    src={userObj.profileImg}
    className="float-end m-5 profile-pic"
    alt=""
   /> */}
    <>
    <div style={{ display: 'flex', alignItems: 'center', marginTop:'40px', marginLeft:'30px' }}>
     <h3 style={{ marginRight: '10px' }}>Hello, {userObj.username}!</h3>
```

```
          <h5>Connect with your alumni network.</h5>
        </div>
        <div className="userdashboard-container">
          <div className="user-dashboard-card">
            <Link to="/userdashboard/profile" className="user-dashboard-card-link">
              <div className="user-dashboard-card-content">
                <h3>View Profile</h3>
              </div>
            </Link>
          </div>
          <div className="user-dashboard-card">
            <Link to="/new-post" className="user-dashboard-card-link">
              <div className="user-dashboard-card-content">
                <h3>Create New Post</h3>
              </div>
            </Link>
          </div>
          <div className="user-dashboard-card">
            <Link to="/userdashboard/edit-profile" className="user-dashboard-card-link">
              <div className="user-dashboard-card-content">
                <h3>Edit Profile</h3>
              </div>
            </Link>
          </div>
        </div>

        <div className="user-dashboard">
          <div className="user-dashboard-sidebar">
          </div>

          <div>
            <Routes>
              <Route path="/userdashboard/profile" element={<Userprofile />} />
              <Route path="/new-post" element={<NewPost />} />
            </Routes>
          </div>
        </div>
      </>
    </>
  );
}

export default Userdashboard;
```

Explanation -
After logging in The user will be shown its user dashboard which will show various options like creating a new post, view profile, edit profile and other functionalities will be added later.
On the nav bar user can see link to posts and events as well.

5) **Posts Functionality -**

```
const Posts = () => {
  const [posts, setPosts] = useState([]);
  const [currentPage, setCurrentPage] = useState(1);
  const [totalPages, setTotalPages] = useState(1);

  useEffect(() => {
    const fetchPosts = async () => {
      try {
        const response = await
axios.get(`http://localhost:4000/post-api/posts?visibility=public&page=${currentPage}`);
        setPosts(response.data.payload.posts);
        setTotalPages(response.data.payload.totalPages);
        console.log(response)
      } catch (error) {
        console.error("Error fetching posts:", error);
      }
    };

    fetchPosts();
  }, [currentPage]);

  const handlePageChange = (newPage) => {
    setCurrentPage(newPage);
  };

  return (
    <div className="container mt-5">
      <h1 className="mb-4">Posts</h1>
      <ul className="list-group">
        {posts.map(post => (
          <li key={post._id} className="list-group-item">
            <h3>{post.title}</h3>
            <p>{post.content}</p>
            <p>Category: {post.category}</p>

          </li>
        ))}
      </ul>
      {/* Pagination */}
      <nav className="mt-4">
        <ul className="pagination">
          {Array.from({ length: totalPages }, (_, i) => i + 1).map(pageNumber => (
            <li key={pageNumber} className={`page-item ${currentPage === pageNumber ?
'active' : ''}`}>
              <button className="page-link" onClick={() =>
handlePageChange(pageNumber)}>{pageNumber}</button>
            </li>
          ))}
        </ul>
      </nav>
    </div>
  );
};

export default Posts;
```

Explanation -

In this JS page we are showing all the posts, we are making use of axios library to make a call to an api which will give us a list of all the posts.
After that we are showing that list in the front end.

6) Create New Post Functionality -

```
function NewPost() {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm();
  const navigate = useNavigate();

  const onFormSubmit = async (postObj) => {
    try {

      const formData = new FormData();
      console.log(postObj);
      formData.append("postObj", JSON.stringify(postObj));

      console.log(formData);
      const response = await axios.post("http://localhost:4000/post-api/new-post", formData, {
        headers: {
          "Content-Type": 'application/json',
        },
      });

      console.log("New Post Added:", response.data);
      alert(response.data.message);
      if(response.data.message=="New Post Added"){
        navigate('/posts')
      }
    } catch (error) {
      console.error("Error adding new post:", error);
      alert(error)
    }
  };

  return (
    <div className="new-post-card" style={{ marginTop: '40px' }}>
      <h2>Create New Post</h2>
      <Form onSubmit={handleSubmit(onFormSubmit)}>
        <Form.Group className="mb-3" controlId="title">
          <Form.Label>Title: </Form.Label>
          <Form.Control
            type="text"
            placeholder="Enter Title.."
            {...register("title", { required: true })}
          />
          {errors.title && (
            <p className="text-danger">* Title is required</p>
          )}
        </Form.Group>

        <Form.Group className="mb-3" controlId="content">
          <Form.Label>Content: </Form.Label>
          <Form.Control
            as="textarea"
            placeholder="Enter Content.."
            {...register("content", { required: true })}
          />
          {errors.content && (
            <p className="text-danger">* Content is required</p>
          )}
        </Form.Group>

        <Form.Group className="mb-3" controlId="category">
          <Form.Label>Category:</Form.Label>
```

```
          <Form.Select
            {...register("category", { required: true })}
          >
            <option value="">Select Category</option>
            <option value="technology">Technology</option>
            <option value="education">Education</option>
            <option value="travel">Travel</option>
          </Form.Select>
          {errors.category && (
            <p className="text-danger">* Category is required</p>
          )}
        </Form.Group>

        <Form.Group className="mb-3" controlId="visibility">
          <Form.Label>Visibility:</Form.Label>
          <Form.Select
            {...register("visibility", { required: true })}
          >
            <option value="public">Public</option>
            <option value="private">Private</option>
          </Form.Select>
          {errors.visibility && (
            <p className="text-danger">* Visibility is required</p>
          )}
        </Form.Group>

        <Button variant="primary" type="submit">
          Create Post
        </Button>
      </Form>

    </div>
  );
}

export default NewPost;
```

Explanation -
When a user wants to create a new post it will go to this page and give The post title, content, category and privacy options to the user and when a user clicks submit it will create a new post and it will be visible in all posts section for the logged in user.

7) Navigation Bar -

```
function Header() {
  //get state from store
  let { userObj, isError, isLoading, isSuccess, errMsg } = useSelector(
    (state) => state.user
  );
  //get dispathc function
  let dispath = useDispatch();

  //get navigate function
  let navigate = useNavigate();

  //logout user
  const userLogout = () => {
    localStorage.clear();
    dispath(clearLoginStatus());
    navigate("/login");
  };

  return (
    <div>
      <Navbar collapseOnSelect expand="sm" bg="dark" variant="dark">
        <Container>
          <Navbar.Brand href="#home">Alma Mingle</Navbar.Brand>
          <Navbar.Toggle aria-controls="responsive-navbar-nav" />
          <Navbar.Collapse id="responsive-navbar-nav">
            <Nav className="ms-auto">
              {isSuccess !== true ? (
                <>
                  {/* These links can be visible when no user logged in */}
                  <Nav.Item>
                    <Nav.Link eventKey="1" as={NavLink} to="/">
                      Home
                    </Nav.Link>
                  </Nav.Item>

                  <Nav.Item>
                    <Nav.Link eventKey="2" as={NavLink} to="/signup">
                      Signup
                    </Nav.Link>
                  </Nav.Item>

                  <Nav.Item>
                    <Nav.Link eventKey="3" as={NavLink} to="/login">
                      Login
                    </Nav.Link>
                  </Nav.Item>

                  <Nav.Item>
                    <Nav.Link eventKey="4" as={NavLink} to="/contactus">
                      ContactUs
                    </Nav.Link>
                  </Nav.Item>
                </>
              ) : (
                <>
                  {/* This dropdown is visible only when a user is logged in */}
                  <Nav.Item>
                    <Nav.Link eventKey="1" as={NavLink} to="/userdashboard">
                      User Dashboard
                    </Nav.Link>
                  </Nav.Item>

                  <Nav.Item>
                    <Nav.Link eventKey="2" as={NavLink} to="/posts">
                      Posts
                    </Nav.Link>
                  </Nav.Item>

                  <Nav.Item>
                    <Nav.Link eventKey="3" as={NavLink} to="/userdashboard/events">
                      Events
                    </Nav.Link>
                  </Nav.Item>
                  <NavDropdown
                    title={userObj.username}
                    //id="collasible-nav-dropdown"
                     id="drop-down"
                  >
                    <NavDropdown.Item>Change password</NavDropdown.Item>

                    <NavDropdown.Divider />
                    <NavDropdown.Item onClick={userLogout}>
                      Logout
                    </NavDropdown.Item>
                  </NavDropdown>
                </>
```

```
          )}
        </Nav>
      </Navbar.Collapse>
    </Container>
  </Navbar>
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/signup" element={<Signup />} />
    <Route path="/login" element={<Login />} />
    <Route path="/contactus" element={<Contactus />} />
    <Route path="/posts" element={<Posts />} />
    <Route path="events" element={<Userprofile />} />
    <Route path="/new-post" element={<NewPost />} />
    <Route path="/userdashboard" element={<Userdashboard />}>
      <Route path="profile" element={<Userprofile />} />
      <Route path="" element={<Navigate to="profile" replace={true} />} />
    </Route>
  </Routes>
 </div>
);
}

export default Header;
```

Explanation -
This is the navigation bar that is shown on every page it contains login and signup options and when the user is logged in then it will show the user dashboard, posts, and events.

Back End

8) Connection of Backend with Frontend and Database -

```
app.use(exp.static(path.join(__dirname,'../Front-End/build')))

//DB connection URL
const dbUrl='mongodb+srv://Varun:Varun@cluster0.klf74.mongodb.net/';

//connect with mongoDB server
mclient.connect(dbUrl)
.then((client)=>{

  //get DB object
  let dbObj=client.db("commondatabase");
  //create collection objects
  let userCollectionObject=dbObj.collection("usercollection");
  let adminCollectionObject=dbObj.collection("admincollection");
  let postCollectionObject=dbObj.collection("postcollection");

  //sharing collection objects to APIs
  app.set("userCollectionObject",userCollectionObject);
  app.set("adminCollectionObject",adminCollectionObject);
  app.set('postCollectionObject',postCollectionObject)

  console.log("DB connection success")
})
.catch(err=>console.log('Error in DB connection ',err))
```

Explanation -

The main backbone of the whole code is the above one. The first line is used to connect the backend with the build folder of the front end. The rest of the code is used to connect the backend with the database.

9) **UserApi Routes -**

```
//create route to handle '/getusers' path
userApp.get(
  "/getusers", verifyToken,
  expressAsyncHandler(async (request, response) => {
    //get userCollectionObject
    let userCollectionObject = request.app.get("userCollectionObject");
    //get all users
    let users = await userCollectionObject.find().toArray();
    //send res
    response.send({ message: "Users list", payload: users });
  })
);

//create route to user login
userApp.post(
  "/login",
  expressAsyncHandler(async (request, response) => {
    //get userCollectionObject
    let userCollectionObject = request.app.get("userCollectionObject");
    //get user credentials obj from client
    let userCredObj = request.body;
    //seacrh for user by username
    let userOfDB = await userCollectionObject.findOne({
      username: userCredObj.username,
    });
    //if username not existed
    if (userOfDB == null) {
      response.send({ message: "Invalid user" });
    }
    //if username existed
    else {
      //compare passwords
      let status = await bcryptjs.compare(
        userCredObj.password,
        userOfDB.password
      );
      //if passwords not matched
      if (status == false) {
        response.send({ message: "Invalid password" });
      }
      //if passwords are matched
      else {
        //create token
        let token = jwt.sign(
          { username: userOfDB.username },
          'mySecretKey',
          { expiresIn: 10 }
        );
        //send token
        response.send({
          message: "success",
          payload: token,
          userObj: userOfDB,
        });
      }
    }
  })
);

//create a route to 'create-user'
userApp.post(
  "/create-user",
  expressAsyncHandler(async (request, response) => {
    //console.log(request.file.path);
    //get userCollectionObject
    let userCollectionObject = request.app.get("userCollectionObject");
    //get userObj as string from client and convert into object
    let newUserObj;
    console.log("Received userObj data:", request.body.userObj);
    try {
      // Try to parse the JSON string
      console.log("hi")
      newUserObj = JSON.parse(request.body.userObj);
    } catch (error) {
      // If parsing fails, handle the error (e.g., log it)
      console.error("Error parsing JSON:", error);
      response.status(400).send({ message: "Invalid JSON data" });
      return;
    }

    //seacrh for user by username
    let userOfDB = await userCollectionObject.findOne({
      username: newUserObj.username,
    });
    console.log(userOfDB)
    //if user existed
    if (userOfDB !== null) {
      response.send({
        message: "Username has already taken..Plz choose another",
      });
    }
    else {
      //hash password
      let hashedPassword = await bcryptjs.hash(newUserObj.password, 6);
      //replace plain password with hashed password in newUserObj
      newUserObj.password = hashedPassword;
      //insert newUser
      console.log(newUserObj)
      await userCollectionObject.insertOne(newUserObj);
      //send response
      response.send({ message: "New User created" });
    }
  })
);


//private route for testing
userApp.get('/test', verifyToken, (request, response) => {
  response.send({ message: "This reply is from private route" })
})
```

Explanation -

The above code is the routes for all the user functionalities like signup, login. When these routes are called, the application verifies the credentials/data received form the user in the database, if already present else it will add the data to the database. It will also verify the correct routes and encrypts the passwords and dashboard and other screens.

10) PostApi Routes -

```
postApp.post(
    "/new-post",
    expressAsyncHandler(async (request, response) => {
        //get postCollectionObject
        let postCollectionObject = request.app.get("postCollectionObject");

        //get newPostObj as string from client and convert into object
        let newPostObj;
        console.log("Received postObj data:", request.body.postObj);
        try {
            // Try to parse the JSON string
            newPostObj = JSON.parse(request.body.postObj);
        } catch (error) {
            // If parsing fails, handle the error (e.g., log it)
            console.error("Error parsing JSON:", error);
            response.status(400).send({ message: "Invalid JSON data" });
            return;
        }

        // Insert the new post object into the database
        await postCollectionObject.insertOne(newPostObj);
        response.send({ message: "New Post created" });
    })
);

const ITEMS_PER_PAGE = 10;

postApp.get(
    "/posts",
    expressAsyncHandler(async (request, response) => {
        let postCollectionObject = request.app.get("postCollectionObject");
        // Parse query parameters
        const page = parseInt(request.query.page) || 1; // Current page, default to 1
        const visibility = request.query.visibility || 'public'; // Post visibility, default to public

        // Calculate skip value for pagination
        const skip = (page - 1) * ITEMS_PER_PAGE;

        // Query to retrieve posts based on visibility and pagination
        let query = { visibility };

        let totalPosts = await postCollectionObject.countDocuments(query);
        let totalPages = Math.ceil(totalPosts / ITEMS_PER_PAGE);

        // Fetch posts for the current page
        let posts = await postCollectionObject.find(query)
            .sort({ createdAt: -1 }) // Sort by creation date, latest first
            .skip(skip) // Skip posts for pagination
            .limit(ITEMS_PER_PAGE) // Limit number of posts per page
            .toArray();

        response.send({ message: "Posts list", payload: { posts, totalPages } });
    })
);


//private route for testing
postApp.get('/test', verifyToken, (request, response) => {
  response.send({ message: "This reply is from private route" })
})
```

Explanation -

The above code helps to add the new post in the database based on the correct data and also gives all the posts that are requested by the users. If the posts are many then to solve this problem, the pagination is added so that 10 posts are shown for each page and the pages can be increased when the posts are increased, this can be find in the getposts route.

11) Verify Token Middleware-

```
const jwt = require("jsonwebtoken");
require("dotenv").config();

//write a middleware to verify token
```

```
const verifyToken = (request, response, next) => {
  //get bearer token
  let bearerToken = request.headers.authorization;
  //check token is existed
  if (bearerToken == undefined) {
   return response.send({ message: "Unauthorized request" });
  }
  //extract token
  let token = bearerToken.split(" ")[1];


  //if token is null
  if (token === null) {

   return response.send({ message: "Unauthorized request" });
  }
  try {
    //verify token
    jwt.verify(token, 'mySecretKey');
    //forwarfd req to private route
    next();
  } catch (err) {


    return response.send({ message: "Session expired..Relogin to continue" });
  }
};

//expor
module.exports = verifyToken;
```

Explanation -

This Verify Token middleware is used to verify the urls that can be authorized only when there is access to the url, therefore only when the used is authorised. For example, the /userdashboard is visible only if the user is authorized and logins to the user.