PROJECT DOCUMENTATION

Twitter Clone

Course Name: Advanced Web Development

Group Members:
- Khanh Nguyen Tran
- Duc Tin Nguyen
- Alago Victor
- Esho Caleb

Submission Date: 18/02/2024

Introduction

Purpose

Our Twitter Clone application was designed as part of our Advance Web Development final project. The project aims to replicate the core functionalities of Twitter. Through this project, we aim to demonstrate our understanding of CSS, ReactJS, MERN stack, and NoSQL database.

Key Features

- User Authentication
- MERN Stack Integration
- Styling with TailwindCSS
- ReactJS focuses on front-end development
- NodeJS for server-side logic
- Search functionality
- NoSQL Database (MongoDB), Firebase

- State management using Redux

Getting Started

Prerequisites

- Node.js
- MongoDB Atlas
- Firebase
- Redux
- Tailwind CSS
- nodemon
- Express.js
- ReactJS

Setting Up Prerequisites

Node.js

- Download and install node.js from https://nodejs.org/en

MongoDB

1. Sign up or sign in at https://www.mongodb.com/
2. Create a cluster after successfully signing in or signing up
3. Configure database and network access
4. Connect to the cluster with the node.js driver then save the connection string

Firebase

1. Create a Firebase project
2. Register your app
3. Save the Firebase SDK
4. Enable authentication

Running The Application

1. Clone the repository

```
git clone <repository-url>
```

2. Install Dependencies

```
#Dependencies for server
cd ../Twitter_Clone/server
npm install

#Dependencies for client
cd ../Twitter_Clone/client
npm install
```

3. Environment Configuration

Create a ".env" file inside the server

```
cd ../Twitter_Clone/server

#Create a .env file
type nul > .env
```

Add the following content inside the ".env" file

```
MONGO=mongodb+srv://<username>:<password>@twitter-cluster.vcly
jdt.mongodb.net/?retryWrites=true&w=majority
JWT=your_password
```

4. Firebase Configuration

Create a "firebase.js" inside Twitter_Clone/client. Add the following content to the file:

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-librar
ies
// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_AUTH_DOMAIN",
  projectId: "YOUR_PROJECT_ID",
  storageBucket: "YOUR_STORAGE_BUCKET",
  messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
  appId: "YOUR_APP_ID"
};
// Initialize Firebase
const app = initializeApp(firebaseConfig);
export default app;
```

5. **Start the server**

```
cd ../Twitter_Clone/server


#Start the back-end server
npm start
```

6. **Start the client**

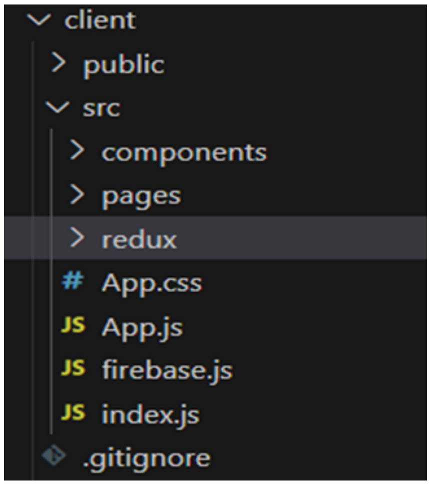
```
cd ../Twitter_Clone/client


#Start the front-end
npm start
```

Developer Documentation

Project Architecture

Front-end

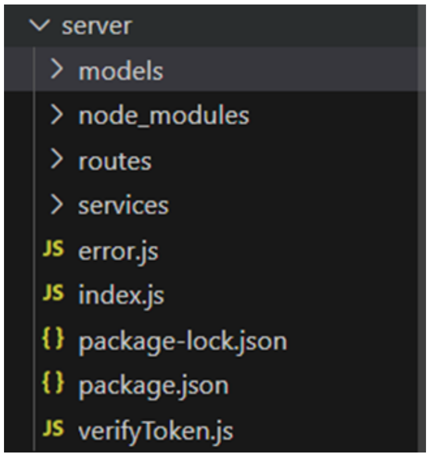


```
client
  public
  src
    components
    pages
    redux
    App.css
    App.js
    firebase.js
    index.js
  .gitignore
```

- **components: contains all the components required to build every single page.**
- **pages: contain the component that represents the entire page of our application.**
- **redux: contains the implementation of Redux for state management.**
- **firebase.js: Configure and initialization of the Firebase SDK for my application.**

Back-end

This directory is responsible for the backend logic, API endpoints, database models, and other server-side functionalities.



```
server
  models
  node_modules
  routes
  services
  error.js
  index.js
  package-lock.json
  package.json
  verifyToken.js
```

- **models:** contains Mongoose models or schema definitions for MongoDB database. Each file in this directory would define the structure of the data such as users, and tweets.

- **routes:** inside this directory, we will have different JavaScript files, each defining the API routes for our application. Routes can include user authentication, tweet management, profile updates, etc. This directory helps in organizing our endpoints into logical groups.

- **services:** this directory is typically used to organize and encapsulate business logic, database interactions, and third-party service integrations. By keeping this logic in separate service files, our route handlers can remain clean and focused on handling requests and responses.
- **error.js:** This is where we define how to respond to various errors that can occur during API operations, such as database errors, authentication errors, or other runtime exceptions.
- **verifyToken.js:** this script is likely a middleware for authenticating tokens sent with client requests. It would verify that the token is valid and allow the request to proceed if it is. This is an essential part of secure authentication, ensuring that protected routes are only accessible to authenticated users.

Technology Stack Justification

Our choices of technologies were driven by scalability, performance, and ease of development considerations, with an emphasis on creating a robust, modern, and user-friendly platform.

- MongoDB:

In the development of EduTweet, the choice of database technology was the most important. We chose MongoDB, a NoSQL document database, over traditional SQL databases due to several key factors that align closely with the needs of our application:

MongoDB's dynamic schema design allows us to develop without the constraints of a fixed table structure. This is particularly advantageous in the early stages of a project when the data model is evolving

MongoDB's query language is powerful and flexible, allowing for complex queries and aggregations.

The combination of flexible schemas and straightforward data manipulation significantly accelerates development

- Redux:

Redux provides a predictable state container, making state changes traceable and manageable across our React components.
It enhances the maintainability of our application by centralizing the application state, thereby facilitating easier bug tracking and testing.

- Firebase:

The real-time database and Firestore provide synchronous data updates across user devices, enabling a dynamic and responsive user experience.

- Tailwind CSS:

By using utility classes, we often avoid the complexity of custom CSS, which can become difficult to maintain. Tailwind helps to keep our styles consistent across our application.

Tailwind's purge feature removes unused CSS, resulting in smaller CSS file sizes in production. This leads to faster load times and improved performance.

The utility classes can scale well for large projects, especially when combined with component-based frameworks like React.

Database Schema

Users Collection Example

```
// Example
{
  "_id": "65d0feb8e0f5963d7fac2220",
  "firstname": "tran",
  "lastname": "nguyen",
  "username": "trannn",
  "email": "nguyen@gmail.com",
  "password": "U2FsdGVkX1+pVCv0GdyjDmC4FA27cFKf2/6A8Emfx/k=",
  "followers": [],
  "following": ["Array of user ObjectIds"],
  "createdAt": "2024-02-17T18:45:12.491+00
}
```

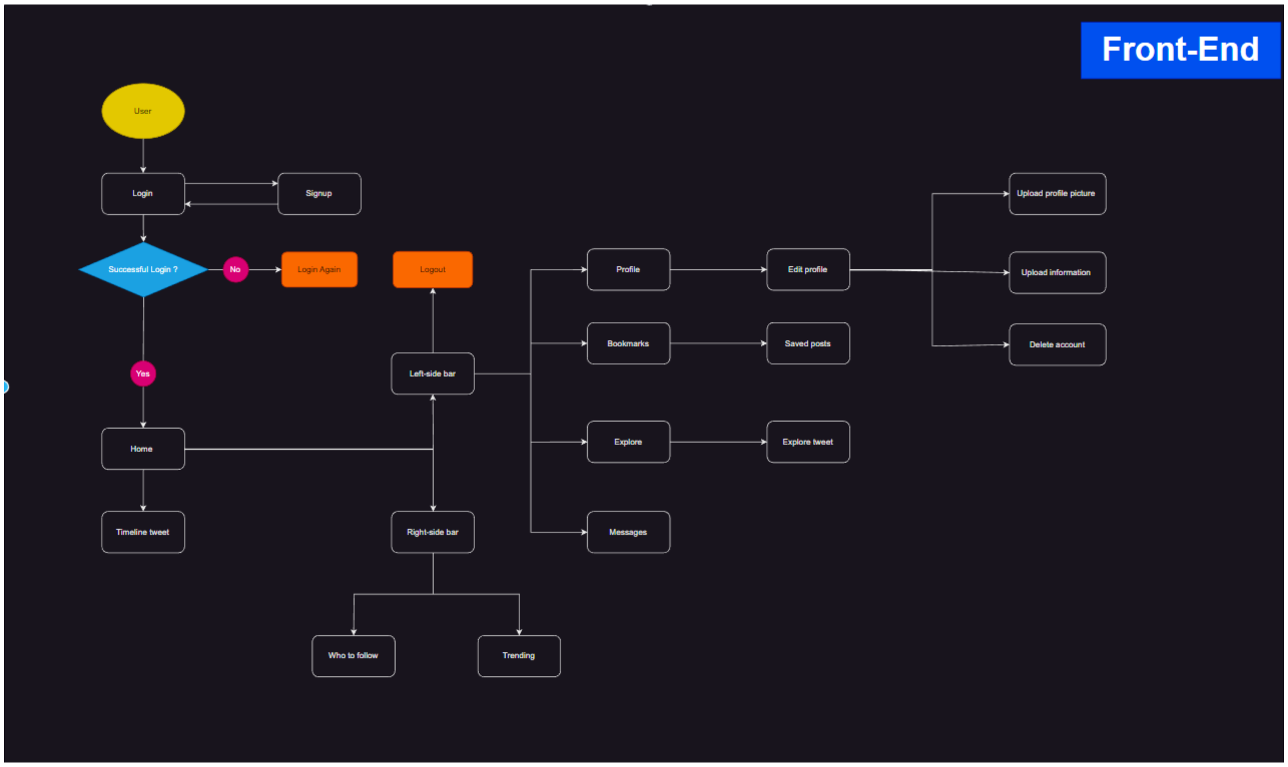
- Advantage:

MongoDB allows for flexible schema design, which is evident in the user model. As new features are added to the platform, the user schema can evolve without the need to alter the entire database structure. For example when the user signup, the schema at the moment is just the firstname, lastname, username,... Later on, when the user upload profile picture, a field to store is automatically updated.

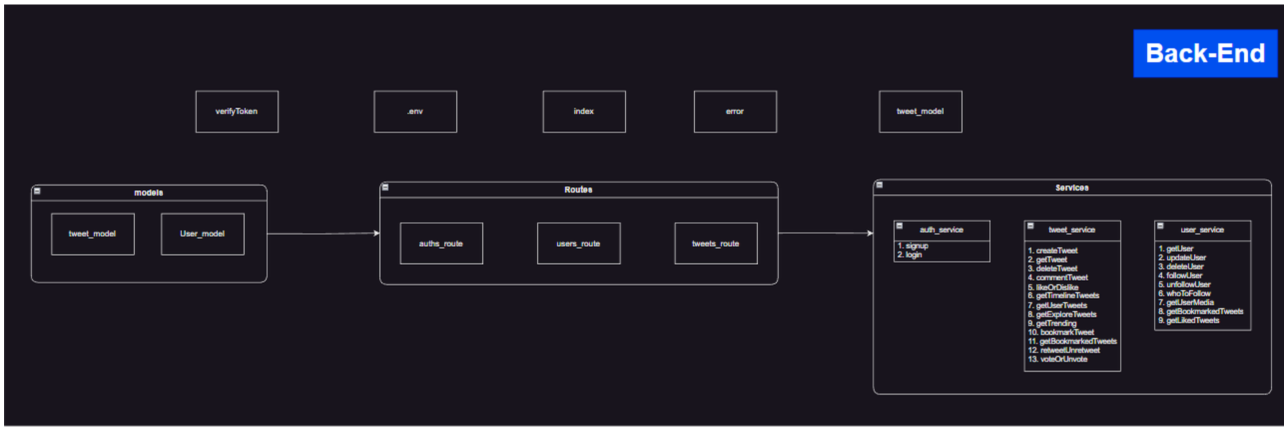
Tweets Collection Example

```
//Example
{
  "_id": "65d1e5c5b1214a485d4d430c",
  "username": "trannn",
  "content": "Vlog",
  "media": "https://firebasestorage.googleapis.com/v0/b/twitte
r-clone-7727a.appspot.com/o/path_to_media",
  "likes": [],
  "retweets": [],
  "comments": [],
  "bookmarks": ["Array of user ObjectIds"],
  "createdAt": "2024-02-18T11:11:01.216+00:00",
  "updatedAt": "2024-02-18T21:39:47.039+00:00",
  "__v": 0
}
```

Application Routing



Routing Distribution of Front-End



Routing Distribution of Back-End

Contribution

Each of our team members contributed to different parts of the whole project, from the back-end server to the front-end development.

Khanh Nguyen TRAN

- User Model
- User Services (Delete user, Follow user, Unfollow user)
- Tweet Services (Get trending tweets)
- Helper Function (Verify Token)
- Client Components (Navbar, Who to follow)
- Client Pages (Login, Tweet)
- Firebase Implementation

Duc Tin NGUYEN

- User Routes
- User Services (Get user, Update user)
- Tweet Services (Get all tweets)
- Authentication Services (Signup service)
- Client Components (Right sidebar, Timeline, Timeline tweets)
- Client Pages (Profile, Signup)
- Redux (User Slice)

Alago VICTOR

- Tweet Routes
- Tweet Services (Get, Create, Comment, Delete)
- Authentication Services (Login)
- Client Components (Explore tweets, Leftside bar)
- Client Pages (Errors, Messages)
- App.js
- Index.js

Esho CALEB

- Tweet Attributes
- Tweet Services (Like, Dislike, Get User Timeline Tweet, Retweet, Get User Tweet)
- Error Handling
- Client Components (Tweet and replies, tweet, edit profile)
- Client Pages (Explore, Home)
- Redux (User Store)