**FULL STACK 2 PROJECT**

**(2020-21)**

**SnapApp**

**A Social Media Platform.**

**FINAL PROJECT REPORT**



**Institute of Engineering & Technology**

**Submitted by-**

**Ayush Singhal**

**(181500181)**

**Aditya Pratap Singh**

**(181500045)**

***Supervised By: -***

**Mr. Pankaj Kapoor**

Technical Trainer

**Department of Computer Engineering & Applications**

# **Contents**

**Abstract 3**

1. **Introduction 3**
   1. General Introduction to the topic **3**
   2. Hardware and Software Requirements **21**
2. **Problem definition 23**
3. **Objectives 23**
4. **Implementation** **24**
5. **References 33**

**Abstract**

Through SnapApp, we will provide the customers an easy and convenient way to interact with people on internet through a less complex and simple Social Media Platform. Here, they can share their daily life activities with others and others will react to their activities.

**Introduction**

## **General Introduction to the topic**

To build a website we need the basic knowledge of React.js, Node.js, Express.js, RESTful API, HTTP requests, Axios, Redux and JWT.

## **About React.js: -**

React (also known as React.js or ReactJS) is an open-source, front end, JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies.

React can be used as a base in the development of single-page or mobile applications.

However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

React was created by Jordan Walke, a software engineer at Facebook, who released an early prototype of React called "FaxJS".

He was influenced by XHP, an HTML component library for PH. It was first deployed on Facebook's News Feed in 2011 and later on Instagram in 2012.[35] It was open-sourced at JSConf US in May 2013.

React Native, which enables native Android, iOS, and UWP development with React, was announced at Facebook's React Conf in February 2015 and open-sourced in March 2015.

On April 18, 2017, Facebook announced React Fiber, a new core algorithm of React library for building user interfaces.

React Fiber was to become the foundation of any future improvements and feature development of the React library.

On September 26, 2017, React 16.0 was released to the public.

**Common idioms**

React does not attempt to provide a complete "application library".

It is designed specifically for building user interfaces and therefore does not include many of the tools some developers might consider necessary to build an application.

This allows the choice of whichever libraries the developer prefers to accomplish tasks such as performing network access or local data storage.

Common patterns of usage have emerged as the library matures.

**Unidirectional data flow**

To support React's concept of unidirectional data flow (which might be contrasted with AngularJS's bidirectional flow), the Flux architecture was developed as an alternative to the popular model-view-controller architecture.

Flux features actions which are sent through a central dispatcher to a store, and changes to the store are propagated back to the view.

When used with React, this propagation is accomplished through component properties. Since its conception, Flux has been superseded by libraries such as Redux and MobX.

**Components**

React code is made of entities called components. Components can be rendered to a particular element in the DOM using the React DOM library.

When rendering a component, one can pass in values that are known as "props":

ReactDOM.render(<Greeter greeting="Hello World!" />, document.getElementById('myReactApp'));

The two primary ways of declaring components in React is via functional components and class-based components.

**Functional components**

Functional components are declared with a function that then returns some JSX.

const Greeting = (props) => <div>Hello, {props.name}!</div>;

**Class-based components**

Class-based components are declared using ES6 classes.

class ParentComponent extends React.Component {

state = {color: 'green’};

render () {

return (

<ChildComponent color={this.state.color} />

);

}

}

**Future development**

However, major changes to React go through the Future of React repository issues and pull requests.

This enables the React community to provide feedback on new potential features, experimental APIs and JavaScript syntax improvements.

**About Node.js** –

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

Node.js lets developers use JavaScript to write command line tools and for server-side scripting—running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser.

Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

Node.js was written initially by Ryan Dahl in 2009, about thirteen years after the introduction of the first server-side JavaScript environment, Netscape's LiveWire Pro Web. The initial release supported only Linux and Mac OS X.

Its development and maintenance were led by Dahl and later sponsored by Joyent.

**Overview**

Node.js allows the creation of Web servers and networking tools using JavaScript and a collection of "modules" that handle various core functionalities.

Modules are provided for file system I/O, networking (DNS, HTTP, TCP, TLS/SSL, or UDP), binary data (buffers), cryptography functions, data streams, and other core functions.

Node.js's modules use an API designed to reduce the complexity of writing server applications.

JavaScript is the only language that Node.js supports natively, but many compile-to-JS languages are available.

As a result, Node.js applications can be written in CoffeeScript, Dart, TypeScript, ClojureScript and others.

**Platform architecture**

Node.js brings event-driven programming to web servers, enabling development of fast web servers in JavaScript.

Developers can create scalable servers without using threading, by using a simplified model of event-driven programming that uses callbacks to signal the completion of a task.

Node.js connects the ease of a scripting language (JavaScript) with the power of Unix network programming.

**Technical details**

Node.js is a JavaScript runtime environment that processes incoming requests in a loop, called the event loop.

**Internals**

Node.js uses libuv underhood to handle asynchronous events. Libuv is an abstraction layer for network and file system functionality on both Windows and POSIX-based systems such as Linux, macOS, OSS on NonStop, and Unix.

**Threading**

Node.js operates on a single-thread event loop, using non-blocking I/O calls, allowing it to support tens of thousands of concurrent connections without incurring the cost of thread context switching.

The design of sharing a single thread among all the requests that use the observer pattern is intended for building highly concurrent applications, where any function performing I/O must use a callback.

To accommodate the single-threaded event loop, Node.js uses the libuv library—which, in turn, uses a fixed-sized thread pool that handles some of the non-blocking asynchronous I/O operations.

**About Express.js** –

Express.js, or simply Express, is a back-end web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs.

It has been called the de facto standard server framework for Node.js.

The original author, TJ Holowaychuk, described it as a Sinatra-inspired server, meaning that it is relatively minimal with many features available as plugins.

Express is the back-end component of popular development stacks like the MEAN, MERN or MEVN stack, together with the MongoDB database software and a JavaScript front-end framework or library.

**History**

Express.js was founded by TJ Holowaychuk. The first release, according to Express.js's GitHub repository, was on the 22nd of May, 2010. Version 0.12

In June 2014, rights to manage the project were acquired by StrongLoop.

StrongLoop was acquired by IBM in September 2015; in January 2016, IBM announced that it would place Express.js under the stewardship of the Node.js Foundation incubator.

**About RESTful API**–

REST is acronym for REpresentational State Transfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation.

Like any other architectural style, REST also does have its own 6 guiding constraints which must be satisfied if an interface needs to be referred as RESTful. These principles are listed below.

**Guiding Principles of REST**

**Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.

**Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.

**Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

**Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components.

REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state

**Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.

**Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

**Resource**

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g., a person), and so on.

REST uses a resource identifier to identify the particular resource involved in an interaction between components.

The state of the resource at any particular timestamp is known as resource representation. A representation consists of data, metadata describing the data and hypermedia links which can help the clients in transition to the next desired state.

The data format of a representation is known as a media type. The media type identifies a specification that defines how a representation is to be processed. A truly RESTful API looks like hypertext.

Every addressable unit of information carries an address, either explicitly (e.g., link and id attributes) or implicitly (e.g., derived from the media type definition and representation structure).

**Resource Methods**

Another important thing associated with REST is resource methods to be used to perform the desired transition. A large number of people wrongly relate resource methods to HTTP GET/PUT/POST/DELETE methods.

Roy Fielding has never mentioned any recommendation around which method to be used in which condition. All he emphasizes is that it should be uniform interface.

If you decide HTTP POST will be used for updating a resource – rather than most people recommend HTTP PUT – it’s alright and application interface will be RESTful.1.2.1

**About HTTP Requests**–

An HTTP client sends an HTTP request to a server in the form of a request message which includes following format:

**A Request-line**

Zero or more header (General|Request|Entity) fields followed by CRLF

An empty line (i.e., a line with nothing preceding the CRLF)

indicating the end of the header fields

Optionally a message-body

The following sections explain each of the entities used in an HTTP request message.

**Request-Line** - The Request-Line begins with a method token, followed by the Request-URI and the protocol version, and ending with CRLF. The elements are separated by space SP characters.

Request-Line = Method SP Request-URI SP HTTP-Version CRLF

Let's discuss each of the parts mentioned in the Request-Line.

**Request Method**

The request method indicates the method to be performed on the resource identified by the given Request-URI. The method is case-sensitive and should always be mentioned in uppercase. The following table lists all the supported methods in HTTP/1.1.

**1) GET**

The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

**2) HEAD**

Same as GET, but it transfers the status line and the header section only.

**3) POST**

A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

**4) PUT**

Replaces all the current representations of the target resource with the uploaded content.

**5) DELETE**

Removes all the current representations of the target resource given by URI.

**6) CONNECT**

Establishes a tunnel to the server identified by a given URI.

**7) OPTIONS**

Describe the communication options for the target resource.

**8) TRACE**

Performs a message loop back test along with the path to the target resource.

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped in five classes:

Informational responses (100–199)

Successful responses (200–299)

Redirects (300–399)

Client errors (400–499)

Server errors (500–599)

**About Axios**–

Axios is a Promise-based HTTP client for JavaScript which can be used in your front-end application and in your Node.js backend.

By using Axios it’s easy to send asynchronous HTTP request to REST endpoints and perform CRUD operations.

The Axios library can be used in your plain JavaScript application or can be used together with more advanced frameworks like Vue.js.

**Axios vs Fetch**

As you’re probably aware, modern browsers ship with the newer Fetch API built in, so why not just use that? There are several differences between the two that many feels give Axios the edge.

One such difference is in how the two libraries treat HTTP error codes.

When using Fetch, if the server returns a 4xx or 5xx series error, your catch () callback won’t be triggered and it is down to the developer to check the response status code to determine if the request was successful.

Axios, on the other hand, will reject the request promise if one of these status codes is returned.

Another small difference, which often trips up developers new to the API, is that Fetch doesn’t automatically send cookies back to the server when making a request. It’s necessary to explicitly pass an option for them to be included.

Axios has your back here.

**Making Requests -**

Similar to jQuery’s $.ajax function, you can make any kind of HTTP request by passing an options object to Axios:

axios ({

method: 'post',

url: '/login',

data: {

user: 'brunos',

lastName: 'ilovenodejs’}

});

Here, we’re telling Axios which HTTP method we’d like to use (e.g., GET/POST/DELETE etc.) and which URL the request should be made to.

We’re also providing some data to be sent along with the request in the form of a simple JavaScript object of key/value pairs.

By default, Axios will serialize this as JSON and send it as the request body.

**About JWT**–

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.

JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Here are some scenarios where JSON Web Tokens are useful:

**Authorization:** This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token.

Single Sign On is a feature that widely uses JWT nowadays, because of its small overhead and its ability to be easily used across different domains.

**Information Exchange**: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are.

Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

**@material-ui –**

Quickly build beautiful React apps. Material-UI is a simple and customizable component library to build faster, beautiful, and more accessible React applications. Follow your own design system, or start with Material Design.

**Installation –**

Material-UI is available as a npm package.

Stable channel v4 –

// with npm

npm install @material-ui/core

// with yarn

yarn add @material-ui/core

Alpha channel v5 –

// with npm

npm install @material-ui/core@next @emotion/react @emotion/styled

// with yarn

yarn add @material-ui/core@next @emotion/react @emotion/styled

**Who sponsors Material-UI?**

1. **Diamond :-** octopus, doit-intl. Diamond Sponsors are those who have pledged $1,500/month or more to Material-UI.
2. **Gold** :- Gold Sponsors are those who have pledged $500/month or more to Material-UI.
   1. **Via Patreon :** tidelift, bitsrc
   2. **Via OpenCollective :** call-em-all, canadacasino, hoodiebees
   3. **Direct :** Elevator

**Usage –**

Here is a quick example to get you started, it's all you need:

import \* as React from 'react';

import ReactDOM from 'react-dom';

import Button from '@material-ui/core/Button';

function App() {

return <Button variant="contained">Hello World</Button>;

}

ReactDOM.render(<App />, document.querySelector('#app'));

Yes, it's really all you need to get started as you can see in this live and interactive demo.

**1.2 Hardware Requirements**

**Hardware:**

1. Laptop or PC with the following configuration (minimum, for decent execution):
   * ***Processor***: Intel Core i3 6th Generation
   * ***Ram***: 4GB
   * ***OS***: Microsoft Windows 7 or above

**Software:**

1. ***GitHub***: For hosting the website (At the initial stage) and to store the website design and related stuff.
2. ***Front End Technologies:***
   1. HTML5
   2. CSS 3
   3. JavaScript using React.js
3. ***Back End Technologies:*** 
   1. MongoDB – document database
   2. Express.js – server-side web framework
   3. Node.js – premier JavaScript web server

**Testing Technologies to be used**

* **Hardware:**

1. Desktop
2. Laptop
3. Tablet
4. Cellphone

* **Software:**

1. ***Browsers (for compatibility check):***
   1. Google Chrome
   2. Mozilla Firefox
   3. Microsoft Internet Explorer
   4. Microsoft Edge
   5. Apple Safari
2. Search Engines

**Problem Statement**

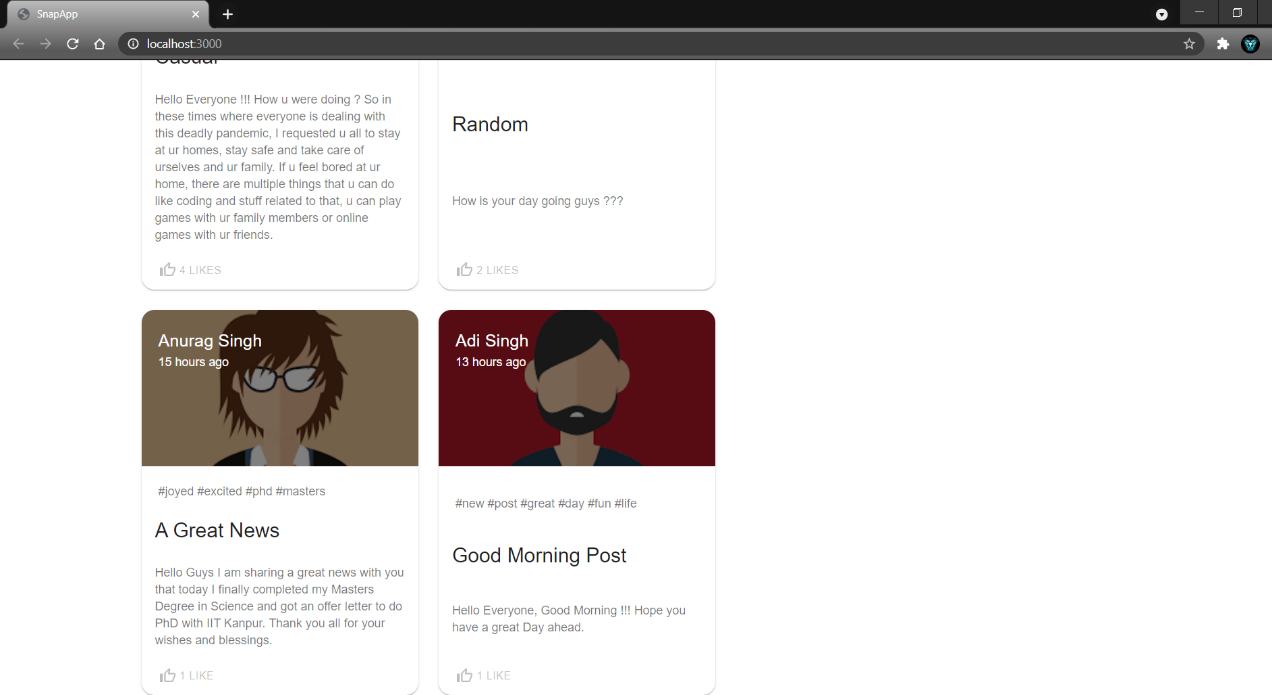
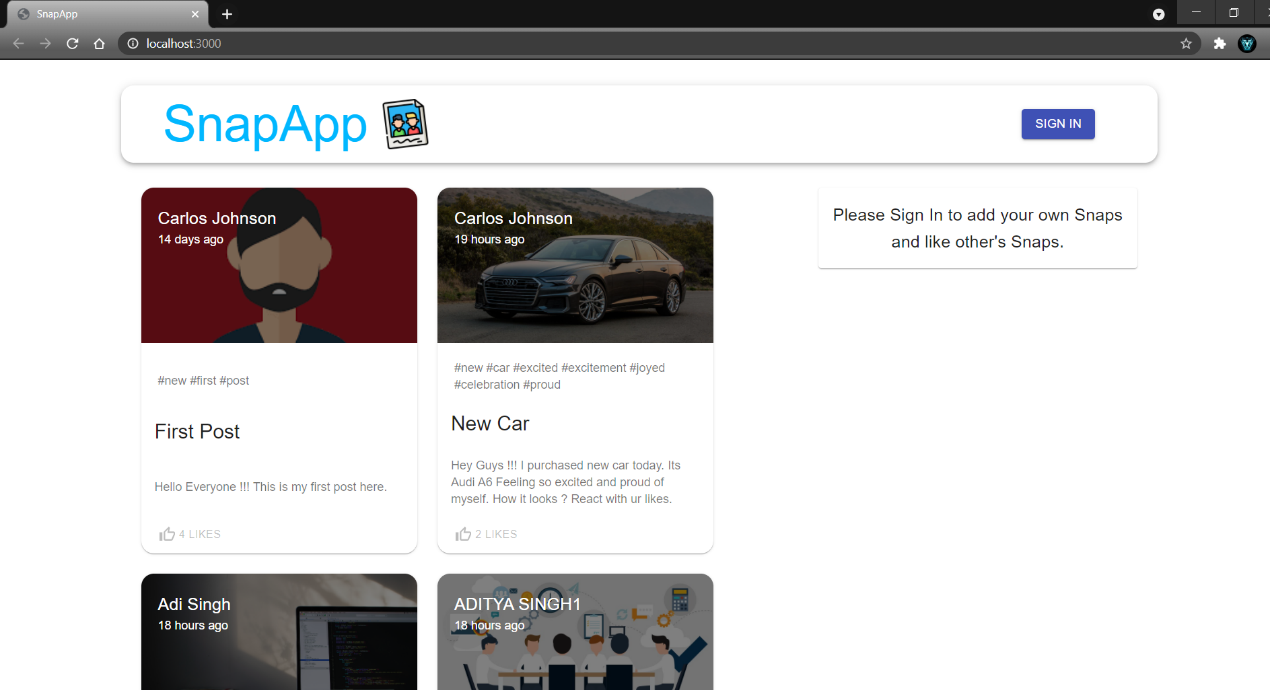
This project is about a social media platform where users can share their daily life activities with everyone there through images. User can add description about their posts. The other users can like their posts if they like it. The features we provided are Like, Update, Delete a post. Also, there are two options for user to signup: one is using Google Log In, other is using your own Email Id.

**Objectives**

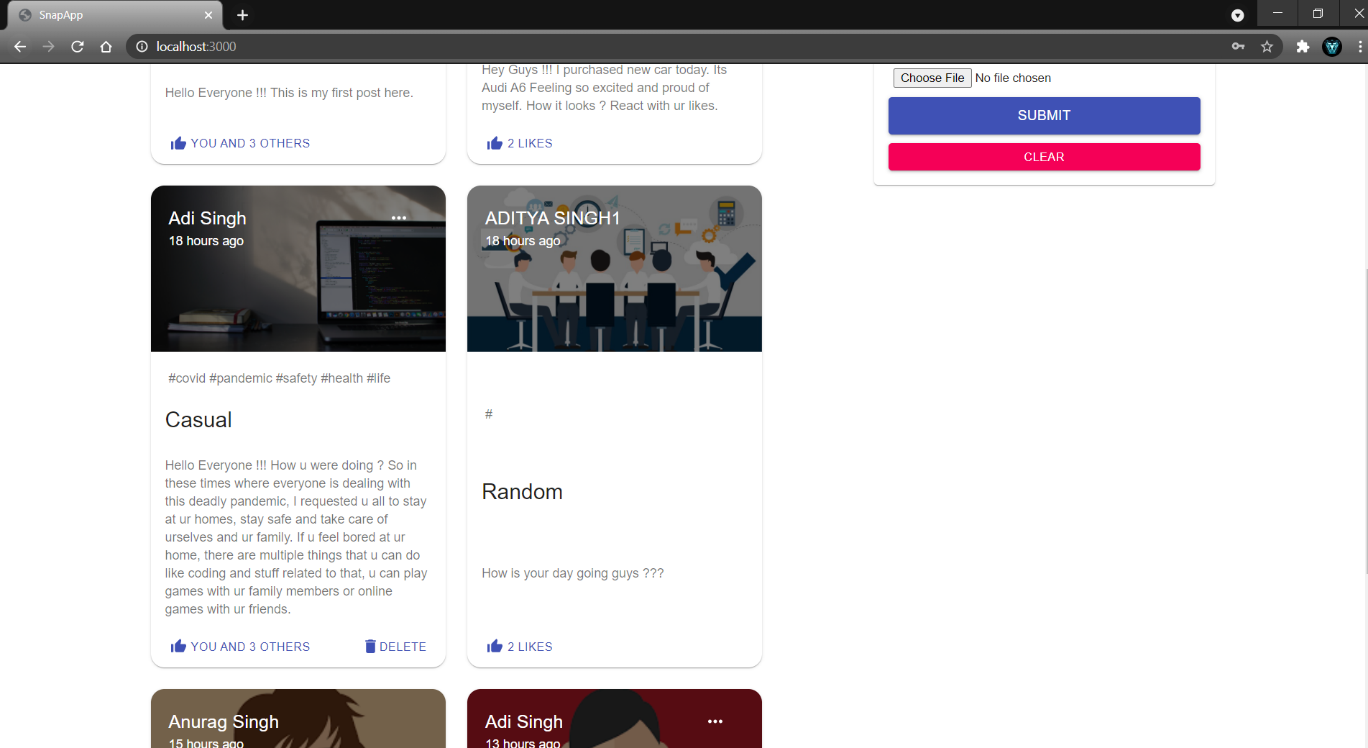
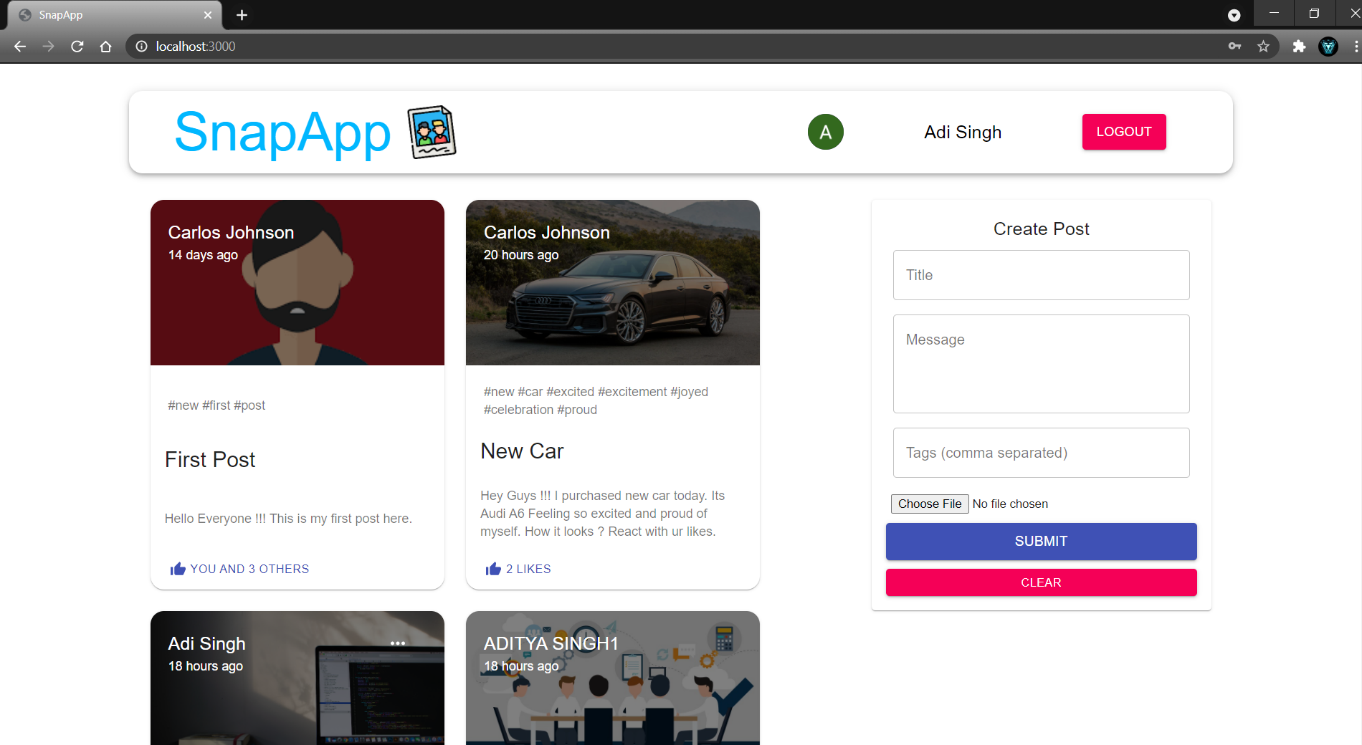
Through SnapApp, we want to give people a simple social media platform which is easily accessible by people of different age groups and where they can share their real-life activities and experiences through snaps and images to everyone there and they can used it as to store their memories for future. People on our platform can connect with each other and share their opinions on others post.

**Implementation**

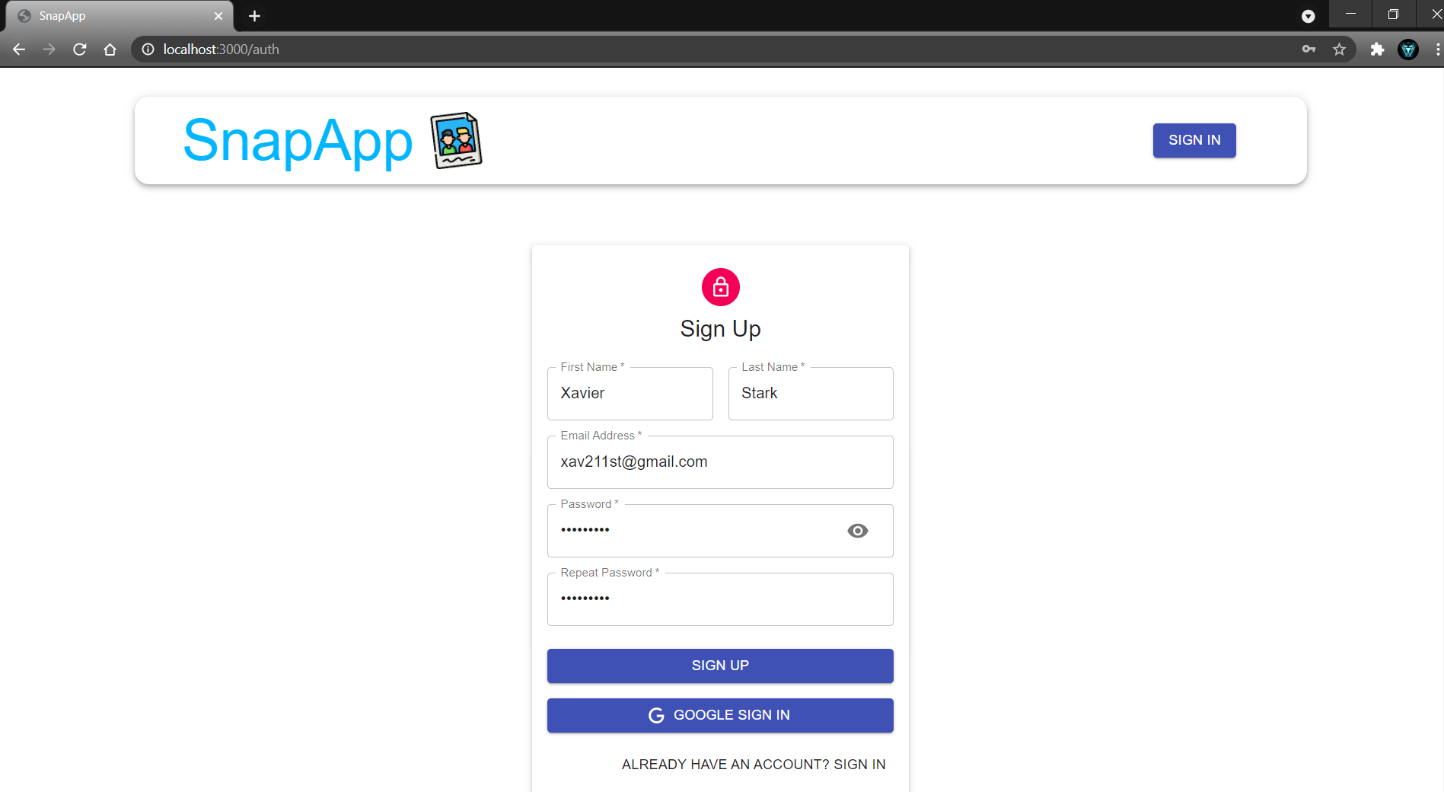
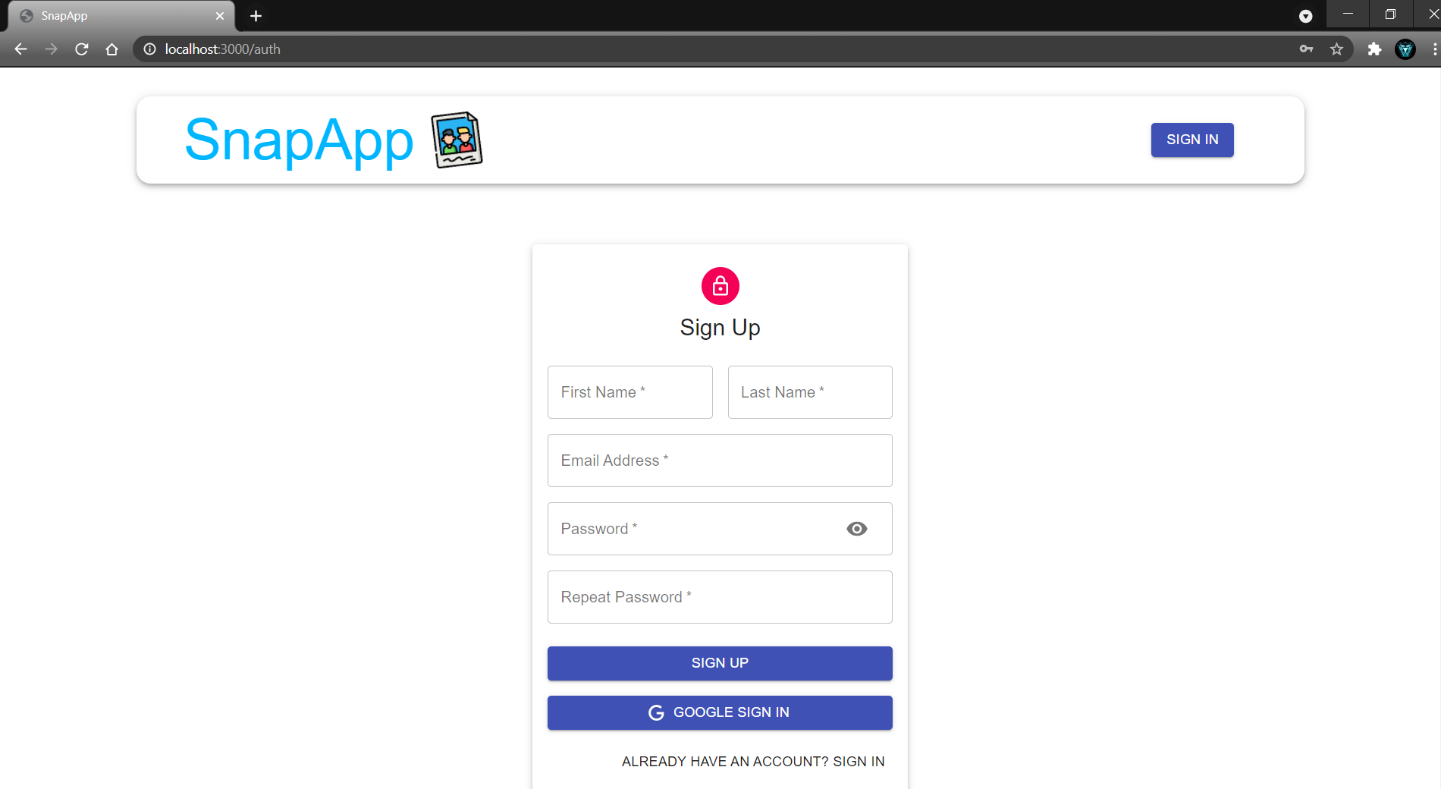
* + - 1. **Dashboard before login :-** This is the page that the user sees when they first visit the application. The user must login before posting something on the application.

****

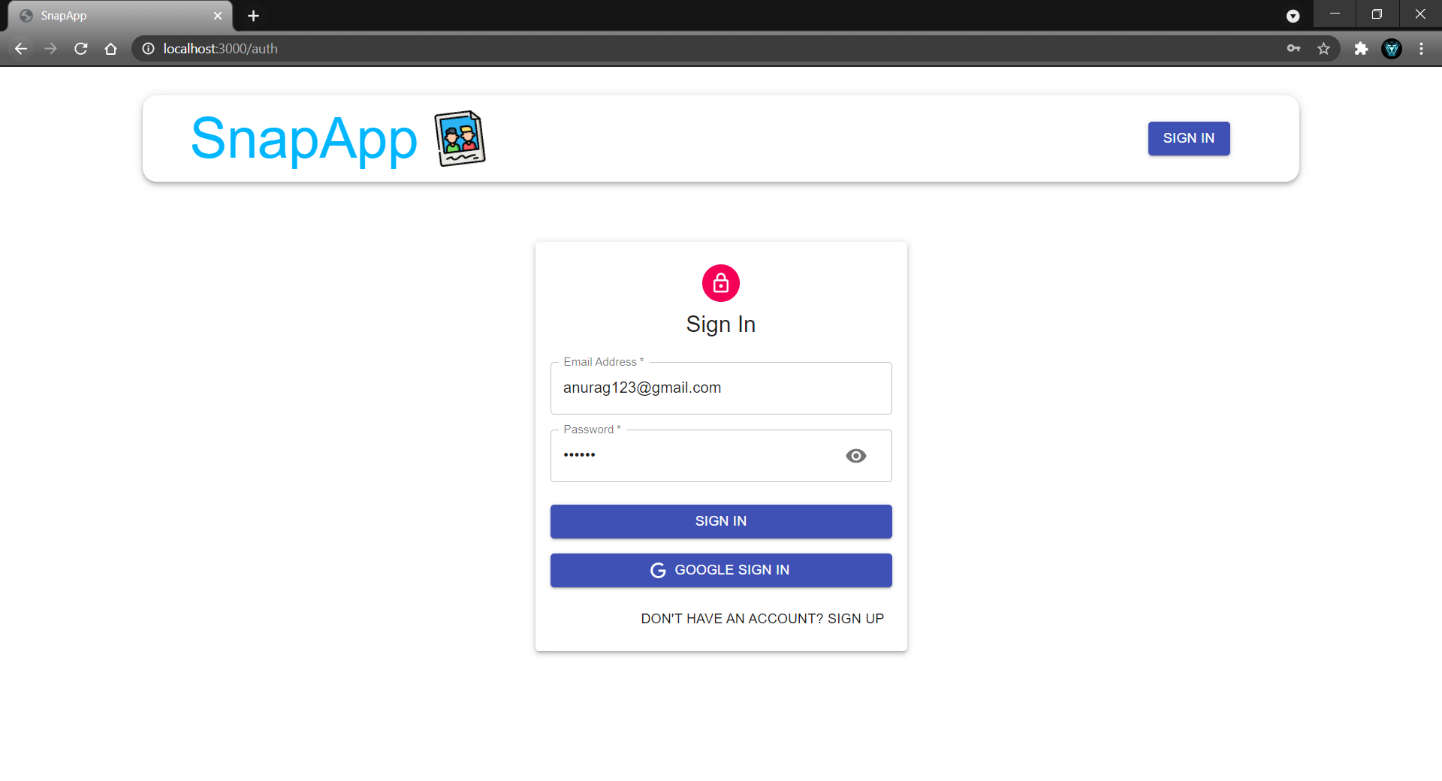
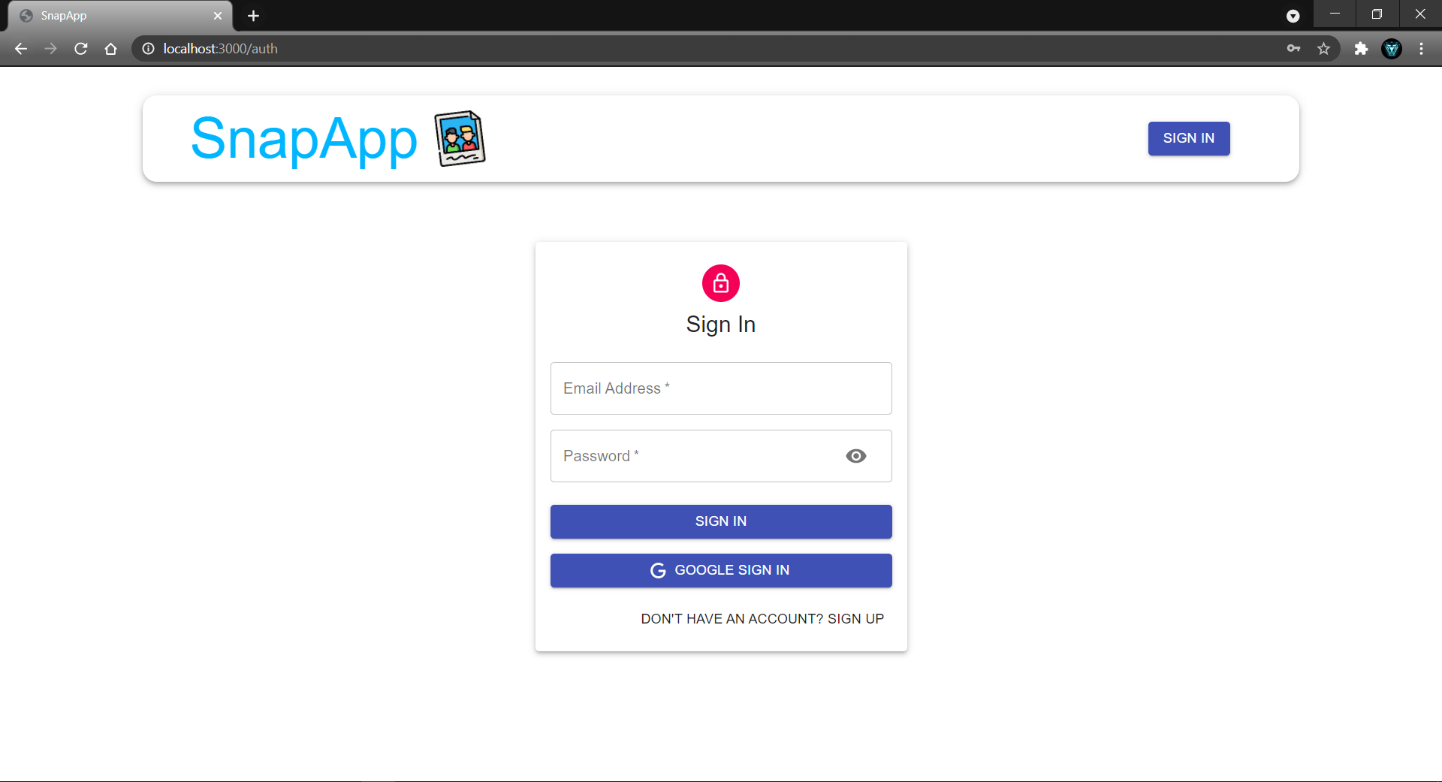
* + - 1. **Dashboard after login :-** This is the page after login / registering for the website as we can clearly see that in the place of login / register button there is a logout button.



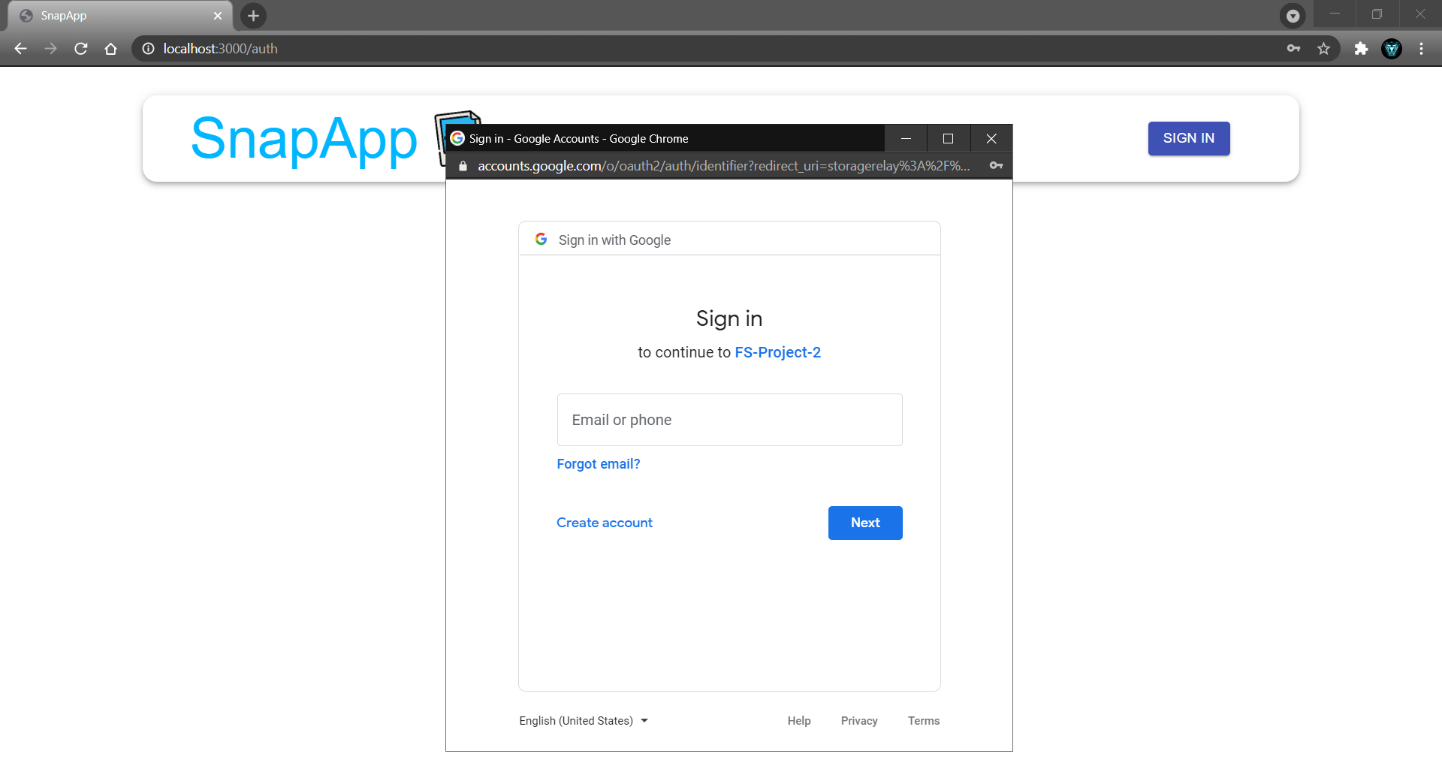
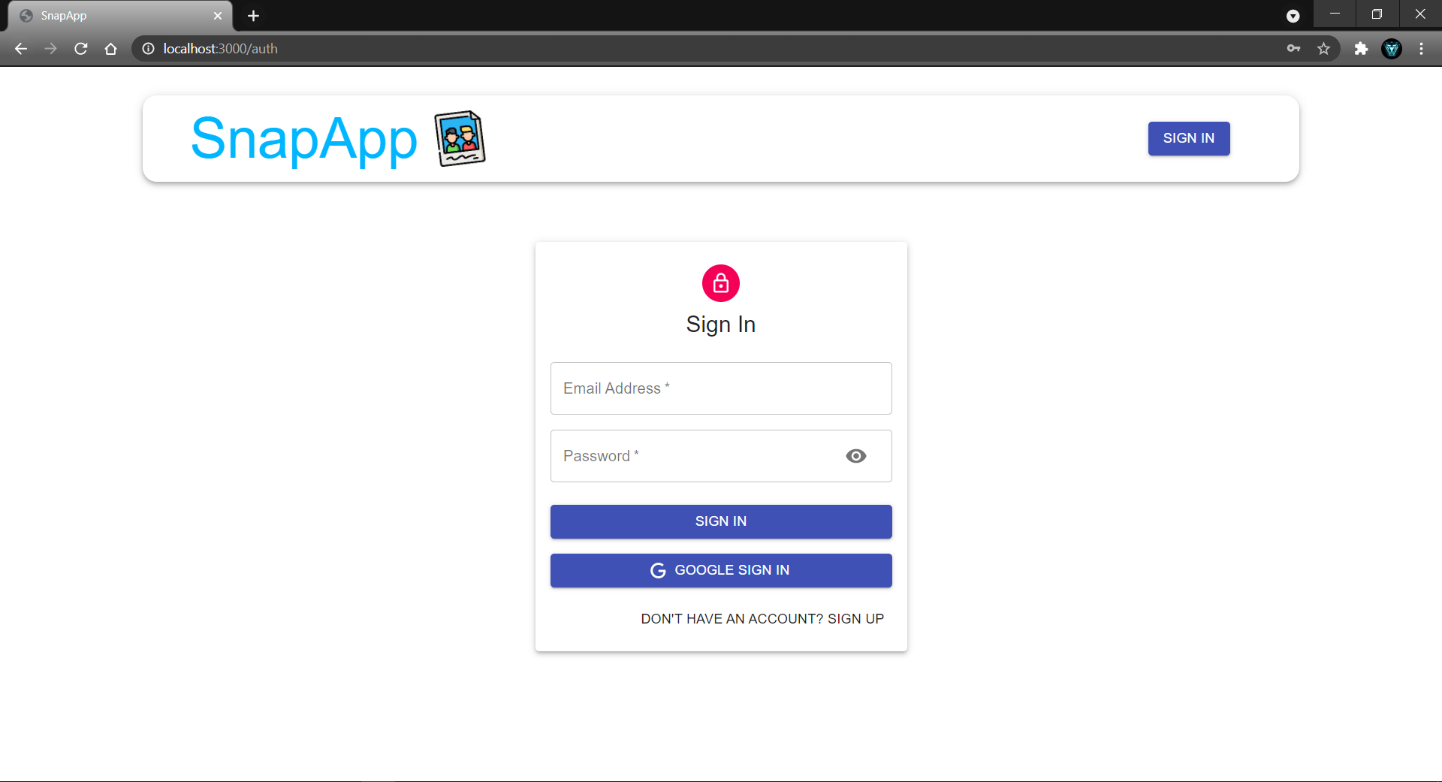
* + - 1. **Sign Up Page :-** This is the signup page for the application where new users will register their accounts in our application.

****

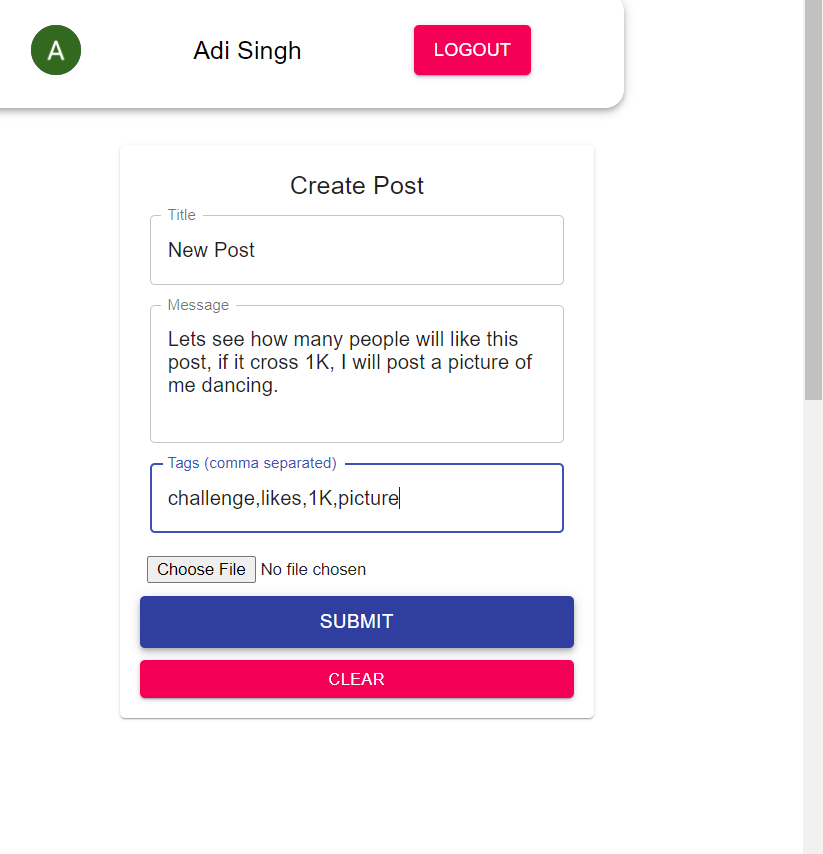
* + - 1. **Sign in Page :-** This is the signin page for the application where the register users after entering their login credentials can signin to our application and user can post there.

****

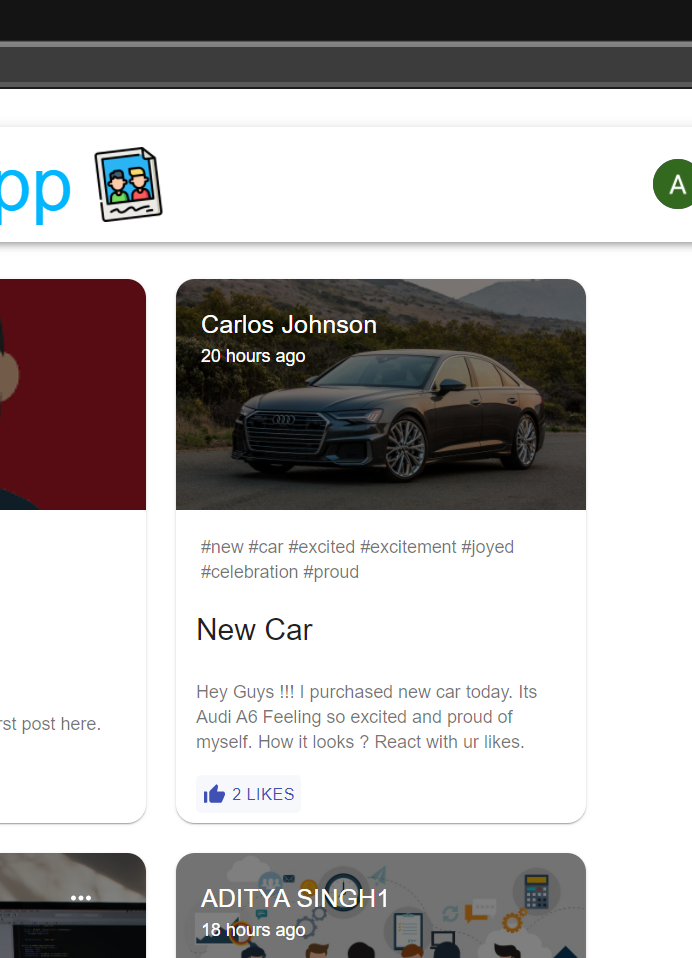
* + - 1. **Google Sign in :-** This is the alternative option for sign up or sign in into the application using Google login if user do not want to register on the platform.

****

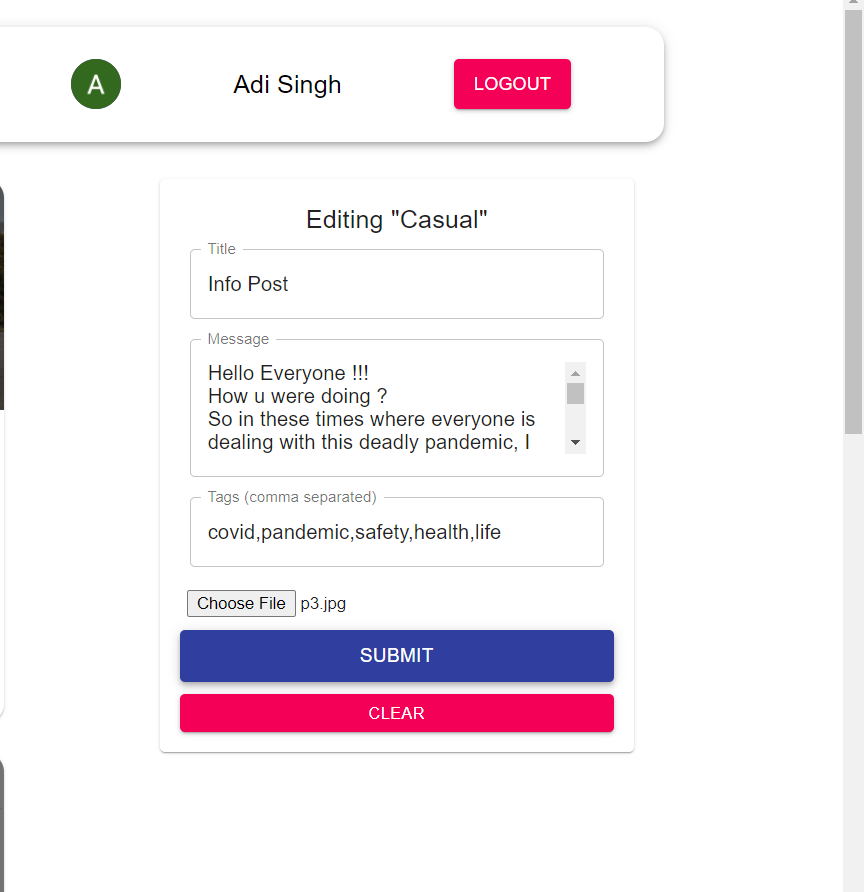
* + - 1. **Create a Post :-** We can create a post by filling out the following form and then submit it.



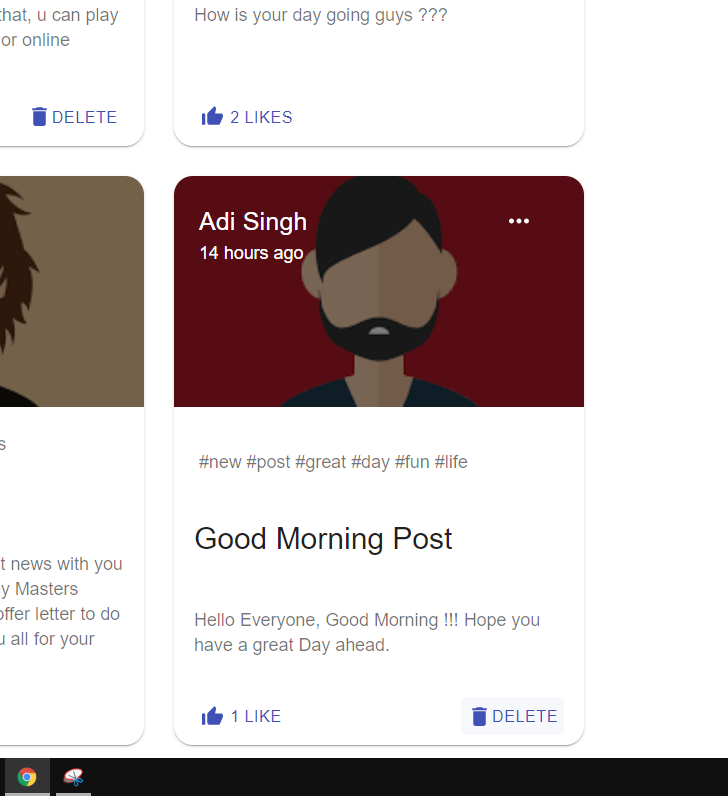
* + - 1. **Like a Post :-** The user can like a post using the Like button below every post. Using Like button the user can also like its own post with other post too. The user can remove like from the post whenever he / she wants.

****

* + - 1. **Update a Post :-** Using the update icon on their own post, user can edit their post. As we used authorization, user can only update their own post and can see others post but not update them.

****

* + - 1. **Delete a Post :-** Using the delete icon on the post, user can delete their post, the same authorization applied here too where a user cannot delete other users post.

****

**References**

* + <https://www.google.com>
  + <https://www.wikipedia.org>
  + <https://www.educba.com>
  + <https://www.skillcrush.com>
  + <https://www.geeksforgeeks.org>
  + <https://www.tutorialspoint.com>