**YS&OP Solutions**

This is Documentation of various Tasks performed by Viraj during his internship at YS&OP Solutions it contains GitHub repository of code , steps of execution and some more details about the tasks.

The Link for GitHub Repository : <https://github.com/Virajb21/YSOP_Internship.git>

1. **Tech Stack Comparison with Anaplan**

|  |  |  |
| --- | --- | --- |
| Tech Stack | Anaplan | Inventory App |
| **Language** | **Java** | **Java script** |
| **Front End Framework** | **Angular.js** | **React.js** |
| **Web designing Framework** | **Bootstrap** | **Bootstrap/Tailwind CSS** |
| **Server Side Scripting** | **Node.js** | **Node.js** |
| **Cloud Services** | **Amazon Web Services** | **Microsoft Azure** |
| **Database** | **MySQL, Cassandra** | **PostgreSQL/MySQL** |
| **Servers** | **Apache HTTP server , Rabbit MQ** | **Azure cloud services** |

1. **Login and Sign Up**

**Step 1: Backend Setup**

* Install Node.js and npm (Node Package Manager) on your machine.
* Set up a MySQL server and client.
* Create a new Node.js project using the command **npm init -y** in your desired project directory.
* Install the required dependencies using the command **npm install express mysql jsonwebtoken bcrypt**.
* Create a **server.js** file in the project's root directory to implement the backend API.

**Theory:**

* Node.js: A JavaScript runtime environment that allows running JavaScript code outside the browser.
* npm: The package manager for Node.js. It helps in installing and managing external libraries and dependencies.
* Express.js: A popular web framework for Node.js that simplifies building web applications.
* MySQL: An open-source relational database management system used to store data.
* JSON Web Tokens (JWT): A standard for securely transmitting information between parties as a JSON object. In this case, it will be used for user authentication.
* bcrypt: A library used to hash passwords securely.

**Step 2: Frontend Setup**

* Install create-react-app globally using the command **npm install -g create-react-app**.
* Create a new React.js project using the command **npx create-react-app frontend**.
* Navigate to the project directory using **cd frontend**.
* Install the required dependencies using the command **npm install axios react-router-dom tailwindcss**.

**Theory:**

* React.js: A JavaScript library for building user interfaces. It allows you to create reusable UI components.
* create-react-app: A tool that sets up a new React.js project with a basic file structure and configuration.
* Axios: A library used for making HTTP requests from the frontend to the backend.
* react-router-dom: A library for handling routing and navigation in React.js applications.
* Tailwind CSS: A utility-first CSS framework that provides a set of pre-built styles and classes to quickly build UI components.

**Step 3: Implement Backend API**

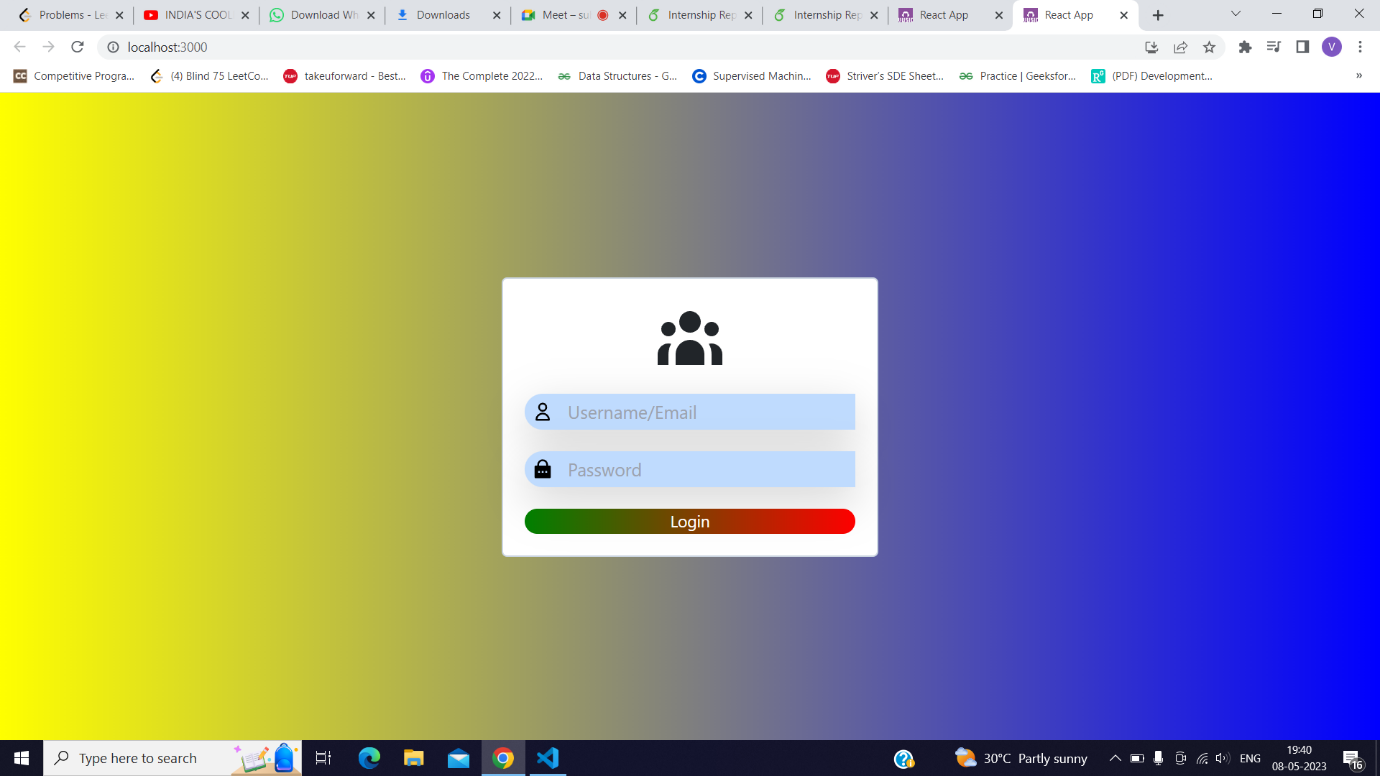
* Open the **server.js** file in your backend project directory.
* Configure the MySQL connection by providing the necessary credentials such as host, username, password, and database name.
* Set up the necessary routes for user sign up and login.
* Implement the sign-up route to hash the password, insert the user data into the MySQL database, and return an appropriate response.
* Implement the login route to query the database for the provided username, verify the password, generate a JWT token, and return it as a response.
* Start the server using the **app.listen()** method.

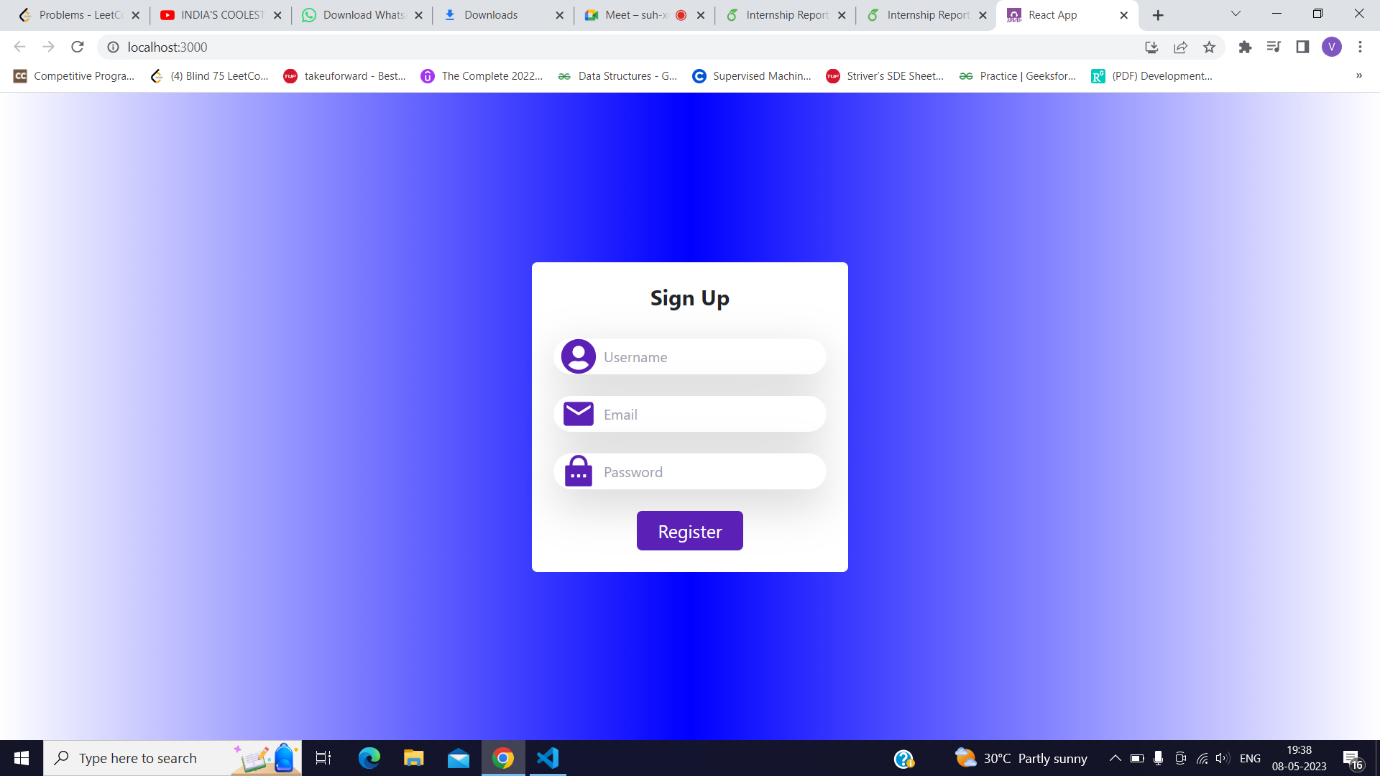
**Theory:**

* Backend API: A set of rules and protocols that allows communication between the frontend and backend systems.
* MySQL Connection: Establishing a connection between the Node.js server and the MySQL database to perform CRUD (Create, Read, Update, Delete) operations.
* Routes: Endpoints defined on the server to handle specific HTTP requests from the frontend.
* Hashing Passwords: Storing passwords securely by hashing them using algorithms like bcrypt.
* JWT Token: A digitally signed token that contains claims to securely transmit information between parties.

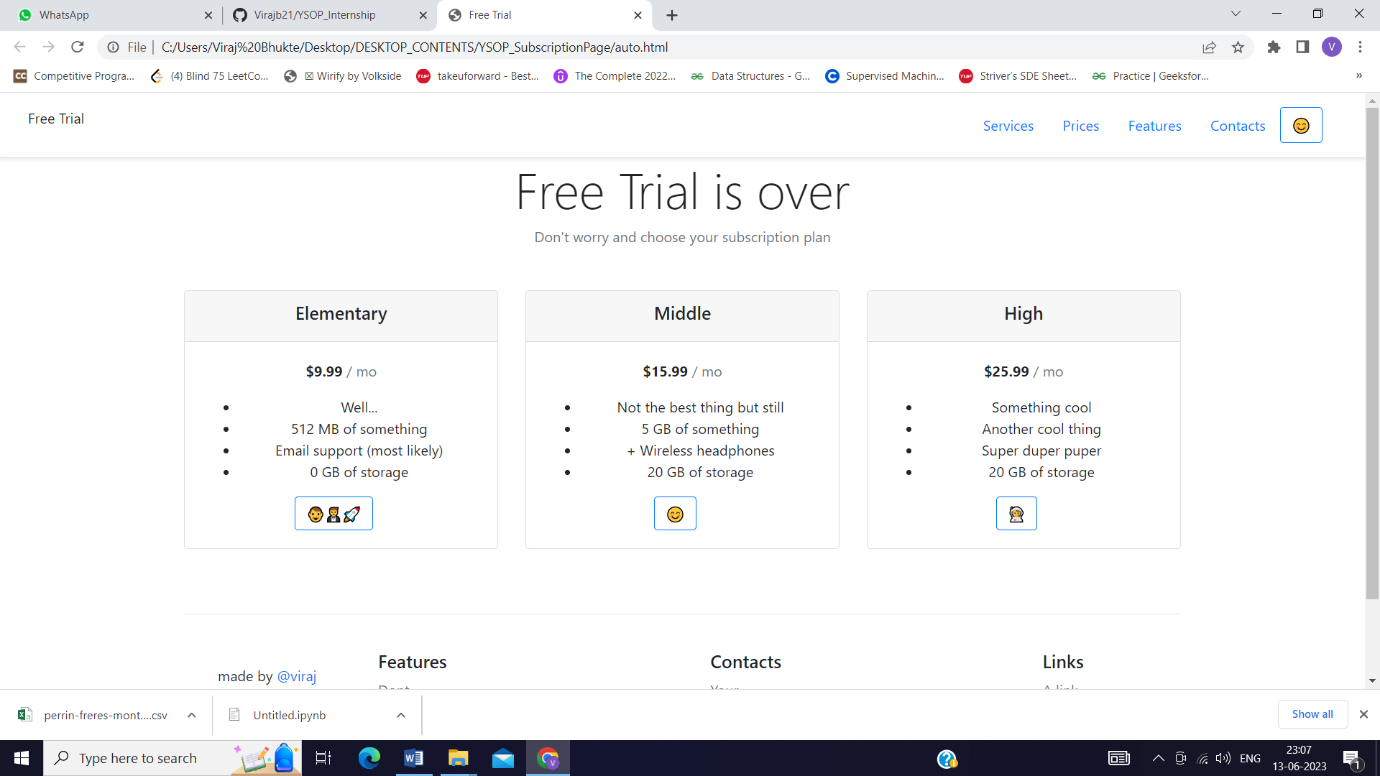
**Step 4: Design Frontend Components**

* Open the React.js project in your preferred code editor.
* Create the necessary components for the login and sign-up pages.
* Implement form fields to capture user input for username and password.
* Add event handlers to capture form submissions.
* Make HTTP requests to the backend API using Axios for user sign up and login.
* Implement client-side validation for the input fields if necessary.
* Use React Router to set up routes for the login and sign-up pages.





1. **Subscription Page :**
2. **Backend Setup**
   * Install Express and Stripe dependencies.
   * Create an Express app.
   * Implement a route to create a subscription.
   * Use Stripe API to create a customer with email and payment method.
   * Create a subscription for the customer with the desired price.
   * Return the subscription object as a response.
3. **Frontend Setup**
   * Install axios, react-stripe-js, and @stripe/react-stripe-js dependencies.
   * Create a React component for the subscription page.
   * Use Stripe Elements and React Stripe.js to collect payment details.
   * Make a POST request to the backend API to create a subscription.
   * Handle any errors and display success messages to the user.



1. **Machine Learning Model for forecasting –**

The ARIMA (Autoregressive Integrated Moving Average) model is a popular time series forecasting method that combines autoregressive (AR), differencing (I), and moving average (MA) components. It is widely used to analyze and forecast stationary time series data.

The theory behind the ARIMA model can be summarized as follows:

1. Autoregressive (AR) Component: The autoregressive part of the ARIMA model represents the relationship between an observation and a certain number of lagged observations (i.e., previous values). It assumes that the current value of a time series can be explained by linearly combining its past values. The order of the AR component is denoted by "p" and determines how many lagged values are considered.

2. Differencing (I) Component: The differencing component of the ARIMA model aims to make a non-stationary time series stationary by computing the differences between consecutive observations. Differencing helps remove trends or seasonality from the data, making it easier to model. The order of differencing is denoted by "d" and indicates the number of times differencing is performed.

3. Moving Average (MA) Component: The moving average part of the ARIMA model captures the dependency between the error terms (residuals) of the model. It assumes that the error at a given time point is a linear combination of the error terms at previous time points. The order of the MA component is denoted by "q" and determines the number of lagged residuals considered.

The general notation for an ARIMA model is ARIMA(p, d, q), where "p" represents the order of the autoregressive component, "d" represents the order of differencing, and "q" represents the order of the moving average component.

The ARIMA model is typically fitted to time series data by estimating the model parameters using methods like maximum likelihood estimation or least squares estimation. The estimated parameters are then used to forecast future values of the time series.

It's worth noting that the ARIMA model assumes that the underlying time series is stationary, meaning its statistical properties remain constant over time. If the time series exhibits non-stationarity, differencing is employed to transform it into a stationary series. Additionally, the identification of the appropriate values for "p", "d", and "q" often involves analyzing autocorrelation and partial autocorrelation plots, along with other statistical tests.

Overall, the ARIMA model provides a flexible framework for analyzing and forecasting time series data, making it a valuable tool in various fields such as finance, economics, and weather forecasting.

1. **Backend for Model –**

Machine Learning (ML) Backend with Flask and Forecast Pipeline

Flask is a lightweight web framework in Python that can be used to build a backend for machine learning applications. A forecast pipeline refers to the process of creating a system that takes input data, applies a trained ML model, and generates predictions or forecasts.

Here's a brief explanation of the theory behind building an ML backend using Flask and a forecast pipeline:

1. Machine Learning Model

- Develop or select an appropriate ML model for forecasting, such as regression, time series analysis (ARIMA, SARIMA), or machine learning algorithms (random forest, gradient boosting, etc.).

- Train the model using historical data, allowing it to learn patterns and make predictions based on the input features.

- Save the trained model to be used later in the backend.

2. Flask Backend

- Install Flask and set up a Flask project.

- Define API endpoints to receive input data and return predictions.

- Create routes and corresponding functions to handle incoming requests.

- Implement the forecast pipeline within the Flask routes to process the input data.

- Load the trained ML model within the Flask application.

- Preprocess the input data, if necessary, to ensure compatibility with the model.

- Pass the preprocessed input data to the ML model for prediction.

- Retrieve the predicted values from the model.

- Return the predictions as a response to the API request.

3. \*\*Deployment and Integration\*\*

- Deploy the Flask backend application to a web server or cloud platform.

- Test the API endpoints to ensure proper functionality.

- Integrate the frontend (e.g., web or mobile app) with the Flask backend API by sending HTTP requests to the appropriate endpoints.

- Handle errors and exceptions within the Flask backend for robustness.

- Continuously monitor and update the ML model as new data becomes available to maintain its accuracy and reliability.

Flask provides a flexible and scalable framework to build ML backends, enabling the deployment of predictive models and making predictions accessible through a well-defined API. By integrating a forecast pipeline within Flask, you can leverage the power of ML models to generate accurate forecasts based on incoming data.

