Guit hub link : https://github.com/Priyagowda22

Assignment 01

Web application mimicking google sheets:

This project will use:

- **Frontend**: React with Tailwind CSS
- **Backend**: Flask (Python)
- **Database**: SQLite (or any other relational DB)

Frontend (React)

We will create a simple spreadsheet interface with formula support.

Backend (Flask)

The backend will handle mathematical functions, data validation, and persistence.

React Frontend

Create a new React app and install required dependencies

1. Features of the Spreadsheet Interface

The spreadsheet should include:
Google Sheets UI: Toolbar, formula bar, and a grid-like cell structure.
Drag Functions: Users should be able to drag and fill cells with content or formulas.
Cell Dependencies: When a referenced cell updates, dependent cells should also update.
Basic Formatting: Support for bold, italics, font size, and color changes.
Row/Column Management: Users should be able to add, delete, and resize rows and columns.

The spreadsheet will be built using:

- Frontend: React (UI rendering and event handling)
- State Management: React Context / Redux
- Drag & Drop: React DnD or custom event handling
- Backend (optional for persistence): Flask / Node.js
- Database (optional for saving data): SQLite / Firebase

Code Implementation

Frontend (React)

Create a new React app and install dependencies:

```
npx create-react-app google-sheets-clone

cd google-sheets-clone

npm install react-table
```

Spreadsheet Component (Spreadsheet.js)

```
import React, { useState } from "react";

import "./Spreadsheet.css";


const Spreadsheet = () => {

   const rows = 10; // Define number of rows

   const cols = 5; // Define number of columns

   const [data, setData] = useState(Array(rows).fill().map(() => Array(cols).fill("")));


   const handleChange = (e, row, col) => {

      const newData = [...data];

      newData[row][col] = e.target.value;

      setData(newData);

   };

   return (

      <div className="spreadsheet">

         <table>

            <tbody>

               {data.map((row, rowIndex) => (

                  <tr key={rowIndex}>

                     {row.map((cell, colIndex) => (

                        <td key={colIndex}>

                           <input

                              type="text"

                              value={cell}

                              onChange={(e) => handleChange(e, rowIndex, colIndex)}
                           />

                        </td>

                     ))}

                  </tr>

               ))}

            </tbody>

         </table>

      </div>

   );

};


export default Spreadsheet;
```

CSS (spreadsheet.css)

```
.spreadsheet {
    width: 100%;
    margin: 20px auto;
}
table {
    border-collapse: collapse;
    width: 100%;
}
td {
    border: 1px solid #ccc;
    padding: 5px;
}
input {
    width: 100%;
    border: none;
    text-align: center;
}
```

App Component (App.js)

```
import React from "react";
import Spreadsheet from "./Spreadsheet";
function App() {
    return (
        <div className="App">
            <h2>Google Sheets Clone</h2>
```

```
          <Spreadsheet />

        </div>

  );

}

export default App;
```

Mathematical Functions for Spreadsheet

Function to Evaluate Formulas

```
function evaluateFormula(formula, data) {

    try {

        if (!formula.startsWith("=")) return formula; // Return if not a formula

        let expression = formula.slice(1).toUpperCase(); // Remove '='

        if (expression.includes("SUM")) {

            return sumFunction(expression, data);

        } else if (expression.includes("AVERAGE")) {

            return averageFunction(expression, data);

        } else if (expression.includes("MAX")) {

            return maxFunction(expression, data);

        } else if (expression.includes("MIN")) {

            return minFunction(expression, data);

        } else if (expression.includes("COUNT")) {

            return countFunction(expression, data);

        } else {

            return "ERROR"; // Invalid function
        }

    } catch (error) {

        return "ERROR";

    }

}
```

Mathematical Functions

1 SUM Function

```
function sumFunction(expression, data) {

    const range = extractRange(expression);

    const values = getValuesFromRange(range, data);

    return values.reduce((sum, val) => sum + (parseFloat(val) || 0), 0);

}
```

2 AVERAGE Function

```
function averageFunction(expression, data) {

    const range = extractRange(expression);

    const values = getValuesFromRange(range, data);

    const validNumbers = values.filter(val => !isNaN(parseFloat(val)));

    return validNumbers.length ? (validNumbers.reduce((sum, val) => sum +
parseFloat(val), 0) / validNumbers.length) : 0;

}
```

3 COUNT Function

```
function countFunction(expression, data) {

   const range = extractRange(expression);

   const values = getValuesFromRange(range, data);

   return values.filter(val => !isNaN(parseFloat(val))).length;

}
```

Complete program on Web Application Mimicking Google Sheets

```
// ---------------- FRONTEND (React + Next.js) ----------------

// pages/index.js - Main Spreadsheet UI

import React, { useState, useRef } from 'react';

import Handsontable from 'handsontable';

import 'handsontable/dist/handsontable.full.css';

import axios from 'axios';


const Home = () => {

   const hotRef = useRef(null);

   const [data, setData] = useState(Array(10).fill(Array(10).fill('')));


   const saveSpreadsheet = async () => {

      await axios.post('http://localhost:5000/save', { data });

   };


   return (

      <div>

         <h1>Google Sheets Clone</h1>

         <button onClick={saveSpreadsheet}>Save</button>
         <div ref={hotRef} />
      </div>
   );
};

export default Home;

// ---------------- BACKEND (Node.js + Express) ----------------
// server.js - Handles API for spreadsheet data
const express = require('express');
const cors = require('cors');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const app = express();

app.use(cors());
app.use(bodyParser.json());

mongoose.connect('mongodb://localhost:27017/spreadsheet', { useNewUrlParser: true,
useUnifiedTopology: true });
const SheetSchema = new mongoose.Schema({ data: Array });
const Sheet = mongoose.model('Sheet', SheetSchema);

app.post('/save', async (req, res) => {
   const sheet = new Sheet({ data: req.body.data });
   await sheet.save();
   res.send('Saved');
});

app.listen(5000, () => console.log('Server running on port 5000'));
```

Assignment 02

Building a Support Agent Chatbot for CDP

Data Sources:

- Segment Documentation: https://segment.com/docs/?ref=nav
- Particle Documentation: https://docs.mparticle.com/
- Lytics Documentation: https://docs.lytics.com/
- Zeotap Documentation: https://docs.zeotap.com/home/en-us/

Core Functionalities:

```
from flask import Flask, request, jsonify
from transformers import pipeline
import requests
from bs4 import BeautifulSoup
import re

app = Flask(__name__)

# Load a Question Answering Model (Can use OpenAI API instead)
qa_pipeline = pipeline("question-answering", model="deepset/roberta-base-squad2")

# CDP Documentation Sources
CDP_DOCS = {
    "segment": "https://segment.com/docs/",
    "mparticle": "https://docs.mparticle.com/",
    "lytics": "https://docs.lytics.com/",
    "zeotap": "https://docs.zeotap.com/home/en-us/"
}

def fetch_documentation(url):
    """Fetches and extracts text from a given documentation URL."""
    response = requests.get(url)
    if response.status_code != 200:
        return "Error fetching documentation"
    soup = BeautifulSoup(response.text, "html.parser")
    return ' '.join([p.text for p in soup.find_all('p')])

@app.route('/ask', methods=['POST'])
def answer_question():
    """Handles user questions and fetches answers from CDP docs."""
    data = request.json
    question = data.get("question", "")
    cdp = data.get("cdp", "")

    if cdp not in CDP_DOCS:
        return jsonify({"error": "Unsupported CDP. Choose from: Segment, mParticle, Lytics, Zeotap."})

    doc_text = fetch_documentation(CDP_DOCS[cdp])

    answer = qa_pipeline({"question": question, "context": doc_text})
    return jsonify({"question": question, "answer": answer['answer']})

@app.route('/compare', methods=['POST'])
def compare_cdps():
    """Handles cross-CDP comparisons."""
    data = request.json
    question = data.get("question", "")

    responses = {}
    for cdp, url in CDP_DOCS.items():
        doc_text = fetch_documentation(url)
        answer = qa_pipeline({"question": question, "context": doc_text})
        responses[cdp] = answer['answer']

    return jsonify({"question": question, "comparisons": responses})

@app.route('/validate', methods=['POST'])
def validate_question():
    """Ensures the question is relevant to CDPs."""
    data = request.json
    question = data.get("question", "")

    irrelevant_keywords = ["movie", "weather", "sports"]
    if any(word in question.lower() for word in irrelevant_keywords):
        return jsonify({"error": "This question is not related to Customer Data Platforms."})

    return jsonify({"status": "valid question"})

if __name__ == '__main__':
    app.run(debug=True)
```