Assignment 1: Web Application Mimicking Google Sheets

Objective

Develop a web application that replicates the core functionalities and user interface (UI) of Google Sheets, focusing on mathematical and data quality functions, data entry, and key UI interactions.

Features and Implementation

Below is a code implementation for the main features of the project.

1. Spreadsheet Interface

Code Implementation:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Google Sheets Mimic</title>
  <link rel="stylesheet" href="styles.css">
  <script src="app.js" defer></script>
</head>
<body>
  <div id="app">
    <header class="toolbar">
      <button onclick="formatText('bold')">Bold</button>
      <button onclick="formatText('italic')">Italic</button>
      <select id="font-size" onchange="changeFontSize()">
        <option value="12">12px</option>
        <option value="14">14px</option>
        <option value="16">16px</option>
      </select>
    </header>
    <div class="spreadsheet">
      <div id="formula-bar">
        <input type="text" id="formula-input" placeholder="Enter formula here...">
      </div>
      <div id="grid-container">
        <!-- Grid will be dynamically generated here -->
      </div>
    </div>
  </div>
```

```
</body>
</html>
```

JavaScript for Grid Creation and Basic Interactivity:

```
// app.js

const rows = 20;
const cols = 10;
const gridContainer = document.getElementById('grid-container');

function createGrid() {
  for (let i = 0; i < rows; i++) {
    const row = document.createElement('div');
    row.className = 'row';
    for (let j = 0; j < cols; j++) {
      const cell = document.createElement('div');
      cell.className = 'cell';
      cell.contentEditable = true;
      cell.dataset.row = i;
      cell.dataset.col = j;
      row.appendChild(cell);
    }
    gridContainer.appendChild(row);
  }
}

createGrid();

function formatText(command) {
  document.execCommand(command, false, null);
}

function changeFontSize() {
  const fontSize = document.getElementById('font-size').value;
  document.execCommand('fontSize', false, '7'); // Set default size level 7
  document.querySelectorAll('.cell').forEach(cell => {
    if (document.getSelection().containsNode(cell, true)) {
      cell.style.fontSize = `${fontSize}px`;
    }
  });
}
```

CSS for Styling:

```css
/* styles.css */

body {
   font-family: Arial, sans-serif;
   margin: 0;
   padding: 0;
}

.toolbar {
   display: flex;
   align-items: center;
   padding: 10px;
   background-color: #f1f1f1;
   border-bottom: 1px solid #ddd;
}

.spreadsheet {
   display: flex;
   flex-direction: column;
}

#formula-bar {
   padding: 10px;
   border-bottom: 1px solid #ddd;
}

#grid-container {
   display: grid;
   grid-template-columns: repeat(10, 1fr);
   gap: 1px;
   background-color: #ddd;
}

.row {
   display: flex;
}

.cell {
   width: 100px;
   height: 30px;
   background-color: white;
   border: 1px solid #ccc;
```

```
  padding: 5px;
}

.cell:focus {
  outline: 2px solid #4caf50;
}
```

2. Mathematical Functions

Code Implementation:

```
function evaluateFormula(input) {
  try {
    return eval(input); // For simplicity, use eval; sanitize for real-world usage
  } catch (error) {
    return "Error";
  }
}

const formulaInput = document.getElementById('formula-input');
formulaInput.addEventListener('change', () => {
  const formula = formulaInput.value;
  const result = evaluateFormula(formula);
  alert(`Result: ${result}`);
});
```

3. Data Quality Functions

Code Implementation:

```
function trimCell(cell) {
  cell.innerText = cell.innerText.trim();
}

function upperCaseCell(cell) {
  cell.innerText = cell.innerText.toUpperCase();
}

function lowerCaseCell(cell) {
  cell.innerText = cell.innerText.toLowerCase();
}

function removeDuplicates() {
  const uniqueRows = new Set();
  document.querySelectorAll('.row').forEach(row => {
    const rowData = Array.from(row.children).map(cell => cell.innerText).join(',');
    if (uniqueRows.has(rowData)) {
      row.remove();
    } else {
      uniqueRows.add(rowData);
    }
  });
}
```

Assignment 2: Support Agent Chatbot for CDP

Code Implementation Outline

The chatbot will use a basic NLP pipeline to extract information from documentation.

Backend Implementation (Python/Flask):

```
from flask import Flask, request, jsonify
from transformers import pipeline

app = Flask(__name__)

# Load a pretrained NLP model for question-answering
qa_pipeline = pipeline("question-answering")

@app.route('/ask', methods=['POST'])
def answer_question():
  data = request.json
  question = data['question']
  context = data['context']  # Extracted documentation content

  result = qa_pipeline({'question': question, 'context': context})
  return jsonify(result)

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

Frontend Integration:

```
async function askQuestion(question) {
  const response = await fetch('http://localhost:5000/ask', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ question, context: document.getElementById('context').value })
  });
  const result = await response.json();
  alert(`Answer: ${result.answer}`);
}

const askButton = document.getElementById('ask-button');
askButton.addEventListener('click', () => {
  const question = document.getElementById('question-input').value;
  askQuestion(question);
});
```