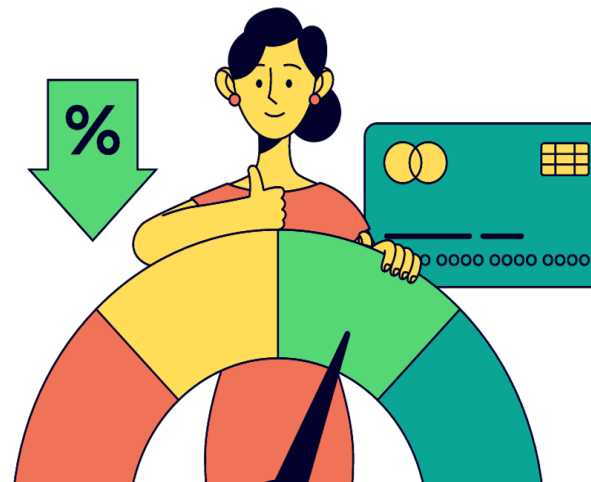


# Credit Scoring Dashboard



# **Mishra Mohammad**

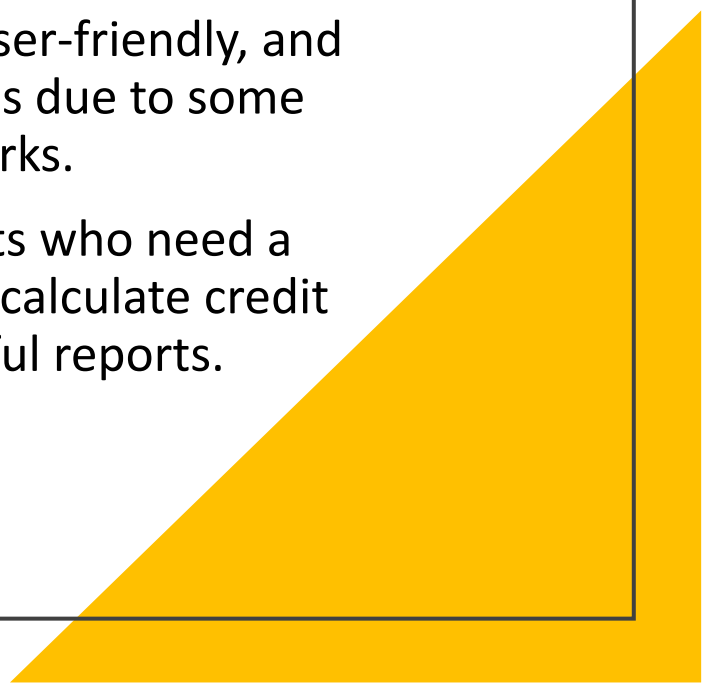
- Final year student – Computer and Information Sciences
- Aspire to create a meaningful impact in the world of tech and encourage females to not hesitate when choosing a field in STEM.
- Love all things pink and bling...and pink 😊

# Agenda



- Introduction
- Project Overview
- Architecture
- Frontend
- Backend
- Code Snippets
- Features
- Challenges
- Future Enhancements
- Q & As

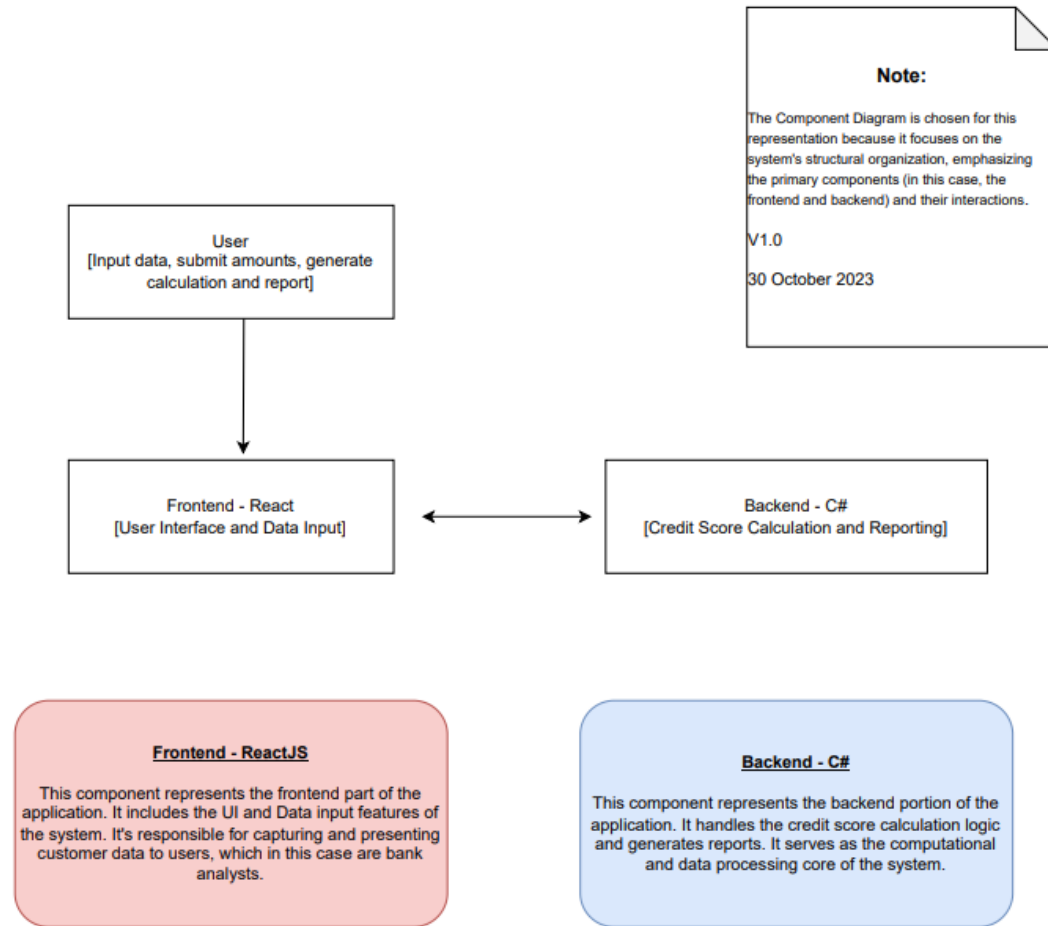
# Project Overview

- The purpose of the project is to develop a Credit Scoring Dashboard, facilitating bank analysts in efficiently evaluating customer creditworthiness.
  - I chose C# for the backend and ReactJS for the frontend to build a robust, user-friendly, and responsive application. This is due to some familiarity with the frameworks.
  - Target users are bank analysts who need a tool to input customer data, calculate credit scores, and generate insightful reports.
- 
- A large yellow triangle is positioned in the bottom right corner of the slide, pointing towards the top right.

# Architecture

- The purpose of the project is to develop a Credit Scoring Dashboard, facilitating bank analysts in efficiently evaluating customer creditworthiness.
- I chose C# for the backend and ReactJS for the frontend to build a robust, user-friendly, and responsive application. This is due to some familiarity with the frameworks.
- Target users are bank analysts who need a tool to input customer data, calculate credit scores, and generate insightful reports.

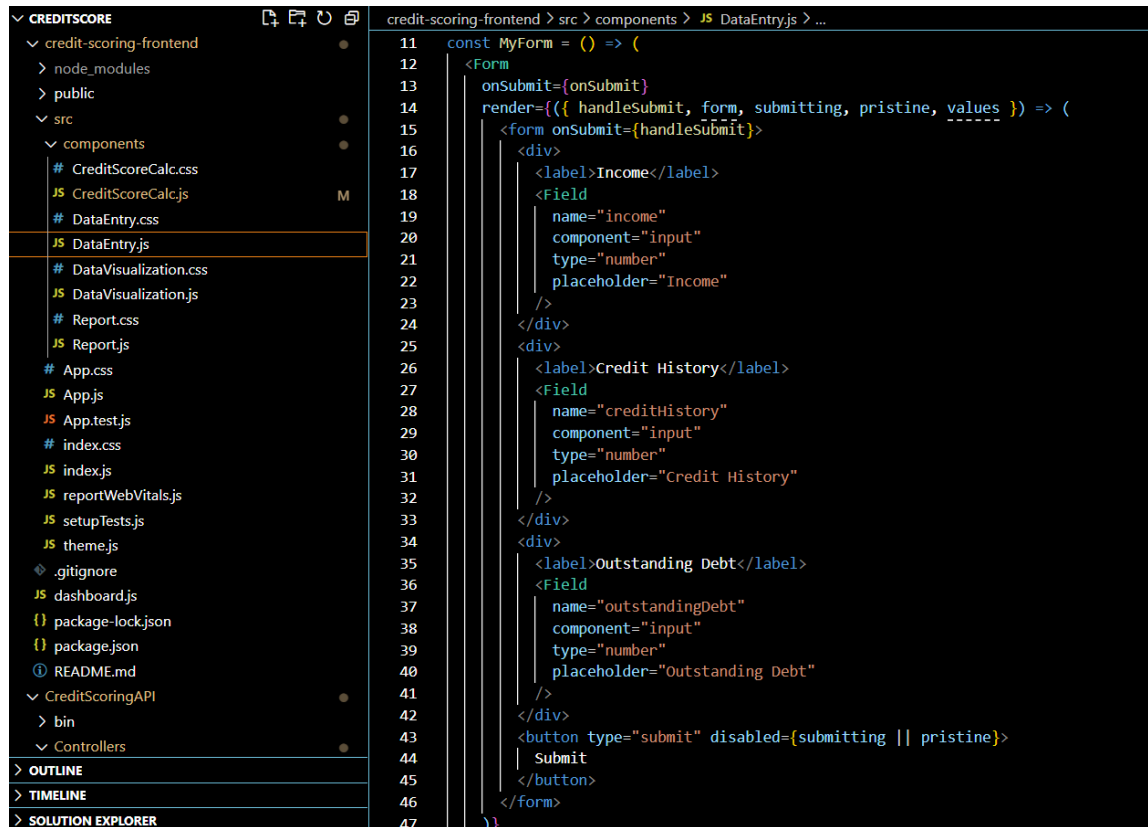
## Component Architecture Diagram



# Frontend

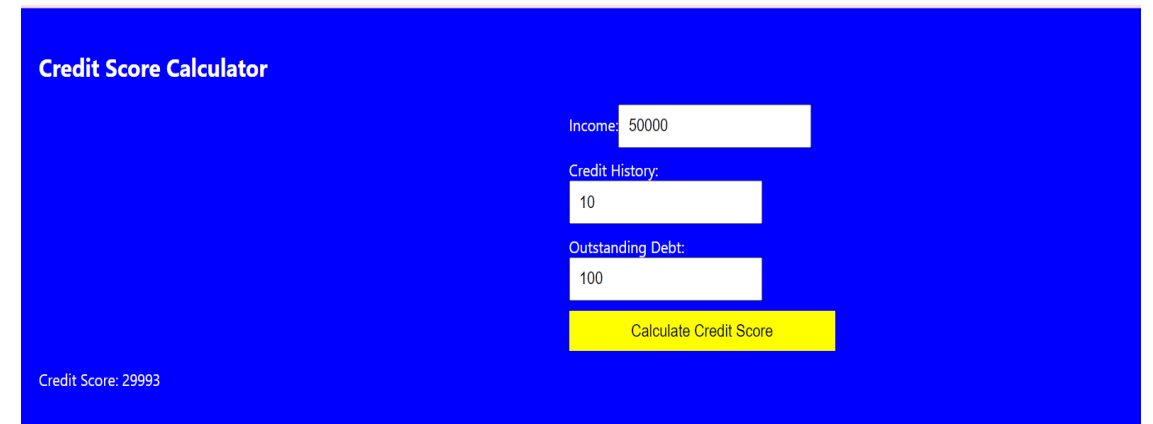
- **Description:**
  - The frontend component in the project is built using ReactJS. It's responsible for providing a user-friendly interface for bank analysts to interact with the credit scoring system.
- **User Interface (UI) Components:**
  - The user interface is designed to be straightforward and easy to use. It consists of:
    - A form for bank analysts to input customer data, including income, credit history, and outstanding debt.
    - A "Calculate Credit Score" button to initiate the credit score calculation.
    - A display area to show the calculated credit score.

# Frontend



```
11 const MyForm = () => (  
12   <Form  
13     onSubmit={onSubmit}  
14     render={({ handleSubmit, form, submitting, pristine, values }) => (  
15       <form onSubmit={handleSubmit}>  
16         <div>  
17           <label>Income</label>  
18           <Field  
19             name="income"  
20             component="input"  
21             type="number"  
22             placeholder="Income"  
23           />  
24         </div>  
25         <div>  
26           <label>Credit History</label>  
27           <Field  
28             name="creditHistory"  
29             component="input"  
30             type="number"  
31             placeholder="Credit History"  
32           />  
33         </div>  
34         <div>  
35           <label>Outstanding Debt</label>  
36           <Field  
37             name="outstandingDebt"  
38             component="input"  
39             type="number"  
40             placeholder="Outstanding Debt"  
41           />  
42         </div>  
43         <button type="submit" disabled={submitting || pristine}>  
44           Submit  
45         </button>  
46       </form>  
47     )  
11
```

*Data Input code snippet*



**Credit Score Calculator**

Income:

Credit History:

Outstanding Debt:

Credit Score: 29993

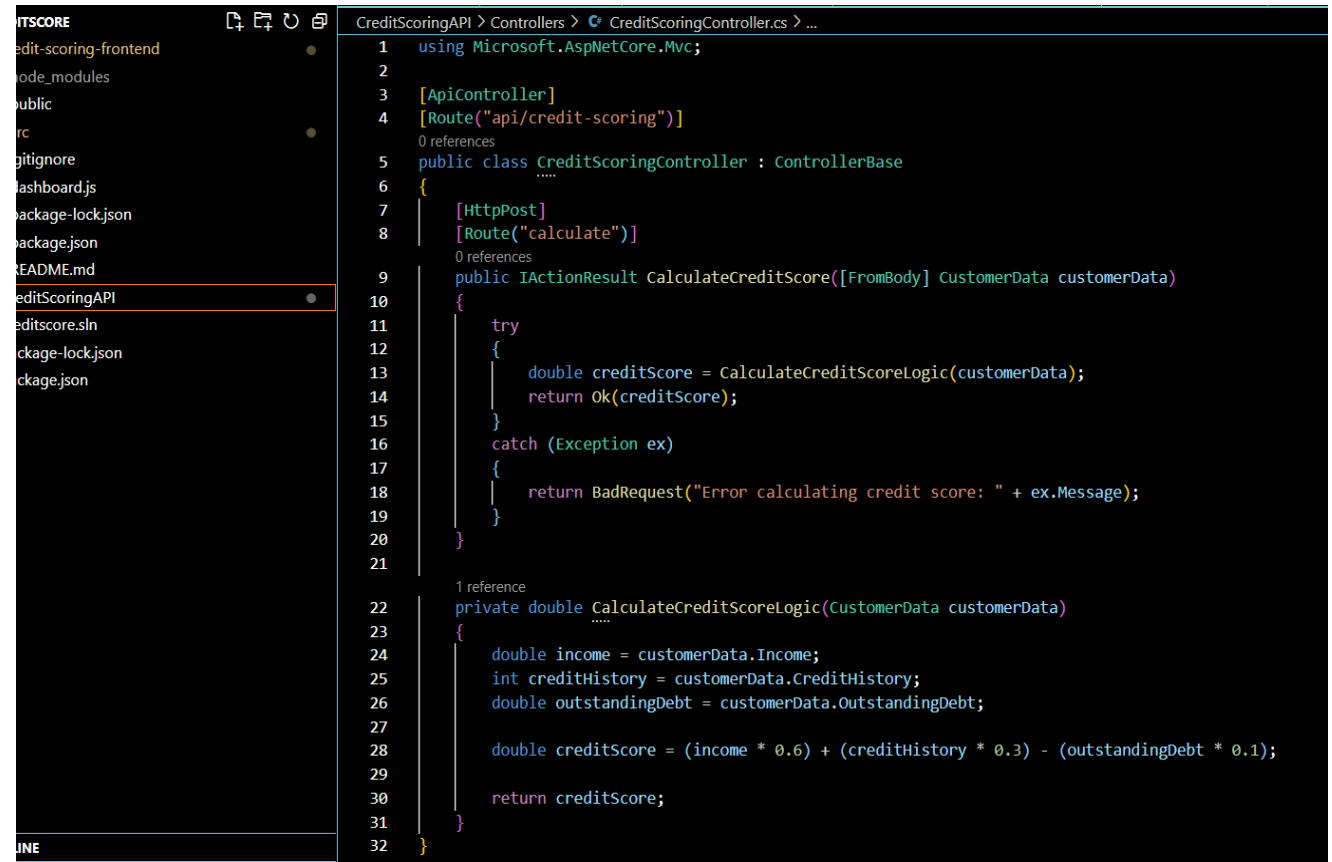
*Low-fidelity prototype of UI*



# Backend

- **Description:**

- The backend component, developed in C#, serves as the server-side application responsible for handling API requests from the frontend (ReactJS). Its primary role is to process incoming data, perform credit score calculations, and return the calculated credit scores to the frontend. It also handles CORS (Cross-Origin Resource Sharing) to enable cross-origin requests from the frontend.



```
1 using Microsoft.AspNetCore.Mvc;
2
3 [ApiController]
4 [Route("api/credit-scoring")]
5 public class CreditScoringController : ControllerBase
6 {
7     [HttpPost]
8     [Route("calculate")]
9     public IActionResult CalculateCreditScore([FromBody] CustomerData customerData)
10    {
11        try
12        {
13            double creditScore = CalculateCreditScoreLogic(customerData);
14            return Ok(creditScore);
15        }
16        catch (Exception ex)
17        {
18            return BadRequest("Error calculating credit score: " + ex.Message);
19        }
20    }
21
22    private double CalculateCreditScoreLogic(CustomerData customerData)
23    {
24        double income = customerData.Income;
25        int creditHistory = customerData.CreditHistory;
26        double outstandingDebt = customerData.OutstandingDebt;
27
28        double creditScore = (income * 0.6) + (creditHistory * 0.3) - (outstandingDebt * 0.1);
29
30        return creditScore;
31    }
32 }
```

*Backend controller code snippet*

# Backend

## Handling Incoming Requests:

**1.API Endpoint (/api/credit-scoring/calculate):** The backend listens for POST requests at the **/api/credit-scoring/calculate** endpoint. When a request is received, it triggers the **CalculateCreditScore** method.

**2.CalculateCreditScore Method:** In the **CalculateCreditScore** method, customer data (income, credit history, outstanding debt) is received from the frontend. The method extracts this data and passes it to the **CalculateCreditScoreLogic** method for credit score calculation.

**3.CalculateCreditScoreLogic Method:** This is where the credit score is calculated. The formula used is straightforward:  $\text{income} * 0.6 + \text{credit history} * 0.3 - \text{outstanding debt} * 0.1$ . The result is the calculated credit score.

**4.Response:** The calculated credit score is then returned as a response to the frontend, indicating the success of the request. If any errors occur during the calculation, an error response is sent.

```
credit-scoring-frontend > src > components > JS CreditScoreCalc.js > ...
1  import React, { useState } from 'react';
2  import './CreditScoreCalc.css'; // Import the CSS file
3  import Report from './Report';
4
5  // reference: https://react.dev/learn/state-a-components-memory
6
7  function CreditScoreCalc() {
8    const [income, setIncome] = useState(0);
9    const [creditHistory, setCreditHistory] = useState(0);
10   const [outstandingDebt, setOutstandingDebt] = useState(0);
11   const [creditScore, setCreditScore] = useState(null);
12   const [showReport, setShowReport] = useState(false);
13
14   const calculateCreditScore = (e) => {
15     e.preventDefault(); // prevents default submission behaviour
16     // code reference: https://stackoverflow.com/questions/19276853/preventing-form-submit-when-button-clicked
17
18     // Example calculation:
19     const calculatedScore = income * 0.6 + creditHistory * 0.3 - outstandingDebt * 0.1;
20     setCreditScore(calculatedScore);
21     setShowReport(true);
22   };
23
24   return (
25     <div className="CreditScoreCalc-Container">
26       <h2>Credit Score Calculator</h2>
27       <form>
28         <div>
29           <label>Income:</label>
30           <input type="number" value={income} onChange={(e) => setIncome(parseFloat(e.target.value))} />
31         </div>
32         <div>
33           <label>Credit History:</label>
34           <input type="number" value={creditHistory} onChange={(e) => setCreditHistory(parseInt(e.target.value))} />
35         </div>
36         <div>
```

*Calculation code snippet*

# Calculation Method

- **Income (Weight: 60%):** Income is a critical factor in credit scoring. The algorithm assigns it the highest weight, 60%. A higher income indicates a borrower's ability to repay debts and, therefore, positively impacts the credit score. A lower income might suggest a higher credit risk.
- **Credit History (Weight: 30%):** Credit history is another vital component. It is assigned a weight of 30%. A positive credit history, such as timely payments and responsible credit use, will have a significant positive influence on the credit score. A poor credit history, with late payments or defaults, can lower the score.
- **Outstanding Debt (Weight: 10%):** Outstanding debt is assigned a weight of 10%. It reflects the borrower's current level of debt relative to their income. High outstanding debt may imply a greater risk, as it can impact the borrower's ability to repay new debts.

This algorithm is a simplified representation of a credit scoring model. In practice, credit scoring models used by financial institutions can be significantly more complex, considering additional factors like the type of credit used, length of credit history, and recent credit inquiries.

```
credit-scoring-frontend > src > components > JS CreditScoreCalc.js > ...
1  import React, { useState } from 'react';
2  import './CreditScoreCalc.css'; // Import the CSS file
3  import Report from './Report';
4
5  // reference: https://react.dev/learn/state-a-components-memory
6
7  function CreditScoreCalc() {
8    const [income, setIncome] = useState(0);
9    const [creditHistory, setCreditHistory] = useState(0);
10   const [outstandingDebt, setOutstandingDebt] = useState(0);
11   const [creditscore, setCreditscore] = useState(null);
12   const [showReport, setShowReport] = useState(false);
13
14   const calculateCreditscore = (e) => {
15     e.preventDefault(); // prevents default submission behaviour
16     // code reference: https://stackoverflow.com/questions/19276853/preventing-form-submit-when-button-clicked
17
18     // Example calculation:
19     const calculatedScore = income * 0.6 + creditHistory * 0.3 - outstandingDebt * 0.1;
20     setCreditscore(calculatedScore);
21     setShowReport(true);
22   };
23
24   return (
25     <div className="CreditScoreCalc-Container">
26       <h2>Credit Score Calculator</h2>
27       <form>
28         <div>
29           <label>Income:</label>
30           <input type="number" value={income} onChange={(e) => setIncome(parseFloat(e.target.value))} />
31         </div>
32         <div>
33           <label>Credit History:</label>
34           <input type="number" value={creditHistory} onChange={(e) => setCreditHistory(parseInt(e.target.value))} />
35         </div>
36         <div>
```

*Calculation code snippet*

# Backend

- **CORS Configuration:** The backend's CORS configuration is designed to allow the ReactJS frontend to make requests to the C# backend, despite being on different origins.
- `policy.AllowAnyOrigin()`: In the code, the policy is configured to allow requests from any origin, which means that the ReactJS application can send requests from any domain.
- **Simplified Handling:** The provided code simplifies the CORS configuration by allowing any origin, any HTTP headers, and methods. This simplification reduces complexity and makes it easier to understand how cross-origin requests are permitted.

The code is intentionally kept simple and clear for ease of understanding. It demonstrates how the backend processes incoming requests and calculates credit scores with a straightforward formula and are key attributes that contribute to its maintainability and ease of use for future development and collaboration.



**Challenges**

# Reports Generation

```
credit-scoring-frontend > src > components > JS Report.js > ...
1 import jsPDF from 'jspdf';
2
3 const generateReport = (income, creditHistory, outstandingDebt, creditscore) => {
4   const doc = new jsPDF();
5   doc.text('Credit Score Report', 10, 10);
6
7   // Add credit score information to the report
8   doc.text(`Income: ${income}`, 10, 30);
9   doc.text(`Credit History: ${creditHistory}`, 10, 40);
10  doc.text(`Outstanding Debt: ${outstandingDebt}`, 10, 50);
11  doc.text(`Credit Score: ${creditscore}`, 10, 60);
12
13  doc.save('report.pdf');
14 };
15
16 export default generateReport;
17
```

```
return (
  <div className="CreditScoreCalc-Container">
    <h2>Credit Score Calculator</h2>
    <form>
      <div>
        <label>Income</label>
        <input type="number" value={income} onChange={(e) => setIncome(parseFloat(e.target.value))} />
      </div>
      <div>
        <label>Credit History</label>
        <input type="number" value={creditHistory} onChange={(e) => setCreditHistory(parseInt(e.target.value))} />
      </div>
      <div>
        <label>Outstanding Debt</label>
        <input type="number" value={outstandingDebt} onChange={(e) => setOutstandingDebt(parseFloat(e.target.value))} />
      </div>
      <button className="CreditScoreCalc-Button" onClick={calculateCreditScore}>
        Calculate Credit Score
      </button>
    </form>
    {creditscore !== null && (
      <div>
        <p>Credit Score: {creditscore}</p>
      </div>
    )}
    {showReport && <report />} {/* Render Report component when showReport is true */}
  </div>
);

export default CreditScoreCalc;
```

- When the "Calculate Credit Score" button is clicked, the form data is processed and used to calculate the credit score.
- The generateReport function is then called with the credit-related data to create a report.
- The report is displayed as a downloadable PDF, providing a summary of the credit score.

## Code Explanation:

- We use the jsPDF library to create PDF documents, making it an efficient tool for generating reports.
- In the code, we populate the report with data including income, credit history, outstanding debt, and the calculated credit score.
- The doc.save('report.pdf') line enables users to download the generated report as 'report.pdf'.

# Data Visualization

```
credit-scoring-frontend > src > components > JS DataVisualization.js > data > datasets
1 import React from 'react';
2 import { Bar } from 'react-chartjs-2';
3 import { registerables } from 'chart.js';
4 import {Chart } from 'react-chartjs-2';
5
6
7 Chart.register(...registerables);
8
9
10 const data = {
11   labels: ['Income', 'Credit History', 'Outstanding Debt'],
12   datasets: [
13     {
14       label: 'Credit Score Components',
15       data: [12, 19, 3],
16       backgroundColor: [
17         'rgba(255, 99, 132, 0.2)',
18         'rgba(54, 162, 235, 0.2)',
19         'rgba(255, 206, 86, 0.2)',
20       ],
21       borderColor: [
22         'rgba(255, 99, 132, 1)',
23         'rgba(54, 162, 235, 1)',
24         'rgba(255, 206, 86, 1)',
25       ],
26       borderWidth: 1,
27     },
28   ],
29 };
30
31 const DataVisualization = () => {
32   return (
33     <div>
34       <h2>Data Visualization</h2>
35       <Bar data={data} />
36     </div>
37   );
38 }
```

A popular library called Chart.js was used to create interactive and visually appealing charts and graphs.

It's widely used for data visualization and offers a straightforward way to represent complex data in an understandable manner.

## Tools and Libraries:

For this project, two main tools or libraries were used:

- React-chartjs-2: This is a wrapper for Chart.js that allows us to seamlessly integrate chart components into our React application.
- Chart.js: The core library for creating the charts and graphs. It provides us with a range of chart types and customization options.

This code snippet is a simple example of a bar chart. It visualizes credit score components, such as income, credit history, and outstanding debt.

# Future Enhancements

- UI – Due to time constraint, the UI is developed to be of a low-fidelity prototype.
- Complete Data Visualization - The data visualization feature is not fully operational yet. In the future, we plan to enhance this component by implementing interactive and insightful data visualization. This will allow bank analysts to gain a deeper understanding of credit score trends over time.
- Robust Report Generation - focus on improving the report generation process will be emphasized to ensure that it accurately reflects the analysed data. Bank analysts should be able to generate meaningful reports based on customer data.
- User Authentication and Authorization - To enhance the security of the application, user authentication and authorizations would be in place. This will restrict access to sensitive customer data and ensure that only authorized personnel can use the system.
- Real-time Data Updates - Real-time data updates are crucial for bank analysts who need the most up-to-date information.

<https://github.com/mishramohammad/creditscoredashboard.git>



