Project Overview

This POC simulates a Loan Officer's home interface with a dashboard displaying loan data, a sliding AI chatbot for querying information, and a backend system to handle data requests. The key components are:

- **Pseudo Browser Interface**: A React-based dashboard showing a table of loans.
- **Slider with AI Chatbot**: A right-sliding panel with a rule-based AI chatbot (recommended for simplicity in the POC) that integrates an MCP client for API calls.
- **MCP Client**: A JavaScript module to handle communication with the MCP server.
- **MCP Server**: A Node.js/Express server interacting with a fake JSON database (loans.json).
- **Fake Database**: A JSON file with sample loan data, simulating the Loan Origination and Accounting Systems.

The deliverables include:

- A detailed folder structure.
- Specific files with their contents.
- Step-by-step instructions for Cursor to execute.
- A LaTeX document for PDF export, ensuring all steps are clear and actionable.

Project Structure

The project is split into two main directories: client/ for the React frontend and server/ for the MCP server.

text
Copy

```
loan-chatbot-poc/
├── client/
│   ├── public/
│   │   └── index.html
│   ├── src/
│   │   ├── components/
│   │   │   ├── Dashboard.js
│   │   │   └── Chatbot.js
│   │   ├── mcp/
│   │   │   └── client.js
│   │   ├── App.js
```

```
│   │   ├── index.js
│   │   └── index.css
│   ├── tailwind.config.js
│   ├── postcss.config.js
│   ├── package.json
│   └── README.md
├── server/
│   ├── server.js
│   ├── loans.json
│   ├── package.json
│   └── README.md
└── README.md
```

Step-by-Step Implementation Plan for Cursor

Below are the detailed steps to build the POC, ensuring Cursor can follow them precisely. Each step includes commands, file contents, and explanations.

Step 1: Create the Project Folder Structure

- **Action**: Create the root folder and subdirectories.
- **Commands**:

  bash

  Copy

  ```
  mkdir loan-chatbot-poc

  cd loan-chatbot-poc

  mkdir client server

  cd client

  mkdir public src

  cd src
  ```

```
mkdir components mcp

cd ../..
```

- **Purpose**: Establishes the folder hierarchy for organizing client and server code.

Step 2: Initialize the Client (React) Project

- **Action**: Set up a new React project in the client/ directory.
- **Commands**:

  bash

  Copy

```
cd client

npx create-react-app .
```

- **Purpose**: Creates a React application with a default structure, which we'll modify.

Step 3: Install Client Dependencies

- **Action**: Install required libraries for the React frontend.
- **Commands**:

  bash

  Copy

```
cd client

npm install axios react-sliding-pane tailwindcss postcss autoprefixer
```

- **Purpose**:
  - axios: For making API calls to the MCP server.
  - react-sliding-pane: For the right-sliding chatbot panel.
  - tailwindcss, postcss, autoprefixer: For styling the UI.

Step 4: Configure Tailwind CSS

- **Action**: Initialize Tailwind CSS and create configuration files.
- **Commands**:

bash

Copy

```
cd client

npx tailwindcss init
```

- **File**: client/tailwind.config.js

  javascript

  Copy

  ```
  module.exports = {
    content: ['./src/**/*.{js,jsx,ts,tsx}'],
    theme: {
      extend: {},
    },
    plugins: [],
  };
  ```

- **File**: client/postcss.config.js

  javascript

  Copy

  ```
  module.exports = {
    plugins: {
      tailwindcss: {},
      autoprefixer: {},
  ```

```
  },
};
```

- **File**: client/src/index.css

  css

  Copy

```
@tailwind base;
@tailwind components;
@tailwind utilities;

/* Custom styles */
.chatbot-container {
  max-height: 80vh;
  overflow-y: auto;
}
```

- **Purpose**: Sets up Tailwind CSS for responsive styling across the application.

Step 5: Create the MCP Client Module

- **Action**: Create a JavaScript module to handle API calls to the MCP server.
- **File**: client/src/mcp/client.js

  javascript

  Copy

```
import axios from 'axios';
```

```
const mcpClient = {
  baseURL: 'http://localhost:3001/api',
  async getAllLoans() {
    try {
      const response = await axios.get(`${this.baseURL}/loans`);
      return response.data;
    } catch (error) {
      console.error('Error fetching loans:', error);
      return [];
    }
  },
  async getLoanStatus(loanId) {
    const response = await
axios.get(`${this.baseURL}/loan/status/${loanId}`);
    return response.data.status;
  },
  async getLoanDetails(loanId) {
    const response = await axios.get(`${this.baseURL}/loan/${loanId}`);
    return response.data;
  },
  async getActiveLoans() {
```

```
    const response = await axios.get(`${this.baseURL}/loans/active`);
    return response.data;
  },
  async getLoansByBorrower(borrower) {
    const response = await
axios.get(`${this.baseURL}/loans/borrower/${encodeURIComponent(borrower
)}`);
    return response.data;
  },
};


export default mcpClient;
```

- **Purpose**: Provides functions to interact with the MCP server, integrated into the chatbot.

Step 6: Create the Dashboard Component

- **Action**: Create a component for the Loan Officer's home view, displaying a table of loans.
- **File**: client/src/components/Dashboard.js

  javascript

  Copy

```
import React, { useState, useEffect } from 'react';
import mcpClient from '../mcp/client';


const Dashboard = () => {
```

```
  const [loans, setLoans] = useState([]);



  useEffect(() => {
    const fetchLoans = async () => {
      const loansData = await mcpClient.getAllLoans();
      setLoans(loansData);
    };
    fetchLoans();
  }, []);



  return (
    <div className="p-4">
      <h1 className="text-2xl font-bold mb-4">Loan Officer Home</h1>
      <p className="text-gray-600 mb-4">View your loan portfolio below.
Use the AI chatbot to query specific details.</p>
      <table className="min-w-full bg-white border">
        <thead>
          <tr>
            <th className="py-2 px-4 border-b">Loan ID</th>
            <th className="py-2 px-4 border-b">Borrower</th>
            <th className="py-2 px-4 border-b">Amount</th>
```

```
              <th className="py-2 px-4 border-b">Status</th>
            </tr>
          </thead>
          <tbody>
            {loans.map((loan) => (
              <tr key={loan.id}>
                <td className="py-2 px-4 border-b">{loan.id}</td>
                <td className="py-2 px-4 border-b">{loan.borrower}</td>
                <td className="py-2 px-4 border-
b">${loan.amount.toLocaleString()}</td>
                <td className="py-2 px-4 border-b">{loan.status}</td>
              </tr>
            ))}
          </tbody>
        </table>
      </div>
    );
};

export default Dashboard;
```

- **Purpose**: Displays a table of loans, fetched from the MCP server, simulating the Loan Officer's home interface.

Step 7: Create the Chatbot Component

- **Action**: Create a component for the AI chatbot, integrated with the MCP client, inside the sliding panel.
- **File**: client/src/components/Chatbot.js

  javascript

  Copy

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

import mcpClient from '../mcp/client';



const Chatbot = ({ onClose }) => {

  const [messages, setMessages] = useState([

    { text: 'Hello! Ask about loan status, details, active loans, or loans by borrower.', sender: 'bot' },

  ]);

  const [input, setInput] = useState('');



  const parseMessage = (message) => {

    const lowerMsg = message.toLowerCase().replace(/[^\w\s]/g, '');

    if (lowerMsg.includes('status') && lowerMsg.includes('loan') && /\d+/.test(lowerMsg)) {

      const loanId = lowerMsg.match(/\d+/)[0];
```

```
      return { action: 'getStatus', loanId };
    } else if (lowerMsg.includes('details') &&
lowerMsg.includes('loan') && /\d+/.test(lowerMsg)) {
      const loanId = lowerMsg.match(/\d+/)[0];
      return { action: 'getDetails', loanId };
    } else if (lowerMsg.includes('active loans')) {
      return { action: 'getActiveLoans' };
    } else if (lowerMsg.includes('loans for')) {
      const borrower = lowerMsg.split('for')[1]?.trim();
      return { action: 'getLoansByBorrower', borrower };
    } else {
      return { action: 'unknown' };
    }
  };

  const handleSend = async () => {
    if (!input.trim()) return;
    const userMessage = { text: input, sender: 'user' };
    setMessages((prev) => [...prev, userMessage]);

    const parsed = parseMessage(input);
```

```
    let responseText;

    try {

      if (parsed.action === 'getStatus') {

        const status = await mcpClient.getLoanStatus(parsed.loanId);

        responseText = `The status of loan ${parsed.loanId} is
${status}.`;

      } else if (parsed.action === 'getDetails') {

        const details = await mcpClient.getLoanDetails(parsed.loanId);

        responseText = `Loan ${details.id}:\nBorrower:
${details.borrower}\nAmount:
$${details.amount.toLocaleString()}\nStatus: ${details.status}`;

      } else if (parsed.action === 'getActiveLoans') {

        const activeLoans = await mcpClient.getActiveLoans();

        responseText = activeLoans.length > 0

          ? `Active loans: ${activeLoans.map((l) => l.id).join(', ')}`

          : 'No active loans found.';

      } else if (parsed.action === 'getLoansByBorrower') {

        const loans = await
mcpClient.getLoansByBorrower(parsed.borrower);

        responseText = loans.length > 0

          ? `Loans for ${parsed.borrower}: ${loans.map((l) =>
l.id).join(', ')}`
```

```
          : `No loans found for ${parsed.borrower}.`;
      } else {
        responseText = "Sorry, I didn't understand. Try: 'status of loan 12345', 'details of loan 12345', 'active loans', or 'loans for John Doe'.";
      }
    } catch (error) {
      responseText = 'Error fetching data. Please check the loan ID or try again.';
    }

    const botMessage = { text: responseText, sender: 'bot' };
    setMessages((prev) => [...prev, botMessage]);
    setInput('');
  };

  return (
    <div className="h-full flex flex-col">
      <div className="flex-1 overflow-y-auto p-4 chatbot-container">
        {messages.map((msg, idx) => (
          <div key={idx} className={`mb-2 ${msg.sender === 'user' ? 'text-right' : 'text-left'}`}>
```

```
        <span
          className={`inline-block p-2 rounded ${
            msg.sender === 'user' ? 'bg-blue-200' : 'bg-gray-200'
          }`}
        >
          {msg.text.split('\n').map((line, i) => (
            <div key={i}>{line}</div>
          ))}
        </span>
      </div>
    ))}
  </div>
  <div className="p-4 border-t">
    <input
      type="text"
      value={input}
      onChange={(e) => setInput(e.target.value)}
      onKeyPress={(e) => e.key === 'Enter' && handleSend()}
      className="w-full p-2 border rounded"
      placeholder="Ask about loans..."
```

```
          />
          <button
            onClick={handleSend}
            className="mt-2 bg-blue-500 text-white px-4 py-2 rounded"
          >
            Send
          </button>
        </div>
      </div>
    );
  };

  export default Chatbot;
```

- **Purpose**: Implements the AI chatbot with rule-based parsing, integrated with the MCP client for querying loan data.

Step 8: Update the Main App Component

- **Action**: Integrate the Dashboard and Chatbot components with a sliding panel.
- **File**: client/src/App.js

  javascript

  Copy

```
import React, { useState } from 'react';
import SlidingPane from 'react-sliding-pane';
```

```
import 'react-sliding-pane/dist/react-sliding-pane.css';

import Dashboard from './components/Dashboard';

import Chatbot from './components/Chatbot';



function App() {

  const [isSliderOpen, setIsSliderOpen] = useState(false);



  return (

    <div className="min-h-screen bg-gray-100">

      <Dashboard />

      <button

        className="fixed bottom-4 right-4 bg-blue-500 text-white px-4
py-2 rounded-full shadow-lg"

        onClick={() => setIsSliderOpen(true)}

      >

        Chat with AI

      </button>

      <SlidingPane

        isOpen={isSliderOpen}

        title="AI Chatbot"

        onRequestClose={() => setIsSliderOpen(false)}
```

```
          from="right"
          width="300px"
        >
          <Chatbot onClose={() => setIsSliderOpen(false)} />
        </SlidingPane>
      </div>
    );
  }

export default App;
```

- **Purpose**: Combines the dashboard and chatbot, with a button to toggle the sliding panel.

Step 9: Update the Index File

- **Action**: Ensure the React app renders correctly.
- **File**: client/src/index.js

  javascript

  Copy

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
```

```
ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

- **Purpose**: Sets up the entry point for the React application.

Step 10: Create the Public Index HTML

- **Action**: Ensure the HTML file is set up for the React app.
- **File**: client/public/index.html

html

Copy

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Loan Officer Chatbot POC</title>
  </head>
  <body>
    <div id="root"></div>
```

```
  </body>

</html>
```

- **Purpose**: Provides the HTML structure for the React app.

Step 11: Initialize the MCP Server

- **Action**: Set up a Node.js/Express server in the server/ directory.
- **Commands**:

  bash

  Copy

```
cd server

npm init -y

npm install express cors
```

- **File**: server/package.json

  json

  Copy

```
{

  "name": "mcp-server",

  "version": "1.0.0",

  "main": "server.js",

  "scripts": {

    "start": "node server.js"

  },

  "dependencies": {
```

```
    "cors": "^2.8.5",
    "express": "^4.17.1"
  }
}
```

- **Purpose**: Initializes the server project with dependencies for Express and CORS.

Step 12: Create the Fake Database

- **Action**: Create a JSON file with sample loan data.
- **File**: server/loans.json

  json

  Copy

```
[
  {
    "id": "12345",
    "borrower": "John Doe",
    "amount": 50000,
    "status": "Active",
    "interestRate": 3.5,
    "paymentHistory": ["On time", "On time", "Late"]
  },
  {
    "id": "67890",
```

```
    "borrower": "Jane Smith",
    "amount": 30000,
    "status": "Pending",
    "interestRate": 4.0,
    "paymentHistory": []
  },
  {
    "id": "23456",
    "borrower": "John Doe",
    "amount": 20000,
    "status": "Active",
    "interestRate": 3.0,
    "paymentHistory": ["On time"]
  },
  {
    "id": "34567",
    "borrower": "Alice Johnson",
    "amount": 40000,
    "status": "Closed",
    "interestRate": 3.8,
```

```
    "paymentHistory": ["On time", "On time"]
  }
]
```

- **Purpose**: Acts as a fake database for the MCP server, simulating loan data.

Step 13: Create the MCP Server

- **Action**: Implement the Express server to handle API requests.
- **File**: server/server.js

  javascript

  Copy

```
const express = require('express');
const cors = require('cors');
const loans = require('./loans.json');

const app = express();
app.use(cors());
app.use(express.json());

// Get all loans
app.get('/api/loans', (req, res) => {
  res.json(loans);
});
```

```
// Get loan by ID
app.get('/api/loan/:id', (req, res) => {
  const loan = loans.find((l) => l.id === req.params.id);
  if (loan) {
    res.json(loan);
  } else {
    res.status(404).json({ error: 'Loan not found' });
  }
});

// Get loan status by ID
app.get('/api/loan/status/:id', (req, res) => {
  const loan = loans.find((l) => l.id === req.params.id);
  if (loan) {
    res.json({ status: loan.status });
  } else {
    res.status(404).json({ error: 'Loan not found' });
  }
});
```

```
// Get active loans
app.get('/api/loans/active', (req, res) => {
  const activeLoans = loans.filter((l) => l.status === 'Active');
  res.json(activeLoans);
});

// Get loans by borrower
app.get('/api/loans/borrower/:name', (req, res) => {
  const borrowerLoans = loans.filter(
    (l) => l.borrower.toLowerCase() === req.params.name.toLowerCase()
  );
  res.json(borrowerLoans);
});

const PORT = 3001;
app.listen(PORT, () => {
  console.log(`MCP server running on port ${PORT}`);
});
```

- **Purpose**: Provides RESTful API endpoints for the MCP client to query loan data.

Step 14: Document the Client

- **Action**: Create a README for the client.
- **File**: client/README.md

  markdown

  Copy

  ```
  # Loan Chatbot POC - Client



  This is the React frontend for the Loan Officer Chatbot POC, featuring
  a dashboard and a sliding AI chatbot.



  ## Prerequisites

  - Node.js and npm installed

  - MCP server running at `http://localhost:3001` (see
  `server/README.md`)



  ## Setup

  1. Navigate to the client directory:

     ```bash

     cd client
  ```

  2. Install dependencies:

     bash

     Copy
  ```

```
npm install
```

3. Start the React app:

   bash

   Copy

   ```
   npm start
   ```

   The app will open at http://localhost:3000.

Features

- **Dashboard**: Displays a table of loans (Loan ID, Borrower, Amount, Status).
- **Chatbot**: A sliding panel from the right, allowing queries like:
  - "status of loan 12345"
  - "details of loan 12345"
  - "active loans"
  - "loans for John Doe"

Notes

- The chatbot uses a rule-based AI for the POC. In production, consider integrating Dialogflow or Rasa for NLP.
- The MCP client (src/mcp/client.js) connects to http://localhost:3001/api. Update the baseURL for a real MCP server.

- **Purpose**: Guides developers on setting up and running the client.

Step 15: Document the Server

- **Action**: Create a README for the server.
- **File**: server/README.md

  markdown

  Copy

  ```
  # Loan Chatbot POC - MCP Server
  ```

```
This is the Node.js/Express server simulating an MCP server,
interacting with a fake JSON database.


## Prerequisites

- Node.js and npm installed


## Setup

1. Navigate to the server directory:

   ```bash

   cd server
```

2. Install dependencies:

   bash

   Copy

   ```
   npm install
   ```

3. Start the server:

   bash

   Copy

   ```
   npm start
   ```

   The server will run at http://localhost:3001.

Endpoints

- GET /api/loans: Get all loans
- GET /api/loan/:id: Get loan details by ID
- GET /api/loan/status/:id: Get loan status by ID
   ```

    - GET /api/loans/active: Get all active loans
    - GET /api/loans/borrower/:name: Get loans by borrower name

Notes

    - Uses loans.json as a fake database.
    - In production, replace with real MCP endpoints connecting to the Loan Origination and Accounting Systems.

- **Purpose**: Guides developers on setting up and running the MCP server.

Step 16: Document the Entire Project

- **Action**: Create a root README for the POC.
- **File**: README.md

markdown

Copy

```
# Loan Chatbot Proof of Concept



This POC demonstrates an AI chatbot interface for Loan Officers, with a dashboard and a sliding chatbot panel.



## Structure

- `client/`: React frontend with dashboard and chatbot.

- `server/`: Node.js/Express MCP server with a fake JSON database.



## Setup

1. **Start the MCP server**:

   ```bash
```
```

```
cd server

npm install

npm start
```

2. **Start the client**:

   bash

   Copy

   ```
   cd client

   npm install

   npm start
   ```

3. Open http://localhost:3000 in a browser.

Testing

- View the dashboard to see the loan table.
- Click "Chat with AI" to open the chatbot.
- Try queries like:
  - "status of loan 12345"
  - "details of loan 12345"
  - "active loans"
  - "loans for John Doe"

Notes

- This is a POC with a rule-based AI. For production, integrate an NLP service like Dialogflow.
- The MCP client connects to a mock server. Update client/src/mcp/client.js for real MCP endpoints.

- **Purpose**: Provides an overview and setup instructions for the entire POC.

Step 17: Test the Application

- **Action**: Run both the server and client to verify functionality.
- **Commands**:

bash

Copy

```
# In one terminal

cd server

npm start



# In another terminal

cd client

npm start
```

- **Steps**:
  1. Open http://localhost:3000.
  2. Verify the dashboard displays the loan table (e.g., loans from loans.json).
  3. Click “Chat with AI” to open the slider.
  4. Test queries:
     - “status of loan 12345” → Should return “The status of loan 12345 is Active.”
     - “details of loan 67890” → Should return loan details (e.g., borrower, amount).
     - “active loans” → Should list active loan IDs.
     - “loans for John Doe” → Should list John Doe’s loans.
- **Purpose**: Ensures the POC works as expected.

Step 18: Add Scalability Comments

- **Action**: Add comments to guide future MCP integration.
- **Files to Update**:
  - client/src/mcp/client.js:

    javascript

    Copy

    ```
    // In production, update baseURL to real MCP endpoint (e.g.,
    https://your-mcp-api.com)
    ```

```
baseURL: 'http://localhost:3001/api',
```

- client/src/components/Chatbot.js (in parseMessage):

  javascript

  Copy

```
// In production, replace rule-based parsing with an NLP service like Dialogflow or Rasa

const parseMessage = (message) => {
```

- **Purpose**: Prepares the codebase for production enhancements.

Step 19: Verify Error Handling

- **Action**: Ensure the chatbot handles errors gracefully.
- **Details**: The Chatbot.js and client.js files already include try-catch blocks:
  - Invalid loan IDs return "Error fetching data."
  - Unknown queries return a helpful message with example queries.
- **Purpose**: Improves user experience for the POC.

Step 20: Export to PDF

- **Action**: Create a LaTeX document for PDF export, detailing all steps.
- **File**: Loan_Chatbot_POC_Plan.tex

  latex

  Copy

```
\documentclass{article}

\usepackage[utf8]{inputenc}

\usepackage{geometry}

\geometry{a4paper, margin=1in}

\usepackage{listings}

\usepackage{xcolor}
```

```
\usepackage{enumitem}
\usepackage{hyperref}

\lstset{
    basicstyle=\ttfamily\small,
    breaklines=true,
    columns=fullflexible,
    frame=single,
    numbers=left,
    numberstyle=\tiny\color{gray},
    keywordstyle=\color{blue},
    stringstyle=\color{red},
    commentstyle=\color{green!50!black},
}

\title{Loan Chatbot Proof of Concept Implementation Plan}
\author{}
\date{May 19, 2025}

\begin{document}
```

\maketitle

\section{Project Overview}

This document outlines the implementation of a proof-of-concept (POC) AI chatbot interface for Loan Officers. It includes a dashboard, a sliding AI chatbot, an MCP client, and an MCP server with a fake JSON database.

\begin{itemize}

\item \textbf{Frontend}: React with Tailwind CSS and react-sliding-pane.

\item \textbf{Backend}: Node.js/Express server with a JSON database.

\item \textbf{Functionality}: Dashboard displays loans; chatbot handles queries like loan status, details, active loans, and loans by borrower.

\end{itemize}

\section{Project Structure}

\begin{lstlisting}

loan-chatbot-poc/

├── client/

```
|   ├── public/
|   |   └── index.html
|   ├── src/
|   |   ├── components/
|   |   |   ├── Dashboard.js
|   |   |   └── Chatbot.js
|   |   ├── mcp/
|   |   |   └── client.js
|   |   ├── App.js
|   |   ├── index.js
|   |   └── index.css
|   ├── tailwind.config.js
|   ├── postcss.config.js
|   ├── package.json
|   └── README.md
├── server/
|   ├── server.js
|   ├── loans.json
|   ├── package.json
|   └── README.md
```

```
└── README.md
```

\end{lstlisting}

\section{Implementation Steps}

\subsection{Step 1: Create Project Folder Structure}

\begin{lstlisting}[language=bash]

```
mkdir loan-chatbot-poc
cd loan-chatbot-poc
mkdir client server
cd client
mkdir public src
cd src
mkdir components mcp
cd ../..
```

\end{lstlisting}

\subsection{Step 2: Initialize Client (React) Project}

\begin{lstlisting}[language=bash]

```
cd client
```

```
npx create-react-app .
\end{lstlisting}


\subsection{Step 3: Install Client Dependencies}
\begin{lstlisting}[language=bash]
cd client
npm install axios react-sliding-pane tailwindcss postcss autoprefixer
\end{lstlisting}


\subsection{Step 4: Configure Tailwind CSS}
\begin{lstlisting}[language=bash]
cd client
npx tailwindcss init
\end{lstlisting}
\textbf{File}: \texttt{client/tailwind.config.js}
\begin{lstlisting}[language=javascript]
module.exports = {
  content: ['./src/**/*.{js,jsx,ts,tsx}'],
  theme: { extend: {} },
  plugins: [],
```

```
};
```

\end{lstlisting}

\textbf{File}: \texttt{client/postcss.config.js}

\begin{lstlisting}[language=javascript]

```
module.exports = {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
};
```

\end{lstlisting}

\textbf{File}: \texttt{client/src/index.css}

\begin{lstlisting}[language=css]

```
@tailwind base;
@tailwind components;
@tailwind utilities;


.chatbot-container {
  max-height: 80vh;
  overflow-y: auto;
```

```
}
\end{lstlisting}


\subsection{Step 5: Create MCP Client Module}

\textbf{File}: \texttt{client/src/mcp/client.js}

\begin{lstlisting}[language=javascript]
import axios from 'axios';


const mcpClient = {
  // In production, update baseURL to real MCP endpoint (e.g.,
https://your-mcp-api.com)
  baseURL: 'http://localhost:3001/api',
  async getAllLoans() {
    try {
      const response = await axios.get(`${this.baseURL}/loans`);
      return response.data;
    } catch (error) {
      console.error('Error fetching loans:', error);
      return [];
    }
  },
```

```
  async getLoanStatus(loanId) {
    const response = await
axios.get(`${this.baseURL}/loan/status/${loanId}`);
    return response.data.status;
  },
  async getLoanDetails(loanId) {
    const response = await axios.get(`${this.baseURL}/loan/${loanId}`);
    return response.data;
  },
  async getActiveLoans() {
    const response = await axios.get(`${this.baseURL}/loans/active`);
    return response.data;
  },
  async getLoansByBorrower(borrower) {
    const response = await
axios.get(`${this.baseURL}/loans/borrower/${encodeURIComponent(borrower
)}`);
    return response.data;
  },
};

export default mcpClient;
```

```
\end{lstlisting}


\subsection{Step 6: Create Dashboard Component}

\textbf{File}: \texttt{client/src/components/Dashboard.js}

\begin{lstlisting}[language=javascript]
import React, { useState, useEffect } from 'react';
import mcpClient from '../mcp/client';


const Dashboard = () => {
  const [loans, setLoans] = useState([]);


  useEffect(() => {
    const fetchLoans = async () => {
      const loansData = await mcpClient.getAllLoans();
      setLoans(loansData);
    };
    fetchLoans();
  }, []);


  return (
```

```
    <div className="p-4">
      <h1 className="text-2xl font-bold mb-4">Loan Officer Home</h1>
      <p className="text-gray-600 mb-4">View your loan portfolio below. Use the AI chatbot to query specific details.</p>
      <table className="min-w-full bg-white border">
        <thead>
          <tr>
            <th className="py-2 px-4 border-b">Loan ID</th>
            <th className="py-2 px-4 border-b">Borrower</th>
            <th className="py-2 px-4 border-b">Amount</th>
            <th className="py-2 px-4 border-b">Status</th>
          </tr>
        </thead>
        <tbody>
          {loans.map((loan) => (
            <tr key={loan.id}>
              <td className="py-2 px-4 border-b">{loan.id}</td>
              <td className="py-2 px-4 border-b">{loan.borrower}</td>
              <td className="py-2 px-4 border-b">${loan.amount.toLocaleString()}</td>
              <td className="py-2 px-4 border-b">{loan.status}</td>
```

```
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  );
};

export default Dashboard;
\end{lstlisting}

\subsection{Step 7: Create Chatbot Component}
\textbf{File}: \texttt{client/src/components/Chatbot.js}
\begin{lstlisting}[language=javascript]
import React, { useState } from 'react';
import mcpClient from '../mcp/client';

const Chatbot = ({ onClose }) => {
  const [messages, setMessages] = useState([
    { text: 'Hello! Ask about loan status, details, active loans, or loans by borrower.', sender: 'bot' },
```

```
  ]);

  const [input, setInput] = useState('');


  // In production, replace rule-based parsing with an NLP service like Dialogflow or Rasa

  const parseMessage = (message) => {

    const lowerMsg = message.toLowerCase().replace(/[^\w\s]/g, '');

    if (lowerMsg.includes('status') && lowerMsg.includes('loan') && /\d+/.test(lowerMsg)) {

      const loanId = lowerMsg.match(/\d+/)[0];

      return { action: 'getStatus', loanId };

    } else if (lowerMsg.includes('details') && lowerMsg.includes('loan') && /\d+/.test(lowerMsg)) {

      const loanId = lowerMsg.match(/\d+/)[0];

      return { action: 'getDetails', loanId };

    } else if (lowerMsg.includes('active loans')) {

      return { action: 'getActiveLoans' };

    } else if (lowerMsg.includes('loans for')) {

      const borrower = lowerMsg.split('for')[1]?.trim();

      return { action: 'getLoansByBorrower', borrower };

    } else {

      return { action: 'unknown' };
```

```
    }
  };

  const handleSend = async () => {
    if (!input.trim()) return;
    const userMessage = { text: input, sender: 'user' };
    setMessages((prev) => [...prev, userMessage]);

    const parsed = parseMessage(input);
    let responseText;
    try {
      if (parsed.action === 'getStatus') {
        const status = await mcpClient.getLoanStatus(parsed.loanId);
        responseText = `The status of loan ${parsed.loanId} is
${status}.`;
      } else if (parsed.action === 'getDetails') {
        const details = await mcpClient.getLoanDetails(parsed.loanId);
        responseText = `Loan ${details.id}:\nBorrower:
${details.borrower}\nAmount:
$${details.amount.toLocaleString()}\nStatus: ${details.status}`;
      } else if (parsed.action === 'getActiveLoans') {
        const activeLoans = await mcpClient.getActiveLoans();
```

```
        responseText = activeLoans.length > 0
          ? `Active loans: ${activeLoans.map((l) => l.id).join(', ')}`
          : 'No active loans found.';
      } else if (parsed.action === 'getLoansByBorrower') {
        const loans = await
mcpClient.getLoansByBorrower(parsed.borrower);
        responseText = loans.length > 0
          ? `Loans for ${parsed.borrower}: ${loans.map((l) =>
l.id).join(', ')}`
          : `No loans found for ${parsed.borrower}.`;
      } else {
        responseText = "Sorry, I didn't understand. Try: 'status of
loan 12345', 'details of loan 12345', 'active loans', or 'loans for
John Doe'.";
      }
    } catch (error) {
      responseText = 'Error fetching data. Please check the loan ID or
try again.';
    }

    const botMessage = { text: responseText, sender: 'bot' };
    setMessages((prev) => [...prev, botMessage]);
```

```
    setInput('');
  };

  return (
    <div className="h-full flex flex-col">
      <div className="flex-1 overflow-y-auto p-4 chatbot-container">
        {messages.map((msg, idx) => (
          <div key={idx} className={`mb-2 ${msg.sender === 'user' ? 'text-right' : 'text-left'}`}>
            <span className={`inline-block p-2 rounded ${msg.sender === 'user' ? 'bg-blue-200' : 'bg-gray-200'}`}>
              {msg.text.split('\n').map((line, i) => (
                <div key={i}>{line}</div>
              ))}
            </span>
          </div>
        ))}
      </div>
      <div className="p-4 border-t">
        <input
          type="text"
```

```
          value={input}
          onChange={(e) => setInput(e.target.value)}
          onKeyPress={(e) => e.key === 'Enter' && handleSend()}
          className="w-full p-2 border rounded"
          placeholder="Ask about loans..."
        />
        <button onClick={handleSend} className="mt-2 bg-blue-500 text-white px-4 py-2 rounded">
          Send
        </button>
      </div>
    </div>
  );
};

export default Chatbot;
```

\end{lstlisting}

\subsection{Step 8: Update Main App Component}

\textbf{File}: \texttt{client/src/App.js}

\begin{lstlisting}[language=javascript]

```
import React, { useState } from 'react';
import SlidingPane from 'react-sliding-pane';
import 'react-sliding-pane/dist/react-sliding-pane.css';
import Dashboard from './components/Dashboard';
import Chatbot from './components/Chatbot';

function App() {
  const [isSliderOpen, setIsSliderOpen] = useState(false);

  return (
    <div className="min-h-screen bg-gray-100">
      <Dashboard />
      <button
        className="fixed bottom-4 right-4 bg-blue-500 text-white px-4 py-2 rounded-full shadow-lg"
        onClick={() => setIsSliderOpen(true)}
      >
        Chat with AI
      </button>
      <SlidingPane
        isOpen={isSliderOpen}
```

```
        title="AI Chatbot"
        onRequestClose={() => setIsSliderOpen(false)}
        from="right"
        width="300px"
      >
        <Chatbot onClose={() => setIsSliderOpen(false)} />
      </SlidingPane>
    </div>
  );
}

export default App;
```

\end{lstlisting}

\subsection{Step 9: Update Index File}

\textbf{File}: \texttt{client/src/index.js}

```
\begin{lstlisting}[language=javascript]
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
```

```
import App from './App';


ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
\end{lstlisting}


\subsection{Step 10: Create Public Index HTML}
\textbf{File}: \texttt{client/public/index.html}
\begin{lstlisting}[language=html]
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1"
/>
    <title>Loan Officer Chatbot POC</title>
  </head>
```

```
  <body>
    <div id="root"></div>
  </body>
</html>
\end{lstlisting}

\subsection{Step 11: Initialize MCP Server}
\begin{lstlisting}[language=bash]
cd server
npm init -y
npm install express cors
\end{lstlisting}
\textbf{File}: \texttt{server/package.json}
\begin{lstlisting}[language=json]
{
  "name": "mcp-server",
  "version": "1.0.0",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
```

```
  },
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.17.1"
  }
}
```

\end{lstlisting}

\subsection{Step 12: Create Fake Database}

\textbf{File}: \texttt{server/loans.json}

\begin{lstlisting}[language=json]

```
[
  {
    "id": "12345",
    "borrower": "John Doe",
    "amount": 50000,
    "status": "Active",
    "interestRate": 3.5,
    "paymentHistory": ["On time", "On time", "Late"]
  },
```

```
{
  "id": "67890",
  "borrower": "Jane Smith",
  "amount": 30000,
  "status": "Pending",
  "interestRate": 4.0,
  "paymentHistory": []
},
{
  "id": "23456",
  "borrower": "John Doe",
  "amount": 20000,
  "status": "Active",
  "interestRate": 3.0,
  "paymentHistory": ["On time"]
},
{
  "id": "34567",
  "borrower": "Alice Johnson",
  "amount": 40000,
```

```
    "status": "Closed",
    "interestRate": 3.8,
    "paymentHistory": ["On time", "On time"]
  }
]
\end{lstlisting}

\subsection{Step 13: Create MCP Server}
\textbf{File}: \texttt{server/server.js}
\begin{lstlisting}[language=javascript]
const express = require('express');
const cors = require('cors');
const loans = require('./loans.json');

const app = express();
app.use(cors());
app.use(express.json());

app.get('/api/loans', (req, res) => {
  res.json(loans);
```

```
});

app.get('/api/loan/:id', (req, res) => {
  const loan = loans.find((l) => l.id === req.params.id);
  if (loan) {
    res.json(loan);
  } else {
    res.status(404).json({ error: 'Loan not found' });
  }
});

app.get('/api/loan/status/:id', (req, res) => {
  const loan = loans.find((l) => l.id === req.params.id);
  if (loan) {
    res.json({ status: loan.status });
  } else {
    res.status(404).json({ error: 'Loan not found' });
  }
});
```

```
app.get('/api/loans/active', (req, res) => {
  const activeLoans = loans.filter((l) => l.status === 'Active');
  res.json(activeLoans);
});

app.get('/api/loans/borrower/:name', (req, res) => {
  const borrowerLoans = loans.filter(
    (l) => l.borrower.toLowerCase() === req.params.name.toLowerCase()
  );
  res.json(borrowerLoans);
});

const PORT = 3001;
app.listen(PORT, () => {
  console.log(`MCP server running on port ${PORT}`);
});
```

\subsection{Step 14: Document Client}

\textbf{File}: \texttt{client/README.md}

\begin{lstlisting}[language=markdown]

Loan Chatbot POC - Client

This is the React frontend for the Loan Officer Chatbot POC, featuring a dashboard and a sliding AI chatbot.

Prerequisites

- Node.js and npm installed

- MCP server running at `http://localhost:3001` (see `server/README.md`)

Setup

1. Navigate to the client directory:

   ```bash

   cd client

2. Install dependencies:

   bash

   Copy

   npm install

3. Start the React app:

   bash

   Copy
   ```

```
npm start
```

The app will open at http://localhost:3000.

Features

- **Dashboard**: Displays a table of loans (Loan ID, Borrower, Amount, Status).
- **Chatbot**: A sliding panel from the right, allowing queries like:
  - "status of loan 12345"
  - "details of loan 12345"
  - "active loans"
  - "loans for John Doe"

Notes

- The chatbot uses a rule-based AI for the POC. In production, consider integrating Dialogflow or Rasa for NLP.
- The MCP client (src/mcp/client.js) connects to http://localhost:3001/api. Update the baseURL for a real MCP server. \end{lstlisting}

\subsection{Step 15: Document Server} \textbf{File}: \texttt{server/README.md} \begin{lstlisting}[language=markdown]

Loan Chatbot POC - MCP Server

This is the Node.js/Express server simulating an MCP server, interacting with a fake JSON database.

Prerequisites

- Node.js and npm installed

Setup

4. Navigate to the server directory:

   bash

   Copy

   ```
   cd server
   ```

5. Install dependencies:

bash

Copy

```
npm install
```

6. Start the server:

   bash

   Copy

   ```
   npm start
   ```

   The server will run at http://localhost:3001.

Endpoints

- GET /api/loans: Get all loans
- GET /api/loan/:id: Get loan details by ID
- GET /api/loan/status/:id: Get loan status by ID
- GET /api/loans/active: Get all active loans
- GET /api/loans/borrower/:name: Get loans by borrower name

Notes

- Uses loans.json as a fake database.
- In production, replace with real MCP endpoints connecting to the Loan Origination and Accounting Systems. \end{lstlisting}

\subsection{Step 16: Document Entire Project} \textbf{File}: \texttt{README.md} \begin{lstlisting}[language=markdown]

Loan Chatbot Proof of Concept

This POC demonstrates an AI chatbot interface for Loan Officers, with a dashboard and a sliding chatbot panel.

Structure

- client/: React frontend with dashboard and chatbot.
- server/: Node.js/Express MCP server with a fake JSON database.

Setup

7. **Start the MCP server**:

   bash

   Copy

   ```
   cd server

   npm install

   npm start
   ```

8. **Start the client**:

   bash

   Copy

   ```
   cd client

   npm install

   npm start
   ```

9. Open http://localhost:3000 in a browser.

Testing

- View the dashboard to see the loan table.
- Click "Chat with AI" to open the chatbot.
- Try queries like:
  - "status of loan 12345"
  - "details of loan 12345"
  - "active loans"
  - "loans for John Doe"

Notes

- This is a POC with a rule-based AI. For production, integrate an NLP service like Dialogflow.

- The MCP client connects to a mock server. Update client/src/mcp/client.js for real MCP endpoints. \end{lstlisting}

\subsection{Step 17: Test the Application} \begin{lstlisting}[language=bash]

In one terminal

cd server npm start

In another terminal

cd client npm start \end{lstlisting} \begin{itemize} \item Open \url{http://localhost:3000}. \item Verify the dashboard displays the loan table. \item Click "Chat with AI" to open the slider. \item Test queries: \begin{itemize} \item "status of loan 12345" → Should return "The status of loan 12345 is Active." \item "details of loan 67890" → Should return loan details. \item "active loans" → Should list active loan IDs. \item "loans for John Doe" → Should list John Doe's loans. \end{itemize} \end{itemize}

\subsection{Step 18: Add Scalability Comments} \begin{itemize} \item In \texttt{client/src/mcp/client.js}: \begin{lstlisting}[language=javascript] // In production, update baseURL to real MCP endpoint (e.g., https://your-mcp-api.com) baseURL: 'http://localhost:3001/api', \end{lstlisting} \item In \texttt{client/src/components/Chatbot.js}: \begin{lstlisting}[language=javascript] // In production, replace rule-based parsing with an NLP service like Dialogflow or Rasa const parseMessage = (message) => { \end{lstlisting} \end{itemize}

\subsection{Step 19: Verify Error Handling} The chatbot handles errors: \begin{itemize} \item Invalid loan IDs return "Error fetching data." \item Unknown queries return a helpful message with examples. \end{itemize}

\section{Action List} \begin{enumerate} \item Create project folder structure. \item Initialize client (React) project. \item Install client dependencies. \item Configure Tailwind CSS. \item Create MCP client module. \item Create dashboard component. \item Create chatbot component. \item Update main app component. \item Update index file. \item Create public index HTML. \item Initialize MCP server. \item Create fake database. \item Create MCP server. \item Document client. \item Document server. \item Document entire project. \item Test the application. \item Add scalability comments. \item Verify error handling. \end{enumerate}

\section{Additional Notes} \begin{itemize} \item The chatbot uses a rule-based AI for simplicity. For production, integrate \href{https://cloud.google.com/dialogflow}{Dialogflow} or \href{https://rasa.com}{Rasa} for NLP. \item The MCP server is a mock. In production,

connect to the real MCP infrastructure. \item Ensure the server is running before starting the client. \item Test thoroughly to validate all queries and error handling. \end{itemize}

\end{document}

- **Purpose**: Provides a comprehensive guide for Cursor to build the POC, exportable as a PDF using latexmk.

Step 21: Compile the LaTeX Document

- **Action**: Generate the PDF from the LaTeX file.
- **Commands**:

bash

Copy

```
latexmk -pdf Loan_Chatbot_POC_Plan.tex
```

- **Purpose**: Creates Loan_Chatbot_POC_Plan.pdf for sharing with the team.

Additional Considerations

- **AI Recommendation**: The rule-based AI in Chatbot.js is sufficient for the POC, handling queries like "status of loan 12345" or "loans for John Doe". For production, integrate Dialogflow or Rasa for advanced NLP, which can be added by replacing the parseMessage function with an API call to the NLP service.
- **MCP Integration**: The mcp/client.js module is designed to work with RESTful APIs. For a real MCP server, update the baseURL and add authentication (e.g., OAuth tokens) as needed.
- **Testing**: The test step (Step 17) includes specific queries to validate functionality. Add more test cases if needed (e.g., invalid inputs, edge cases).
- **Compliance**: For a POC, compliance (e.g., GDPR) isn't critical, but ensure audit logs or data security are planned for production.
- **Responsive Design**: Tailwind CSS classes (e.g., min-w-full, p-4) ensure basic responsiveness. Test on mobile devices if time permits.

Action List

1. Create project folder structure (loan-chatbot-poc, client, server, etc.).
2. Initialize client (React) project with npx create-react-app.

3. Install client dependencies (axios, react-sliding-pane, tailwindcss, etc.).
4. Configure Tailwind CSS (tailwind.config.js, postcss.config.js, index.css).
5. Create MCP client module (client/src/mcp/client.js).
6. Create dashboard component (client/src/components/Dashboard.js).
7. Create chatbot component (client/src/components/Chatbot.js).
8. Update main app component (client/src/App.js).
9. Update index file (client/src/index.js).
10. Create public index HTML (client/public/index.html).
11. Initialize MCP server (server/package.json).
12. Create fake database (server/loans.json).
13. Create MCP server (server/server.js).
14. Document client (client/README.md).
15. Document server (server/README.md).
16. Document entire project (README.md).
17. Test the application with server and client running.
18. Add scalability comments to client.js and Chatbot.js.
19. Verify error handling in the chatbot.
20. Create LaTeX document (Loan_Chatbot_POC_Plan.tex).
21. Compile LaTeX document to PDF using latexmk.

Conclusion

This plan provides a detailed, step-by-step guide for Cursor to build the Loan Chatbot POC, including all files, commands, and documentation. The LaTeX document can be compiled into a PDF for sharing with your team, ensuring every step is clear and actionable. The POC meets all requirements: a pseudo browser interface, a sliding AI chatbot with MCP client integration, an MCP server, and a fake database. Let me know if you need further refinements! GO GO GO!