Todo Summary Assistant Implementation Guide

Project Overview

The Todo Summary Assistant is a full-stack application that allows users to:

1. Create and manage personal to-do items
2. Generate summaries of pending to-dos using a real LLM API
3. Send the generated summaries to a Slack channel

Tech Stack

Frontend

- React
- React Router for navigation
- Axios for API requests
- Context API or Redux for state management
- CSS framework (Tailwind CSS recommended)

Backend

- Node.js with Express.js
- Middleware: CORS, body-parser, express-validator
- Database: PostgreSQL via Supabase
- External APIs: LLM provider (OpenAI, Anthropic, etc.), Slack webhooks

Implementation Timeline

Phase 1: Setup and Configuration (Days 1-2)

Frontend Setup

```bash
# Create a new React application
npm create vite@latest todo-summary-frontend -- --template react
cd todo-summary-frontend

# Install dependencies
npm install axios react-router-dom react-toastify
```

Project structure:

```
src/
├── components/       # Reusable UI components
├── pages/            # Main page components
├── api/              # API client functions
├── context/          # React context for state management
├── utils/            # Helper functions
└── App.jsx           # Main application component
```

Backend Setup

bash

```bash
# Initialize a Node.js project
mkdir todo-summary-backend
cd todo-summary-backend
npm init -y

# Install dependencies
npm install express cors dotenv pg openai @slack/webhook
npm install --save-dev nodemon
```

Project structure:

```
src/
├── models/           # Data models
├── routes/           # API endpoints
├── controllers/      # Route handlers
├── services/         # Business logic
├── config/           # Configuration files
└── index.js          # Entry point
```

Database Setup (Supabase)

1. Create a Supabase account at https://supabase.com
2. Create a new project
3. Set up a todos table with the following schema:

```sql
CREATE TABLE todos (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  title TEXT NOT NULL,
  description TEXT,
  completed BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

Environment Setup

Create `.env` files for frontend and backend:

Backend `.env`:

```
PORT=5000
SUPABASE_URL=your_supabase_url
SUPABASE_KEY=your_supabase_key
OPENAI_API_KEY=your_openai_api_key
SLACK_WEBHOOK_URL=your_slack_webhook_url
```

Frontend `.env`:

```
VITE_API_URL=http://localhost:5000/api
```

Phase 2: Core Todo Functionality (Days 3-4)

Backend Implementation

1. Set up database connection (`config/db.js`):

```javascript
const { createClient } = require('@supabase/supabase-js');

const supabaseUrl = process.env.SUPABASE_URL;
const supabaseKey = process.env.SUPABASE_KEY;

const supabase = createClient(supabaseUrl, supabaseKey);

module.exports = supabase;
```

2. Create Todo model (`models/Todo.js`):

javascript

```
const supabase = require('../config/db');

class Todo {
  static async getAll() {
    const { data, error } = await supabase
      .from('todos')
      .select('*')
      .order('created_at', { ascending: false });

    if (error) throw error;
    return data;
  }

  static async create(todoData) {
    const { data, error } = await supabase
      .from('todos')
      .insert([todoData])
      .select();

    if (error) throw error;
    return data[0];
  }

  static async update(id, todoData) {
    const { data, error } = await supabase
      .from('todos')
      .update({ ...todoData, updated_at: new Date() })
      .eq('id', id)
      .select();

    if (error) throw error;
    return data[0];
  }

  static async delete(id) {
    const { error } = await supabase
      .from('todos')
      .delete()
      .eq('id', id);

    if (error) throw error;
    return true;
  }
}
```

```
module.exports = Todo;
```

3. Implement Todo routes (`routes/todoRoutes.js`):

javascript

```
const express = require('express');
const router = express.Router();
const todoController = require('../controllers/todoController');

router.get('/', todoController.getAllTodos);
router.post('/', todoController.createTodo);
router.put('/:id', todoController.updateTodo);
router.delete('/:id', todoController.deleteTodo);

module.exports = router;
```

4. Implement Todo controller (`controllers/todoController.js`):

javascript

```
const Todo = require('../models/Todo');

exports.getAllTodos = async (req, res) => {
  try {
    const todos = await Todo.getAll();
    res.json(todos);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

exports.createTodo = async (req, res) => {
  try {
    const { title, description } = req.body;

    if (!title) {
      return res.status(400).json({ error: 'Title is required' });
    }

    const todo = await Todo.create({ title, description });
    res.status(201).json(todo);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

exports.updateTodo = async (req, res) => {
  try {
    const { id } = req.params;
    const updatedTodo = await Todo.update(id, req.body);
    res.json(updatedTodo);
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};

exports.deleteTodo = async (req, res) => {
  try {
    const { id } = req.params;
    await Todo.delete(id);
    res.json({ message: 'Todo deleted successfully' });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
};
```

5. Set up Express server (`index.js`):

```javascript
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');

// Load environment variables
dotenv.config();

// Import routes
const todoRoutes = require('./routes/todoRoutes');
const summaryRoutes = require('./routes/summaryRoutes');

// Initialize app
const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(cors());
app.use(express.json());

// Routes
app.use('/api/todos', todoRoutes);
app.use('/api/summarize', summaryRoutes);

// Start server
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Frontend Implementation

1. Create API service (`api/todoApi.js`):

javascript

```javascript
import axios from 'axios';

const API_URL = import.meta.env.VITE_API_URL;

export const getAllTodos = async () => {
  const response = await axios.get(`${API_URL}/todos`);
  return response.data;
};

export const createTodo = async (todoData) => {
  const response = await axios.post(`${API_URL}/todos`, todoData);
  return response.data;
};

export const updateTodo = async (id, todoData) => {
  const response = await axios.put(`${API_URL}/todos/${id}`, todoData);
  return response.data;
};

export const deleteTodo = async (id) => {
  const response = await axios.delete(`${API_URL}/todos/${id}`);
  return response.data;
};
```

2. Create Todo context (`context/TodoContext.jsx`):

javascript

```jsx
import React, { createContext, useState, useEffect, useContext } from 'react';
import { getAllTodos, createTodo, updateTodo, deleteTodo } from '../api/todoApi';
import { toast } from 'react-toastify';

const TodoContext = createContext();

export const useTodos = () => useContext(TodoContext);

export const TodoProvider = ({ children }) => {
  const [todos, setTodos] = useState([]);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    fetchTodos();
  }, []);

  const fetchTodos = async () => {
    try {
      setLoading(true);
      const data = await getAllTodos();
      setTodos(data);
    } catch (error) {
      toast.error('Failed to fetch todos');
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  const addTodo = async (todoData) => {
    try {
      const newTodo = await createTodo(todoData);
      setTodos([newTodo, ...todos]);
      toast.success('Todo added successfully');
      return newTodo;
    } catch (error) {
      toast.error('Failed to add todo');
      console.error(error);
    }
  };

  const editTodo = async (id, todoData) => {
    try {
      const updatedTodo = await updateTodo(id, todoData);
      setTodos(todos.map(todo => todo.id === id ? updatedTodo : todo));
      toast.success('Todo updated successfully');
```

```
      return updatedTodo;
    } catch (error) {
      toast.error('Failed to update todo');
      console.error(error);
    }
  };

  const removeTodo = async (id) => {
    try {
      await deleteTodo(id);
      setTodos(todos.filter(todo => todo.id !== id));
      toast.success('Todo deleted successfully');
    } catch (error) {
      toast.error('Failed to delete todo');
      console.error(error);
    }
  };

  const toggleComplete = async (id, completed) => {
    return editTodo(id, { completed: !completed });
  };

  return (
    <TodoContext.Provider value={{
      todos,
      loading,
      addTodo,
      editTodo,
      removeTodo,
      toggleComplete,
      fetchTodos
    }}>
      {children}
    </TodoContext.Provider>
  );
};
```

3. Create Todo components:

`components/TodoForm.jsx`:

jsx

```jsx
import React, { useState } from 'react';
import { useTodos } from '../context/TodoContext';

const TodoForm = () => {
  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const { addTodo } = useTodos();

  const handleSubmit = async (e) => {
    e.preventDefault();

    if (!title.trim()) return;

    await addTodo({ title, description });
    setTitle('');
    setDescription('');
  };

  return (
    <form onSubmit={handleSubmit} className="mb-6">
      <div className="mb-4">
        <input
          type="text"
          placeholder="Todo title"
          value={title}
          onChange={(e) => setTitle(e.target.value)}
          className="w-full p-2 border rounded"
          required
        />
      </div>
      <div className="mb-4">
        <textarea
          placeholder="Description (optional)"
          value={description}
          onChange={(e) => setDescription(e.target.value)}
          className="w-full p-2 border rounded"
          rows="3"
        />
      </div>
      <button
        type="submit"
        className="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600"
      >
        Add Todo
      </button>
    </form>
```

```
    );
  };

  export default TodoForm;
```

`components/TodoItem.jsx`:

jsx

```jsx
import React from 'react';
import { useTodos } from '../context/TodoContext';

const TodoItem = ({ todo }) => {
  const { toggleComplete, removeTodo } = useTodos();

  return (
    <div className={`border p-4 mb-2 rounded ${todo.completed ? 'bg-gray-100' : ''}`}>
      <div className="flex justify-between">
        <div>
          <h3 className={`text-lg font-semibold ${todo.completed ? 'line-through text-gray-500'
            {todo.title}
          </h3>
          {todo.description && (
            <p className={`text-gray-700 mt-1 ${todo.completed ? 'text-gray-400' : ''}`}>
              {todo.description}
            </p>
          )}
        </div>
        <div className="flex items-start space-x-2">
          <button
            onClick={() => toggleComplete(todo.id, todo.completed)}
            className={`px-2 py-1 rounded ${todo.completed ? 'bg-yellow-500' : 'bg-green-500'}
          >
            {todo.completed ? 'Undo' : 'Complete'}
          </button>
          <button
            onClick={() => removeTodo(todo.id)}
            className="px-2 py-1 bg-red-500 text-white rounded"
          >
            Delete
          </button>
        </div>
      </div>
    </div>
  );
};

export default TodoItem;
```

components/TodoList.jsx:

jsx

```jsx
import React from 'react';
import { useTodos } from '../context/TodoContext';
import TodoItem from './TodoItem';

const TodoList = () => {
  const { todos, loading } = useTodos();

  if (loading) {
    return <div className="text-center py-4">Loading todos...</div>;
  }

  if (todos.length === 0) {
    return <div className="text-center py-4">No todos yet. Add some!</div>;
  }

  return (
    <div>
      {todos.map(todo => (
        <TodoItem key={todo.id} todo={todo} />
      ))}
    </div>
  );
};

export default TodoList;
```

4. Create main dashboard page (`pages/Dashboard.jsx`):

jsx

```jsx
import React from 'react';
import TodoForm from '../components/TodoForm';
import TodoList from '../components/TodoList';
import SummaryButton from '../components/SummaryButton';

const Dashboard = () => {
  return (
    <div className="container mx-auto px-4 py-8">
      <h1 className="text-3xl font-bold mb-6">Todo Summary Assistant</h1>

      <div className="bg-white p-6 rounded-lg shadow-md mb-6">
        <h2 className="text-xl font-semibold mb-4">Add New Todo</h2>
        <TodoForm />
      </div>

      <div className="mb-6">
        <SummaryButton />
      </div>

      <div className="bg-white p-6 rounded-lg shadow-md">
        <h2 className="text-xl font-semibold mb-4">Your Todos</h2>
        <TodoList />
      </div>
    </div>
  );
};

export default Dashboard;
```

5. Set up main App component (`App.jsx`):

jsx

```jsx
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import { TodoProvider } from './context/TodoContext';
import Dashboard from './pages/Dashboard';

function App() {
  return (
    <Router>
      <TodoProvider>
        <Routes>
          <Route path="/" element={<Dashboard />} />
        </Routes>
        <ToastContainer position="bottom-right" />
      </TodoProvider>
    </Router>
  );
}

export default App;
```

Phase 3: LLM Integration (Days 5-6)

Backend Implementation

1. Create LLM service (`services/llmService.js`):

javascript

```javascript
const { Configuration, OpenAIApi } = require('openai');

const configuration = new Configuration({
  apiKey: process.env.OPENAI_API_KEY,
});

const openai = new OpenAIApi(configuration);

const generateSummary = async (todos) => {
  try {
    // Filter for incomplete todos
    const pendingTodos = todos.filter(todo => !todo.completed);

    if (pendingTodos.length === 0) {
      return "You have no pending todos. Great job!";
    }

    const prompt = `
      Below is a list of pending tasks:
      ${pendingTodos.map(todo => `- ${todo.title}${todo.description ? `: ${todo.description}` :

      Please provide a concise summary of these tasks. Group similar tasks if possible,
      identify priorities, and suggest an efficient order for completing them.
      The summary should be well-structured and easy to read.
    `;

    const response = await openai.createCompletion({
      model: "gpt-3.5-turbo-instruct",
      prompt: prompt,
      max_tokens: 500,
      temperature: 0.7,
    });

    return response.data.choices[0].text.trim();
  } catch (error) {
    console.error('Error generating summary:', error);
    throw new Error('Failed to generate summary');
  }
};

module.exports = { generateSummary };
```

2. Create Slack service (`services/slackService.js`):

javascript

```
const { IncomingWebhook } = require('@slack/webhook');

// Initialize webhook
const webhook = new IncomingWebhook(process.env.SLACK_WEBHOOK_URL);

const sendToSlack = async (summary) => {
  try {
    await webhook.send({
      text: "📋 *Todo Summary*",
      blocks: [
        {
          type: "section",
          text: {
            type: "mrkdwn",
            text: "*📋 Todo Summary*"
          }
        },
        {
          type: "divider"
        },
        {
          type: "section",
          text: {
            type: "mrkdwn",
            text: summary
          }
        },
        {
          type: "context",
          elements: [
            {
              type: "mrkdwn",
              text: `*Generated:* ${new Date().toLocaleString()}`
            }
          ]
        }
      ]
    });
    return true;
  } catch (error) {
    console.error('Error sending to Slack:', error);
    throw new Error('Failed to send to Slack');
  }
};
```

```
module.exports = { sendToSlack };
```

3. Create summary routes (`routes/summaryRoutes.js`):

```javascript
const express = require('express');
const router = express.Router();
const summaryController = require('../controllers/summaryController');

router.post('/', summaryController.summarizeAndSend);

module.exports = router;
```

4. Create summary controller (`controllers/summaryController.js`):

javascript

```javascript
const Todo = require('../models/Todo');
const { generateSummary } = require('../services/llmService');
const { sendToSlack } = require('../services/slackService');

exports.summarizeAndSend = async (req, res) => {
  try {
    // Get all todos
    const todos = await Todo.getAll();

    // Generate summary using LLM
    const summary = await generateSummary(todos);

    // Send to Slack
    await sendToSlack(summary);

    res.json({
      success: true,
      message: 'Summary generated and sent to Slack successfully',
      summary
    });
  } catch (error) {
    console.error('Summary error:', error);
    res.status(500).json({
      success: false,
      error: error.message
    });
  }
};
```

Frontend Implementation

1. Create summary API service (`api/summaryApi.js`):

javascript

```javascript
import axios from 'axios';

const API_URL = import.meta.env.VITE_API_URL;

export const generateAndSendSummary = async () => {
  const response = await axios.post(`${API_URL}/summarize`);
  return response.data;
};
```

2. Create summary button component (`components/SummaryButton.jsx`):

jsx

```jsx
import React, { useState } from 'react';
import { generateAndSendSummary } from '../api/summaryApi';
import { toast } from 'react-toastify';

const SummaryButton = () => {
  const [loading, setLoading] = useState(false);
  const [summary, setSummary] = useState('');
  const [showSummary, setShowSummary] = useState(false);

  const handleGenerateSummary = async () => {
    try {
      setLoading(true);
      const result = await generateAndSendSummary();

      if (result.success) {
        setSummary(result.summary);
        setShowSummary(true);
        toast.success('Summary sent to Slack successfully!');
      } else {
        toast.error('Failed to generate summary');
      }
    } catch (error) {
      toast.error(error.response?.data?.error || 'An error occurred');
      console.error(error);
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="bg-white p-6 rounded-lg shadow-md">
      <div className="flex items-center justify-between mb-4">
        <h2 className="text-xl font-semibold">Todo Summary</h2>
        <button
          onClick={handleGenerateSummary}
          disabled={loading}
          className="bg-purple-600 text-white px-4 py-2 rounded hover:bg-purple-700 disabled:bg
        >
          {loading ? 'Generating...' : 'Generate & Send to Slack'}
        </button>
      </div>

      {showSummary && (
        <div className="mt-4">
          <h3 className="font-medium mb-2">Generated Summary:</h3>
          <div className="bg-gray-100 p-4 rounded whitespace-pre-wrap">
```

```
            {summary}
          </div>
        </div>
      )}
    </div>
  );
};

export default SummaryButton;
```

Phase 4: Setup Instructions & Documentation

Slack Webhook Setup Instructions

1. Go to your Slack workspace
2. Go to https://api.slack.com/apps
3. Click "Create New App" and choose "From scratch"
4. Give your app a name and select your workspace
5. Click on "Incoming Webhooks" in the sidebar
6. Toggle "Activate Incoming Webhooks" to On
7. Click "Add New Webhook to Workspace"
8. Choose the channel where you want to receive summaries
9. Copy the Webhook URL provided and add it to your .env file

OpenAI API Setup Instructions

1. Go to https://platform.openai.com/
2. Create an account or login
3. Navigate to the API keys section
4. Create a new API key
5. Copy the key and add it to your .env file

Project Setup Instructions

1. Clone the repository

```bash
git clone <repository-url>
cd todo-summary-app
```

2. Set up the backend

bash

```
cd backend
npm install
# Create .env file with required variables
npm run dev
```

3. Set up the frontend

bash

```
cd ../frontend
npm install
# Create .env file with required variables
npm run dev
```

4. Open your browser and navigate to http://localhost:5173

Next Steps and Improvements

Authentication

- Implement user authentication using Supabase Auth
- Add user-specific todos

Enhanced UI

- Add dark mode support
- Create a mobile-responsive design
- Add animations for better user experience

Additional Features

- Add due dates to todos
- Implement todo categories/tags
- Create recurring todos
- Add ability to prioritize todos
- Implement drag-and-drop reordering

Deployment Options

- Frontend: Vercel, Netlify, or Firebase Hosting
- Backend: Vercel, Railway, or Render
- Database: Keep using Supabase

Architecture Decisions

Why Node.js for Backend?

Node.js with Express provides a lightweight and efficient backend solution that's well-suited for API development. It's also JavaScript-based, which allows for code sharing between frontend and backend.

Why React for Frontend?

React's component-based architecture makes it easy to build and maintain a complex UI. Its virtual DOM implementation ensures efficient updates and rendering.

Why Supabase for Database?

Supabase provides a PostgreSQL database with a convenient REST API, making it easy to integrate with both frontend and backend. It also offers authentication services if needed in the future.

Why OpenAI API for LLM?

OpenAI's GPT models provide high-quality text generation capabilities ideal for summarizing todos. The API is well-documented and reliable.

Why Slack Webhooks?

Slack's Incoming Webhooks provide a simple way to send messages to a Slack channel without complex OAuth flows, making it easy to integrate into our application.