I'll provide a step-by-step implementation plan for each team member's work on the Personal Life and Social Engagement Assistant project. Here's a detailed breakdown:

# Suraj's Implementation (AI Model + n8n Automation)

## Step 1: Set Up the AI Model

1. Install required packages:

bash Copy

```
pip install transformers torch flask
```

2. Create a Python script `model_server.py`:

python Copy

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

app = Flask(__name__)

# Load Falcon-7B model
task_pipe = pipeline("text-generation", model="tiiuae/falcon-7b-instruct")

def extract_structured_data(text):
    prompt = f"""
    Extract task information from this text: "{text}"
    Format the response as JSON with task, date, and time fields.
    """
    response = task_pipe(prompt, max_length=200, do_sample=True)
    # Basic parsing - in production would use regex or more robust parsing
    result_text = response[0]["generated_text"].split("{")[1].split("}")[0]
    lines = result_text.strip().split("\n")
    task_dict = {}
    for line in lines:
        if ":" in line:
            key, value = line.split(":", 1)
            task_dict[key.strip().strip('"')] = value.strip().strip('"').strip(',')
    return task_dict

@app.route('/process', methods=['POST'])
def process_task():
    user_input = request.json.get('task')
    structured_data = extract_structured_data(user_input)
    return jsonify(structured_data)
```

```
if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

3. Run the Flask server:

bash Copy

```
python model_server.py
```

## Step 2: Set Up n8n Workflow

1. Install n8n:

bash Copy

```
npm install n8n -g
```

2. Start n8n:

bash Copy

```
n8n start
```

3. Create a new workflow in n8n with these nodes:
   - **Webhook Node**:
     - Method: POST
     - Path: /task-process
     - Response Mode: Last Node
   - **HTTP Request Node** (for Google Sheets backup):
     - Method: POST
     - URL: https://sheets.googleapis.com/v4/spreadsheets/YOUR_SHEET_ID/values/A:D:append
     - Authentication: OAuth2
     - Body:

       json Copy

       ```
       {
         "values": [
           ["{{$json.id}}", "{{$json.task}}", "{{$json.date}}", "{{$json.time}}"]
         ]
       }
       ```

   - **Supabase Node**:
     - Operation: Insert
     - Table: tasks

- Columns mapping:
  - task: {{$json.task}}
  - date: {{$json.date}}
  - time: {{$json.time}}

- **Respond to Webhook Node**:
  - Response Code: 200
  - Response Data: {{$node["Supabase"].json}}

4. Connect the nodes in sequence and activate the workflow

# Kuldeep's Implementation (Database & Storage)

## Step 1: Set Up Supabase

1. Create account at supabase.com and create a new project
2. Note down Project URL and API Key
3. Create the tasks table in Supabase SQL Editor:

sql Copy

```sql
CREATE TABLE tasks (
    id SERIAL PRIMARY KEY,
    task TEXT NOT NULL,
    date TEXT,
    time TEXT,
    user_id TEXT,
    created_at TIMESTAMPTZ DEFAULT NOW()
);

-- Add indexes for faster queries
CREATE INDEX idx_user_id ON tasks(user_id);
CREATE INDEX idx_date ON tasks(date);
```

4. Create API access policies:

sql Copy

```sql
-- Enable row-level security
ALTER TABLE tasks ENABLE ROW LEVEL SECURITY;

-- Create policy for task access
CREATE POLICY "Users can view their own tasks"
ON tasks FOR SELECT
USING (auth.uid()::text = user_id);

-- Create policy for task insertion
CREATE POLICY "Users can insert their own tasks"
```

```
ON tasks FOR INSERT
WITH CHECK (auth.uid()::text = user_id);
```

## Step 2: Create API Integration Module

1. Create a file `supabase_connector.py`:

python Copy

```python
import os
from supabase import create_client

class SupabaseConnector:
    def __init__(self):
        self.url = os.getenv("SUPABASE_URL")
        self.key = os.getenv("SUPABASE_KEY")
        self.client = create_client(self.url, self.key)

    def store_task(self, task_data):
        """Store a task in Supabase"""
        response = self.client.table('tasks').insert(task_data).execute()
        return response.data

    def get_user_tasks(self, user_id):
        """Get all tasks for a specific user"""
        response = self.client.table('tasks')\
                    .select('*')\
                    .eq('user_id', user_id)\
                    .order('created_at', desc=True)\
                    .execute()
        return response.data
```

2. Create database testing script `test_db.py`:

python Copy

```python
from supabase_connector import SupabaseConnector
import os

# Set environment variables (in production use proper env management)
os.environ["SUPABASE_URL"] = "https://your-project-url.supabase.co"
os.environ["SUPABASE_KEY"] = "your-api-key"

# Test connection
connector = SupabaseConnector()

# Test inserting a task
```

```
test_task = {
    "task": "Buy groceries",
    "date": "Tomorrow",
    "time": "5 PM",
    "user_id": "test-user-123"
}

result = connector.store_task(test_task)
print(f"Task stored with ID: {result[0]['id']}")

# Test retrieving tasks
tasks = connector.get_user_tasks("test-user-123")
print(f"Retrieved {len(tasks)} tasks")
```

# Manav's Implementation (Frontend - Streamlit Chat UI)

## Step 1: Create Streamlit App

1. Install required packages:

bash Copy

```
pip install streamlit requests python-dotenv
```

2. Create `.env` file:

Copy

```
AI_MODEL_ENDPOINT=http://localhost:5000/process
SUPABASE_URL=https://your-project-url.supabase.co
SUPABASE_KEY=your-api-key
```

3. Create `app.py`:

python Copy

```
import streamlit as st
import requests
import os
import json
from datetime import datetime
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

# Import Supabase connector
import sys
```

```
sys.path.append('../') # Adjust as needed
from database.supabase_connector import SupabaseConnector

# Initialize Supabase
db = SupabaseConnector()

# Configure app
st.set_page_config(
    page_title="Personal Life Assistant",
    page_icon="🧠",
    layout="wide"
)

# Initialize session state
if 'tasks' not in st.session_state:
    st.session_state.tasks = []
if 'user_id' not in st.session_state:
    st.session_state.user_id = "user-" + datetime.now().strftime("%Y%m%d%H%M%S")

# App layout
st.title("🧠 Personal Life & Social Engagement Assistant")

with st.sidebar:
    st.header("Your Profile")
    st.write(f"User ID: {st.session_state.user_id}")
    if st.button("Load My Tasks"):
        st.session_state.tasks = db.get_user_tasks(st.session_state.user_id)

# Main area - two columns
col1, col2 = st.columns([3, 2])

with col1:
    st.subheader("Add New Task or Event")
    user_input = st.text_input("What would you like to schedule? (e.g., 'Remind me to c

    if st.button("Process"):
        with st.spinner("Processing your request..."):
            # Call AI model
            response = requests.post(
                os.getenv("AI_MODEL_ENDPOINT"),
                json={"task": user_input}
            )

            if response.status_code == 200:
                task_data = response.json()
                task_data["user_id"] = st.session_state.user_id
```

```
                # Display extracted information
                st.success("Task extracted successfully!")
                st.write("📌 Task: ", task_data.get("task", "Not specified"))
                st.write("🗓️ Date: ", task_data.get("date", "Not specified"))
                st.write("⏰ Time: ", task_data.get("time", "Not specified"))

                # Option to save
                if st.button("Save to My Calendar"):
                    result = db.store_task(task_data)
                    st.session_state.tasks = db.get_user_tasks(st.session_state.user_id
                    st.success(f"Task saved! ID: {result[0]['id']}")
            else:
                st.error(f"Error: {response.status_code} - {response.text}")

with col2:
    st.subheader("Your Upcoming Tasks")
    if not st.session_state.tasks:
        st.info("No tasks found. Add some tasks to see them here!")
    else:
        for task in st.session_state.tasks:
            with st.expander(f"{task['task']} ({task['date']})"):
                st.write(f"⏰ Time: {task['time']}")
                st.write(f"🆔 ID: {task['id']}")
                st.write(f"🗓️ Created: {task['created_at']}")
```

4. Run the app:

bash Copy

```
streamlit run app.py
```

# Integration Steps

1. **Connect All Components**:
   - Ensure Suraj's AI model is running on port 5000
   - Verify Kuldeep's Supabase connection is working
   - Start Manav's Streamlit interface
2. **Test End-to-End Flow**:
   - Input a task in Streamlit UI
   - Confirm AI model extracts structured data
   - Verify task stores in Supabase
   - Check task appears in UI
3. **Production Deployment**:
   - Deploy Flask API on a cloud service (Heroku/AWS)

- Host n8n workflow on a server
- Deploy Streamlit app publicly
- Update environment variables with production endpoints

This implementation follows the project requirements for creating a Personal Life and Social Engagement Assistant with AI-powered task understanding, database storage, and a user-friendly interface.