System Design Document

AI Study Planner Agent

Live Application: https://studyplanner-ai-agent.streamlit.app/

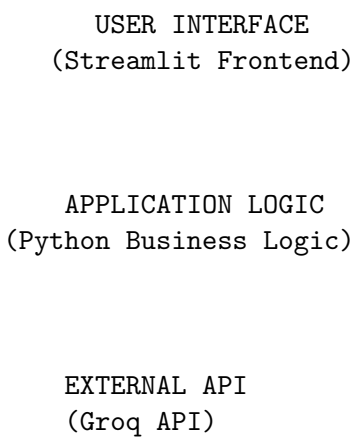
Author: Chodabattula Moni Praneeth

Contents

1 System Architecture

1.1 Overall Architecture

The AI Study Planner follows a simple 3-layer architecture:



1.2 Architecture Pattern

- **Pattern:** Monolithic Web Application
- **Reason:** Simple to develop, deploy, and maintain for a prototype. Single codebase with all functionality contained in one application.

1.3 Data Flow

User Input → Validation → API Request → AI Response → Display

2 Data Design

2.1 Data Models

2.1.1 Course Data Structure

```
{
    "course": str,         // Course name (e.g., "Mathematics")
    "assignment": str,     // Assignment description
    "deadline": str,       // Date in "YYYY-MM-DD" format
    "hours": int          // Estimated hours (1-100)
}
```

2.1.2 User Preferences

```
{
    "hours_per_day": int,     // Daily study hours (1-8)
    "study_style": str        // "Focused sessions" | "Spaced repetition" | "
        Mixed"
}
```

2.1.3 Session State Schema

```
{
    "study_plan": str,           // Generated AI plan (markdown text)
    "courses": List[Dict],       // List of course dictionaries
}
```

2.2 Data Storage

- **Method:** Streamlit Session State (in-memory)
- **Reason:** No persistent storage needed for prototype. Session-based data is sufficient and eliminates database complexity.
- **Data Persistence:** Only during browser session, automatically cleared when session ends

2.3 Data Processing

1. Input Collection: Forms collect course and preference data
2. Validation: Check required fields and data types
3. API Formatting: Convert data to text prompt for AI
4. Response Handling: Display AI response as formatted text

3 Component Breakdown

3.1 Frontend Components

3.1.1 User Interface (Streamlit)

Main UI components in the application:

1. **Application Header**
   - `st.set_page_config()` for page configuration
   - `st.title()` for main application title
   - Purpose: Application branding and configuration
2. **Course Input Form**
   - `st.columns(2)` layout with text inputs and date picker
   - `st.text_input()` for course name and assignment
   - `st.date_input()` for deadline selection
   - `st.number_input()` for hours estimation
   - Purpose: Collect comprehensive course information
3. **Course Management**
   - Add button with validation logic
   - Course list display with formatted output
   - Purpose: Add and view course data

4. **Study Preferences**
   - `st.slider()` for hours per day (1-8 range)
   - `st.selectbox()` for study style selection
   - Purpose: User customization of study parameters
5. **Plan Generation**
   - Primary button for triggering AI plan generation
   - Loading spinner during API processing
   - Purpose: Initiate study plan creation
6. **Plan Display**
   - `st.markdown()` for rich text display of AI output
   - Purpose: Show AI-generated study plan with formatting
7. **Export Feature**
   - `st.download_button()` for file download
   - Purpose: Save plan as text file for offline use
8. **Session Management**
   - Clear all button to reset application state
   - `st.rerun()` for state refresh
   - Purpose: Reset functionality and session control

3.2 Application Logic Components

3.2.1 Session State Management

```
# Initialize session state variables
if 'study_plan' not in st.session_state:
    st.session_state.study_plan = ""
if 'courses' not in st.session_state:
    st.session_state.courses = []
```

3.2.2 Input Validation

```
# Validation in button click handler:
if not st.session_state.courses:
    st.error("Please add at least one course")
# Course addition validation
if st.button("Add") and course and assignment:
    # Add course to session state
```

3.2.3 API Integration

```
def call_groq_api(prompt):
    # HTTP request to Groq API with hardcoded key
    # JSON response processing
    # Error handling for API failures and timeouts
    # Returns formatted study plan or None on error
```

3.2.4 Data Processing

```
# Convert courses to text format for AI prompt:
courses_text = ""
for c in st.session_state.courses:
    courses_text += f"- {c['course']}: {c['assignment']} (Due: {c['deadline']},
        {c['hours']} hours)\n"

# Comprehensive prompt construction with context
prompt = f"""Create a study plan for:
{courses_text}
Available time: {hours_per_day} hours/day
Style: {style}
Current date: {datetime.now().strftime("%Y-%m-%d")}
Provide a weekly schedule with daily tasks and time blocks."""
```

3.3 External Integration

3.3.1 Groq API Client

- **Purpose:** Send prompts to AI and receive study plans
- **Implementation:** HTTP POST requests using requests library with hardcoded API key
- **Model:** llama-3.1-8b-instant for fast inference
- **Error Handling:** Status code checking, timeout handling, and user feedback
- **Security:** API key embedded in code (hardcoded for simplicity)

4 Technology Choices & Justifications

4.1 Frontend: Streamlit

Choice: Streamlit Python framework
Reasons:

1. Development Speed: Build web UI with pure Python, no HTML/CSS/JavaScript needed
2. Built-in Components: Forms, buttons, file downloads included out-of-the-box
3. State Management: Automatic session state handling with st.session_state
4. Deployment: Free hosting on Streamlit Cloud with GitHub integration
5. Python Integration: No context switching between languages

Alternatives Considered:

- React: Too complex for prototype, requires separate backend
- Flask: Requires HTML templates and more setup

4.2 AI Service: Groq API

Choice: Groq API with LLaMA 3.1-8B-Instant model
Reasons:

1. Cost: Free tier available, no payment required for development
2. Speed: Fastest inference in the market (∼5 seconds response time)
3. Quality: Good text generation for planning tasks
4. API: Simple REST API, OpenAI-compatible format
5. Model: Current model (not deprecated), stable and reliable

Alternatives Considered:

- OpenAI GPT: Requires payment, quota exceeded during development
- Hugging Face: Slower inference, more complex setup
- Local LLM: Complex setup, hardware requirements

4.3 Deployment: Streamlit Cloud

Choice: Streamlit Cloud hosting
Reasons:

1. Cost: Completely free for public repositories
2. Integration: Direct GitHub integration, automatic deployments
3. Simplicity: One-click deployment, no server management
4. SSL: Automatic HTTPS and custom domains
5. Scaling: Automatic scaling handled by platform

Alternatives Considered:

- Heroku: Costs $5-7/month for hobby tier
- AWS/GCP: Too complex for simple prototype
- Local hosting: Not accessible for evaluation

4.4 Architecture: Monolithic

Choice: Single application architecture
Reasons:

1. Simplicity: All code in one file, easy to understand and modify
2. Deployment: Single deployment unit, no orchestration needed
3. Development Speed: Faster to build and test
4. Appropriate Scale: Perfect for prototype with limited users
5. No Network Latency: All components in same process

Alternatives Considered:

- Microservices: Overkill for prototype, adds unnecessary complexity
- Serverless: More complex deployment, not needed for current scale

4.5 Data Storage: Session State

Choice: Streamlit session state (in-memory)
Reasons:

1. Simplicity: No database setup or management required
2. Privacy: No persistent storage of user data
3. Performance: Fastest access, no database queries
4. Prototype Appropriate: Sufficient for demonstration purposes
5. Security: Data automatically cleared when session ends

Alternatives Considered:

- SQLite: Unnecessary complexity for temporary data
- Cloud Database: Overkill and costs money
- File Storage: Persistence not needed

5 Implementation Summary

5.1 What Was Built

- Single-page web application using Streamlit
- Course management system with add/display functionality
- AI integration with Groq API for plan generation
- Export feature for downloading generated plans
- Error handling for API failures and validation
- Live deployment at `https://studyplanner-ai-agent.streamlit.app/`

5.2 Key Features

1. User Input: Course details and study preferences
2. AI Processing: Automated study plan generation
3. Plan Display: Rich text formatting of AI output
4. File Export: Download plans as text files
5. Session Management: Temporary data storage during use

5.3 Technical Implementation

- Frontend: ∼100 lines of Python using Streamlit
- API Integration: HTTP requests to Groq API with embedded key
- Data Handling: Simple dictionaries and lists
- Deployment: Automatic via Streamlit Cloud
- Dependencies: Only streamlit and requests libraries

This system design demonstrates a practical, working solution that automates study planning using AI while maintaining simplicity and ease of use.