StudyPilot Project Design Document

Cevdet Onat Cerit – 231101078
Alvin Dora Akıncı – 231101052
Emin Arslan – 231101007
Nehir Tıraş – 231101065
Nehir Aydın – 231101053

Document-Specific Task Matrix:

Task	Team Member Responsible
System Overview	Cevdet Onat Cerit
Implementation Details	Alvin Dora Akıncı
Use Case Support in Design	Emin Arslan
Design Decisions	Nehir Tıraş, Nehir Aydın

1. System Overview

1.1 Project Description

StudyPilot is a productivity-oriented web application built for students to manage courses, assignments, exams, and study sessions efficiently.
By combining task management, a Pomodoro timer, music integration through Spotify, and real-time weather-based study recommendations, StudyPilot offers a personalized and motivating study environment.

Key features include:
- Task creation, editing, deletion, and categorization
- Smart prioritization algorithm
- Pomodoro timer with session customization
- Spotify playlist integration for focus sessions
- Weather API integration for personalized study suggestions
- Data-driven statistics and analytics dashboard

The system aims to help users track their productivity and establish consistent study habits. Success will be measured by user engagement, ease of use, accuracy of analytics, and reliability of integrated tools.

1.2 System Architecture

General Structure:
The system is developed using a **client-side modular architecture**, where all application logic runs in the browser while external services provide data.

Layers:
- **Presentation Layer (Frontend):** HTML, CSS, JavaScript-based UI
- **Service Layer:** Spotify API, Weather API
- **Data Layer:** Browser LocalStorage for persistence

Main Components:
- Dynamic user interface built with modular JavaScript components
- API service modules (SpotifyService, WeatherService)
- Logic modules (TaskManager, PomodoroModule, AnalyticsEngine)
- Local storage service for offline accessibility

Communication:
- REST API communication with Spotify & OpenWeatherMap
- JSON-based requests/responses

1.3 Technology Stack

- **Frontend:** HTML, CSS, JavaScript
- **APIs:** Spotify Web API, OpenWeatherMap API
- **Storage:** LocalStorage
- **Tools:** GitHub (Version Control), Figma (UI Design)

---

2. Implementation Details

2.1 Code Structure

The project follows a modular folder structure:

```
/src
  /components     → UI elements (TaskCard, TimerUI, DashboardWidgets)
  /modules        → Core logic (TaskManager, Timer, Prioritizer, Analytics)
  /services       → External APIs (spotifyService.js, weatherService.js)
  /assets         → Images, icons, UI graphics
  /styles         → CSS files
  /PA1Docs        → Project Definition, Plan, Requirements
  /PA2Docs        → Design Document, QA Plan
```

Modular Structure: Each module is developed independently and integrated through event-based interactions.

2.2 Main Components & Business Logic

● Task Management Module

Handles:
✓ Add/edit/delete tasks
✓ Categorization (course, assignment, exam)
✓ Priority calculation via Smart Prioritizer

● Smart Prioritization Engine

Factors:
✓ Deadline proximity
✓ Task type weight
✓ Estimated workload

Output: ordered task list.

● Pomodoro Timer

Functions:
✓ Start/pause/stop
✓ Custom session length
✓ Automatic break cycles
✓ Update analytics on session end

● **Spotify Integration**

Handles:
✓ Fetching playlists via Spotify Web API
✓ Playing focus/study music
✓ Managing authorization token

● **Weather Integration**

Handles:
✓ Fetch current weather
✓ Produce study suggestion text (e.g., “Cold weather → indoor focus recommended”)

● **Analytics Engine**

Tracks:
✓ Completed tasks
✓ Total Pomodoro time
✓ Weekly productivity stats
✓ Visual data output for dashboard

2.3 Visual Interfaces

Wireframes from PA1 will be used as the foundation. Main pages include:

● **Dashboard:** Weather, quick stats, daily recommendations
● **Task Page:** Task list, prioritization indicators
● **Pomodoro Timer Page:** Timer UI, session logs
● **Analytics Page:** Charts displaying weekly and monthly progress

---

3. Use Case Support

3.1 Selected Use Cases (5 Required Use Cases)

StudyPilot supports **five primary use cases**, aligned with PA2 expectations:

Use Case 1: Task Creation

User creates, edits, deletes tasks.

Use Case 2: Pomodoro Study Session

User starts, pauses, and stops Pomodoro sessions.

Use Case 3: Spotify Focus Playlist Playback

System retrieves playlists and plays recommended study music.

Use Case 4: Smart Task Prioritization

System sorts tasks using prioritization algorithms.

Use Case 5: Weather-Based Study Recommendation

System fetches weather data and generates dynamic study recommendations.

3.2 Requirement Mapping

Use Case	Requirement
Task Management	Functional Requirement 1
Smart Prioritization	Functional Requirement 2
Pomodoro Timer	Functional Requirement 3
Spotify Integration	Functional Requirement 4

Weather-Based Recommendation	Functional Requirement 5

3.3 Use Case Design

External Services & APIs

1. Spotify Web API

Used for:

- Retrieving playlists
- Fetching track recommendations
- Handling OAuth authorization

Key Endpoints:

- GET /v1/playlists
- GET /v1/recommendations
- Token refresh every 3600 seconds

2. OpenWeatherMap API

Used for:

- Fetching real-time weather
- Creating study suggestions

Endpoints:

- GET /weather?city=Ankara

3. Browser LocalStorage

Stores:

- Tasks
- Timer logs
- Completed Pomodoro sessions
- Dark mode preferences
- Analytics statistics

Data Flow Explanation

1. **User Input**
    - Creates task, starts timer, or selects playlist
    - Sent to relevant module
2. **Module Processing**
    - TaskManager saves task
    - Prioritizer recalculates priority
    - TimerModule updates session logs
    - SpotifyService fetches playlists
    - WeatherService returns weather & suggestions
3. **LocalStorage**
    - Stores persistent data
    - Enables offline access

4. **UI Rendering**
    - Task list updates dynamically
    - Timer state displayed
    - Playlist rendered
    - Dashboard suggestions updated
5. **External APIs Return Data**
    - Weather data → Dashboard
    - Spotify playlists → Music page

4. Design Decisions

4.1 Technology Comparisons

LocalStorage vs Firebase

Option	Pros	Cons
LocalStorage	Fast, offline, simple, no backend required	Limited storage, no sync
Firebase	Real-time, cloud sync	Requires backend & auth complexity

→ LocalStorage chosen for PA2 simplicity.

Vanilla JavaScript vs React

Option	Pros	Cons
Vanilla JS	Lightweight, simple, fast, no build process	Harder to scale
React	Component-based, great ecosystem	Requires bundler, adds complexity

→ Vanilla JS chosen for lightweight PA2 scope.

Spotify Web API vs YouTube Music API

→ Spotify chosen because:
✓ Better focus playlists
✓ Strong developer documentation
✓ Easier token management

4.2 Decision Rationale

The chosen technologies minimize complexity, reduce dependency on backend services, and ensure the application remains lightweight and fast.
This approach aligns with PA2 constraints and provides a smooth user experience.