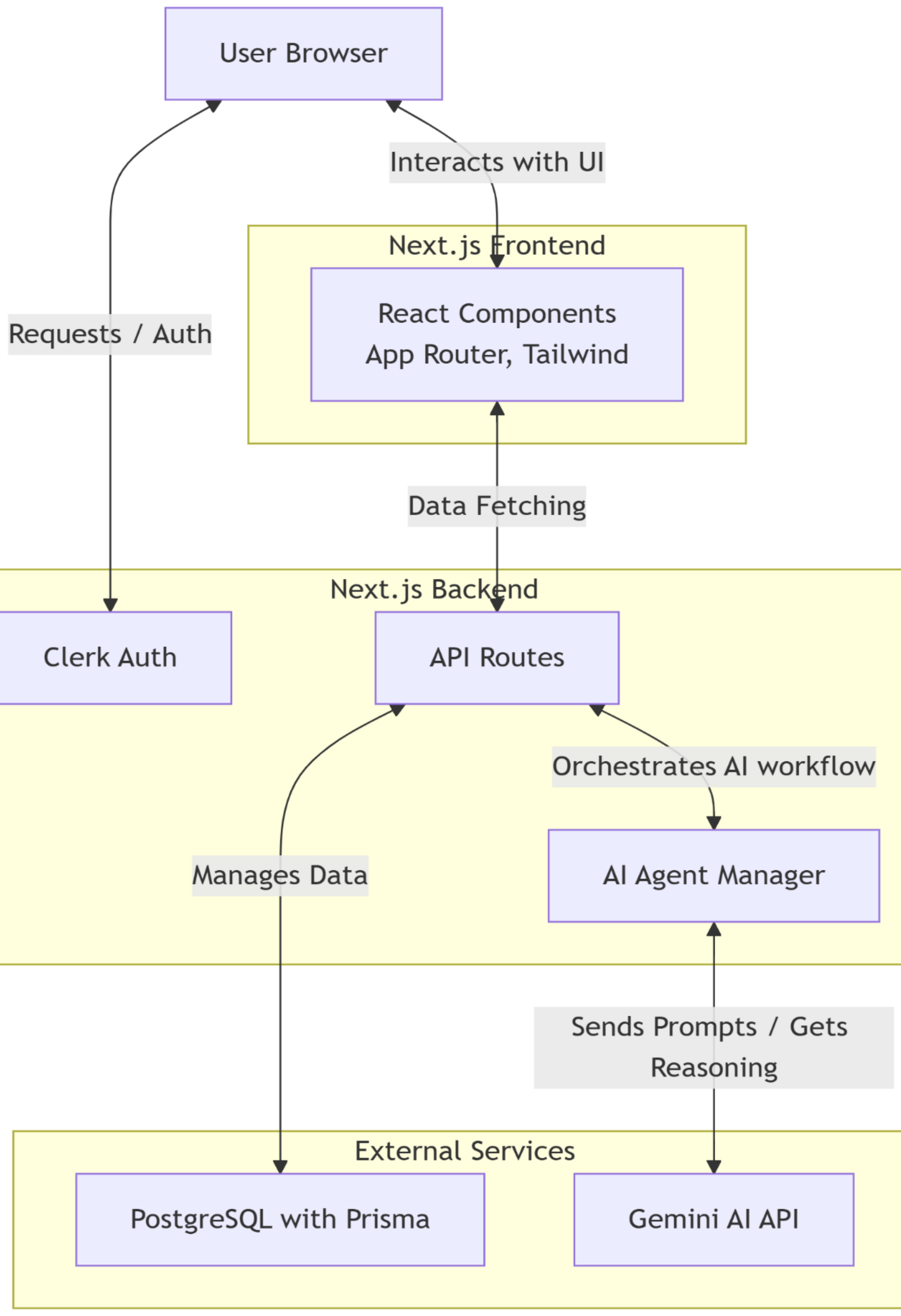
SYSTEM DESIGN DOCUMENT

ARCHITECTURE:

The system follows a full-stack Next.js application architecture with integrated AI services as shown below.



The **Study Buddy** system follows a modular client–server architecture built with **Next.js**. The architecture integrates user-facing interfaces, backend API routes, authentication, database management, and external AI services.

Key Layers

1. **User Browser**
    - Acts as the entry point for the system.
    - Users interact with the application UI, upload study material, and view generated summaries.
2. **Frontend (Next.js Frontend)**
    - Built with **React components, App Router, and Tailwind CSS** for responsive UI.
    - Handles user input, renders summaries, and displays analytics.
    - Fetches data securely from backend API routes.
3. **Backend (Next.js Backend)**
    - Implements **API routes** that act as intermediaries between frontend and services.
    - Integrates **Clerk Authentication** for secure user login and session management.
    - Contains an **AI Agent Manager** that orchestrates summarization workflows.
    - Manages data flow between frontend, database, and external APIs.
4. **External Services**
    - **PostgreSQL with Prisma:** Stores user accounts, uploaded files, summaries, and progress. Prisma ORM ensures type-safe queries and schema management.
    - **Gemini AI API (Google Generative AI):** Processes user-provided text or documents to generate summaries and reasoning outputs.

DATA DESIGN:

Data Inputs & Outputs

- Inputs (from user):
    - Authentication details → handled by Clerk (email, password, OAuth).
    - Notes → text or PDF.
    - Preferences for task → e.g., quiz difficulty, number of questions, type of flashcards.
    - Scheduling info → due date, priority level.
- Outputs (to user):
    - Summary (text)
    - Quiz (Q/A pairs)
    - Flashcards (text cards)
    - Mindmap (hierarchical/nested JSON → rendered visually)
    - Saved history of notes/tasks (for dashboard & search)

Database Schema

PK: Primary Key , FK: Functional Key

User

- user_id (PK)
- username
- email
- password_hash
- created_at

Note

- note_id (PK)
- user_id (FK)

- title
- content
- created_at
- updated_at

Flashcard

- flashcard_id (PK)
- user_id (FK)
- question
- answer
- created_at
- note_id (FK, optional, if linked to a note)

Quiz

- quiz_id (PK)
- user_id (FK)
- title
- created_at

QuizQuestion

- question_id (PK)
- quiz_id (FK)
- question_text
- answer
- options (array or JSON, if multiple choice)

Mindmap

- mindmap_id (PK)
- user_id (FK)
- title
- structure (JSON or text for nodes/edges)
- created_at

SummarizationHistory

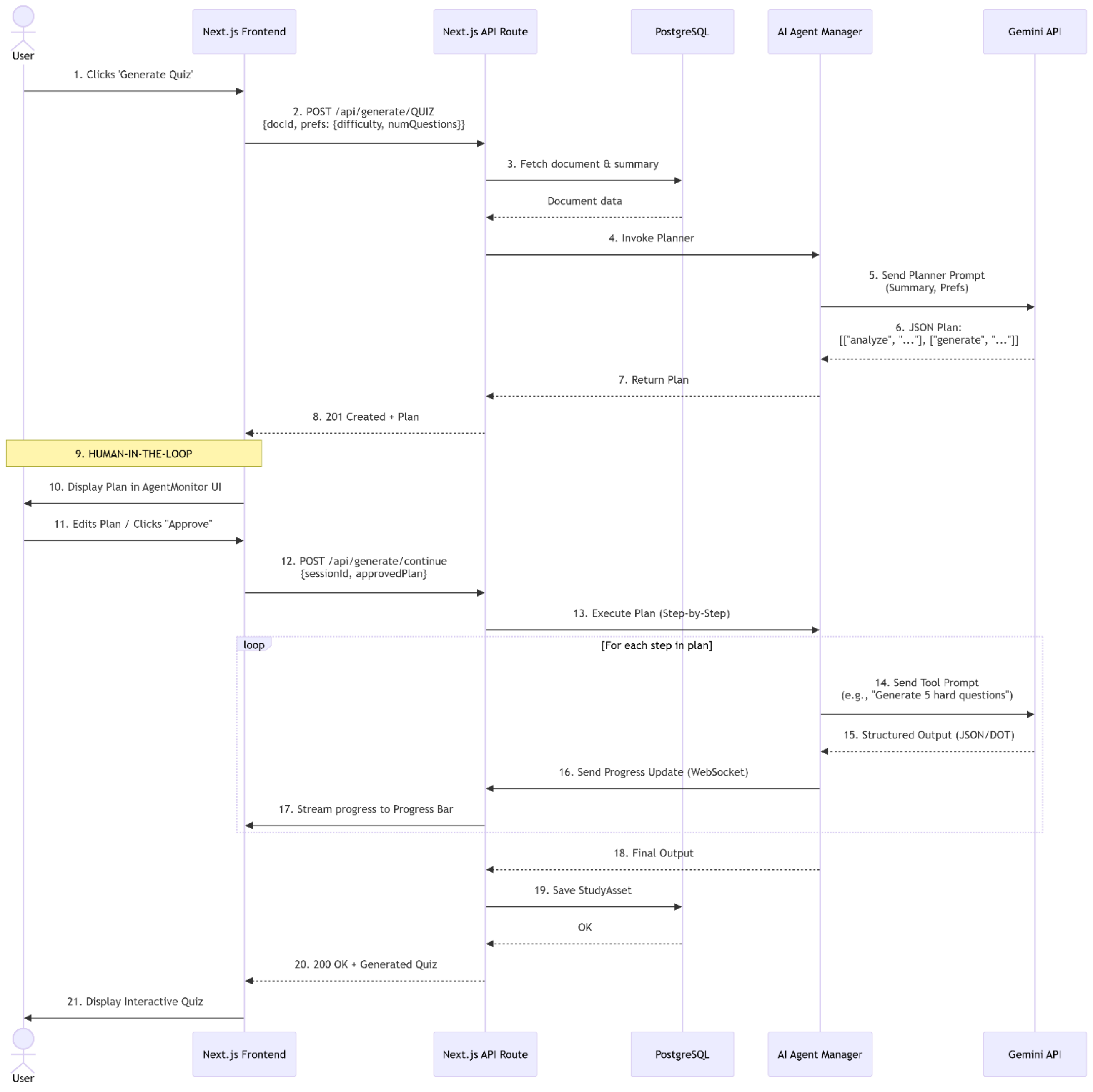
- summary_id (PK)
- user_id (FK)
- original_text
- summary_text
- created_at

Data Flow in the System

1. User Sign In
    - Clerk generates user record.
    - User profile stored in Postgres.
2. Upload Notes
    - File uploaded → backend extracts text (pdf-parse/mammoth).
    - Notes content + metadata stored in Notes table.
3. Generate Summary (Planner Agent)
    - Backend creates a Task record with type = "SUMMARY", status = "RUNNING".
    - Sends content to Gemini → receives summary.
    - Stores summary in History.
    - Updates Task status → "COMPLETED".
4. Extend into Quiz/Flashcards/Mindmap (Executor Agent)
    - User selects method + preferences.
    - New Task created with preferences.
    - Gemini generates content step by step, progress updates sent back.
    - Final output stored in History.
5. Dashboard View

- Frontend queries:
  - Notes table → list of notes.
  - Tasks & History → task results for each note.
  - Due dates + priorities → shown on home page.
- Search → queries Notes + History for matches.

Detailed Workflow


User
Next.js Frontend
Next.js API Route
PostgreSQL
AI Agent Manager
Gemini API
1. Clicks 'Generate Quiz'
2. POST /api/generate/QUIZ {docId, prefs: {difficulty, numQuestions}}
3. Fetch document & summary
Document data
4. Invoke Planner
5. Send Planner Prompt (Summary, Prefs)
6. JSON Plan: [["analyze", "..."], ["generate", "..."]]
7. Return Plan
8. 201 Created + Plan
9. HUMAN-IN-THE-LOOP
10. Display Plan in AgentMonitor UI
11. Edits Plan / Clicks "Approve"
12. POST /api/generate/continue {sessionId, approvedPlan}
13. Execute Plan (Step-by-Step)
loop
[For each step in plan]
14. Send Tool Prompt (e.g., "Generate 5 hard questions")
15. Structured Output (JSON/DOT)
16. Send Progress Update (WebSocket)
17. Stream progress to Progress Bar
18. Final Output
19. Save StudyAsset
OK
20. 200 OK + Generated Quiz
21. Display Interactive Quiz

COMPONENT BREAKDOWN:

Frontend (Next.js + Tailwind CSS)

- Authentication (Clerk)
    - Handles user sign-up, login, and session management.
- File Upload Component
    - Allows users to upload notes in PDF, DOCX, or TXT format.
- Task Selection UI
    - Lets users choose actions: Summarize, Generate Quiz, Mindmap, or Flashcards.
- Progress Bar
    - Displays Executor Agent's progress during task execution.
- Editable Planner UI
    - Shows the AI-generated study plan; users can edit steps before execution.
- Dashboard
    - Displays saved notes, task history, search bar, upcoming due dates, and priority scheduling.
- Notes Viewer/Downloader
    - Allows users to view, download, and manage their notes.
- Feature Components
    - Flashcards: Create, review, and practice flashcards.
    - Quiz: Generate and take quizzes based on notes.
    - Mindmap: Visualize and edit mind maps.
    - Summarizer: Summarize uploaded or written text.
    - SummarizationHistoryPanel: View past summarizations.
    - Sidebar/Header/Footer: Navigation and layout components.

Backend (Next.js API Routes + Prisma + Postgres)

- Authentication Sync
  - Syncs Clerk user data with backend and database.
- File Parsing
  - Converts uploaded PDF/DOCX/TXT files to plain text for processing.
- Notes API
  - CRUD operations for user notes.
- Task API
  - Create tasks, run AI agents, update task status.
- History API
  - Fetches saved results (summaries, quizzes, mindmaps, flashcards).
- Search API
  - Enables searching notes by keyword or title.

AI Agent (Gemini AI)

- Planner Agent
  - Generates a fixed-step study plan from user notes; plan is editable by the user.
- Executor Agent
  - Executes each step of the plan (summarization, quiz, flashcards, mindmap), sending progress updates to the frontend.
- Task Types Supported
  - Summarization (default first step)
  - Quiz generation (with user preferences)
  - Flashcards generation

    - Mindmap generation

Database (Postgres + Prisma ORM)

- Users Table
    - Stores user profiles and authentication references.
- Notes Table
    - Stores uploaded and created notes.
- Tasks Table
    - Stores tasks (summarization, quiz, etc.) and their statuses.
- History Table
    - Stores results of completed tasks for dashboard/history.
- Preferences Table
    - Stores user preferences for quiz difficulty, flashcard type, etc.
- Scheduling Table
    - Stores due dates and priority levels for tasks/notes.

TECHNOLOGIES CHOSEN:

1. Frontend Framework: Next.js 14 (App Router)
   I chose Next.js because it is a comprehensive full-stack framework that allows me to build both the user interface and the backend API logic within a single, cohesive project. This eliminates the complexity of managing two separate codebases. Its built-in server-side rendering (SSR) ensures the application loads quickly and performs well for users, which is essential for a good user experience. The modern App Router was selected for its improved performance, built-in data fetching, and streamlined layout system.

2. Styling: Tailwind CSS
   For styling, I selected Tailwind CSS, a utility-first framework. This

choice enables rapid UI development by providing a set of pre-designed, low-level utility classes that can be combined to build any design directly in the markup. This approach offers greater flexibility and a more unique look than using a component-based library like Material-UI, while also preventing the common maintenance headaches of traditional CSS.

3. Database: PostgreSQL
   The application requires a robust and reliable relational database to manage complex relationships between entities like users, documents, generated study assets, and schedules. PostgreSQL is the ideal choice as a powerful, open-source, and production-ready database that expertly handles these relational data needs with strong integrity and performance.

4. ORM: Prisma
   To interact with the PostgreSQL database, I use Prisma as the Object-Relational Mapper (ORM). Prisma provides type-safe database access, meaning it catches potential errors at compile time rather than at runtime. This greatly simplifies database operations like queries and migrations, ensures data integrity, and makes the code more intuitive and maintainable compared to writing raw SQL or using other ORMs.

5. Authentication: Clerk
   For user authentication, I integrated Clerk. Building a secure, full-featured auth system from scratch is a significant undertaking and a potential security risk. Clerk provides pre-built, customizable sign-in and sign-up components that seamlessly integrate with Next.js, saving immense development time and ensuring a professional, secure, and scalable authentication flow.

6. AI Provider: Google Gemini AI
   The core of the application's functionality relies on a powerful large language model (LLM) to act as the reasoning engine for the AI agent.

I chose Google's Gemini AI for its advanced capabilities in understanding context, following complex instructions, and generating structured outputs like JSON. This makes it perfectly suited for the multi-step planner/executor agent workflow, where it must decide on a plan and then execute it precisely.

7. Backend Logic: Next.js API Routes
   For the backend logic, such as processing file uploads and managing the AI agent workflow, I utilize Next.js API Routes. This allows me to build the backend endpoints directly within the Next.js project, creating a seamless full-stack application. This integrated approach is more efficient for this project than setting up a separate backend server with a framework like Express.js.