CMPS-450 Senior Project

“AI Tutor”

Developer and Author: Elijah Campbell-Ihim

Important Links

Deployed Vercel App → AI Tutor

Project Github → ECampbell37/AI_Tutor_SeniorProject

Python API Github → ECampbell37/Python-API_SeniorProject

Table of Contents

- Introduction
- Installation Instructions
- User's Manual
- Design Documentation
- File Structures
- Data Structures and Classes
- Program Source Code
- Test Plan and Results
- Summary and Conclusions
- Bibliography
- Screenshots
- Video Demonstration

Introduction

The purpose of this project is to develop an AI-powered tutoring web application that uses recent advancements in large language models (LLMs) to create a personalized, adaptable educational experience. Designed as my capstone project as a Computer Science student, the application aims to combine natural language processing, full-stack web development, and database integration into a single cohesive educational tool.

At its core, the AI Tutor uses OpenAI’s GPT-4o-mini model to mirror the role of an educational instructor across a wide variety of academic topics. Users can engage in educational conversations, receive guided lessons, and take dynamic quizzes that adapt based on their performance. The goal is to encourage learning through conversation and active participation, rather than static, one-size-fits-all lessons.

The platform is built using a modern web development stack. The frontend is created with Next.js and styled using Tailwind CSS to ensure a clean, responsive user interface. The backend consists of a Python API built with FastAPI, where the AI logic is managed using the LangChain framework. LangChain provides essential tools for conversation handling, context memory, quiz generation, and integration with OpenAI’s LLMs. Additionally, the project incorporates Supabase for authentication and persistent data storage, including user profiles, quiz attempts, and usage tracking.

Key features of the application include user login and account management, mode selection for different tutoring styles (e.g., casual, professional, kids mode), adaptive quizzes based on conversation context, PDF-based tutoring support, API usage limits per user, and achievement badges for learning milestones. The app is deployed on Vercel for the frontend and Render for the backend, making it accessible and scalable.

This project utilizes all of the skills I’ve gained during my time as a Computer Science major—ranging from AI and web development to software design and database management. More than just a technical demonstration, this project represents a practical, usable application that can help others learn in a more interactive and engaging way.

Installation Instructions

The AI Tutor consists of two main components: a **Next.js frontend** and a **Python FastAPI backend**, connected via REST API. The project also uses **Supabase** for user authentication, database storage, and API usage tracking.

Prerequisites

Before installing the project locally, ensure the following are installed on your machine:

- **Node.js** (v18 or later)
- **Python** (v3.10 or later)
- **Git**
- **An OpenAI API key** (GPT-4o-mini or other supported model)

*This project requires a **Supabase account** and an **OpenAI account** to run locally. The Supabase setup and **purchasing** an OpenAI API Key are two key barriers to running locally. **It is best to run the app on my deployed version.** However, once these two barriers are overcome, running locally is relatively straightforward.*

Project Structure

- `AI_Tutor_SeniorProject/` — Next.js frontend application
- `Python-API_SeniorProject/` — Python FastAPI backend with LangChain integration

Frontend (Next.js)

1) **Clone the repository:**

```
git clone https://github.com/ECampbell37/AI_Tutor_SeniorProject.git
cd AI_Tutor_SeniorProject
```

2) **Install dependencies:**

```
npm install
```

3) **Configure environment variables:**

Create a `.env.local` file in the root of the frontend with:

```
NEXT_PUBLIC_SUPABASE_URL=your_supabase_url
NEXT_PUBLIC_SUPABASE_ANON_KEY=your_supabase_anon_key
SUPABASE_SERVICE_ROLE_KEY=your_supabase_role_key
NEXTAUTH_URL=http://localhost:3000
NEXTAUTH_SECRET=your_secret_key
NEXT_PUBLIC_PYTHON_API=your_python_api_url
```

4) **Run the development server:**

```
npm run dev
```

This will launch the app on http://localhost:3000.

Backend (Python FastAPI + LangChain)

1) **Clone the backend repository (if not already):**

```
git clone https://github.com/ECampbell37/Python-API_SeniorProject.git
cd Python-API_SeniorProject
```

2) **Create and activate a virtual environment:**

```
python -m venv venv
venv\Scripts\activate
```

3) **Install dependencies:**

```
pip install -r requirements.txt
```

4) **Set environment variables:**

Create a `.env` file or set in your terminal:

```
OPENAI_API_KEY=your_openai_key
```

5) **Run the FastAPI server:**

```
uvicorn main:app --reload
```

This will start the API on http://localhost:8000.

Testing the Connection

Once both servers are running:

- Visit http://localhost:3000 in your browser.
- Sign up or log in with Supabase credentials.
- Begin a tutoring session and interact with the AI.
- Monitor backend API requests via the Python console or browser dev tools.

Deployment

- **Frontend**: Deployed using Vercel
- **Backend**: Deployed on Render
- Ensure environment variables are configured in both platforms' dashboards.
- Supabase code must be hosted in your own online Supabase database

If you have trouble installing and running locally, no worries! You can test my deployed site here: AI Tutor

User’s Manual

The AI Tutor web application is designed to provide an intuitive, engaging, and personalized learning experience across multiple topics and skill levels. Below is a walkthrough of how a user interacts with the application from start to finish.

Home Page

When users visit the application, they are greeted with a welcome screen that includes a clear introductory message, a "Select Your Tutor" button, and scrolling images representing various learning categories. Below this, there are links to key features such as Free Chat, PDF Mode, and the About Page. At the very bottom of the screen, there is a footer link to check the API connection, to ensure the server is running smoothly.

Authentication

To access any of the tutoring features, users must first sign up or log in. Authentication is managed through Supabase and NextAuth. New users can register with a username and password, and returning users can log in with their credentials. After logging in, the user is redirected to their account page and their session begins tracking stats data and API usage. Every user has a daily limit to how many chat messages they can receive (API Limit). If a user exceeds that limit, they can no longer interact with the AI until the next day.

Mode Selection

After clicking "Select Your Tutor," users are presented with a menu of learning modes:

- **Casual Mode**: A general-purpose tutor designed to assist with learning in a conversational, friendly tone. Features an “Other” button where users can type in any topic of their choosing and get a custom lesson.

- **Kids Mode**: A simplified version of the tutor with more kid-friendly language and easier explanations, aimed at younger users.

- **Professional Mode**: A more formal tutor that uses markdown formatting, code snippets, and LaTeX for advanced topics. Great for coding and technical questions.

Other modes include outside of the tutor section include:

- **Free Chat Mode**: An unstructured, open-ended chat session with the AI for casual questions or exploration.

- **PDF Mode**: Allows the user to upload a PDF file and ask questions based on its content. Ideal for text based summarization and information extraction. This mode is a new and experimental feature, and can sometimes give incoherent or unclear responses.

Each mode changes the behavior and tone of the AI to match the intended audience and purpose.

Chat Interface

Once a mode is selected, the user enters the chat screen. This page includes a text input box for typing questions, a send button, and a live conversation feed that displays both AI and user messages. The AI responds based on the current mode and topic, offering tailored explanations and suggestions.

In Causal and Kids Modes, the AI will start the conversation with an introduction to the topic and encourage the user to choose an aspect of the topic that intrigues them most. These modes also offer a “Take Quiz” button that generates a custom quiz based on the conversation.

Quizzes

After interacting with the tutor, the user may start a quiz by pressing the “Take Quiz” button. The AI generates five multiple choice questions based on the prior conversation. The user answers each question one at a time.

After submitting answers, the AI evaluates the results, calculates a score (as a percentage), and provides feedback. If the user performs well, the tutor may introduce more advanced material. If not, it will offer clarification or review.

Badges and Progress Tracking

Each user has an account page that tracks their progress. This includes:

- Total logins
- Number of quizzes taken
- Topics explored
- API usage with a daily limit progress bar
- Earned badges for milestones such as completing a first quiz, earning a perfect score, or exploring multiple topics

These features are designed to encourage continued learning and user engagement.

PDF Mode

In PDF Mode, users can upload a PDF document (such as notes or textbook chapters) and ask questions based on its contents. The AI will reference specific portions of the document to provide answers. Again, this mode is experimental, and can sometimes fail to give useful answers.

Logging Out

Users can log out at any time using the menu icon on the top of the screen. Session memory resets whenever the user navigates away from a chat or logs out. Each tutoring mode maintains its own behavior and logic during use.

Server Status

This app is hosted on Render Free Tier. This means that **the server will go to sleep every 15 minutes of inactivity (no requests being sent to it from anyone).** When a user first gets on the site, there will likely be an increased wait time to use the AI features, as the server needs to wake back up again on Render. **Once the server is awake, however, loading times should be significantly improved.** If, for whatever reason, there seems to be problems with connecting to the server, visit the Status Check page to see if the server is online. If it is working as intended, the loading icon will show up green. If there is a more serious backend issue with the server, the loading icon will show up red.

Design Documentation

Design Summary

The AI Tutor is a full-stack AI-powered web application, which consists of a responsive frontend built in Next.js, a Python-based backend powered by FastAPI and LangChain, and a Supabase database for user authentication, progress tracking, and API usage enforcement.

This design architecture utilizes separation of concerns, allowing each functional component—AI conversation, quiz generation, PDF-based tutoring, UI rendering, and user management—to operate independently while sharing a consistent user session. Multiple modes (Casual, Kids, Professional, Free Chat, and PDF) tailor the AI's behavior to different user needs, all while maintaining appropriate conversational context through memory management.

System Architecture

The system is composed of three primary layers:

1. **Frontend (Next.js + Tailwind CSS)**
    - Built using React-based routing with dynamic components.
    - Handles user navigation, chat UI, API interaction, and progress tracking.
    - Integrates next-auth for secure authentication via Supabase.
2. **Backend (FastAPI + LangChain + OpenAI)**
    - Uses REST API endpoints to handle different learning modes.
    - LangChain is used to build chains for each mode: including intro messages, response handling, quizzes, and continuation logic.
    - GPT-4o-mini is the core LLM used for language generation and comprehension tasks across all tutor modes
3. **Supabase (PostgreSQL + Auth)**
    - Provides user authentication with hashed password storage.

- Stores quiz attempts, daily sign ins, topics explored, API usage data, and earned badges.
- Enforces security using Row Level Security (RLS) tied to user sessions, which is recommended by Supabase.

Top-Level Design Map

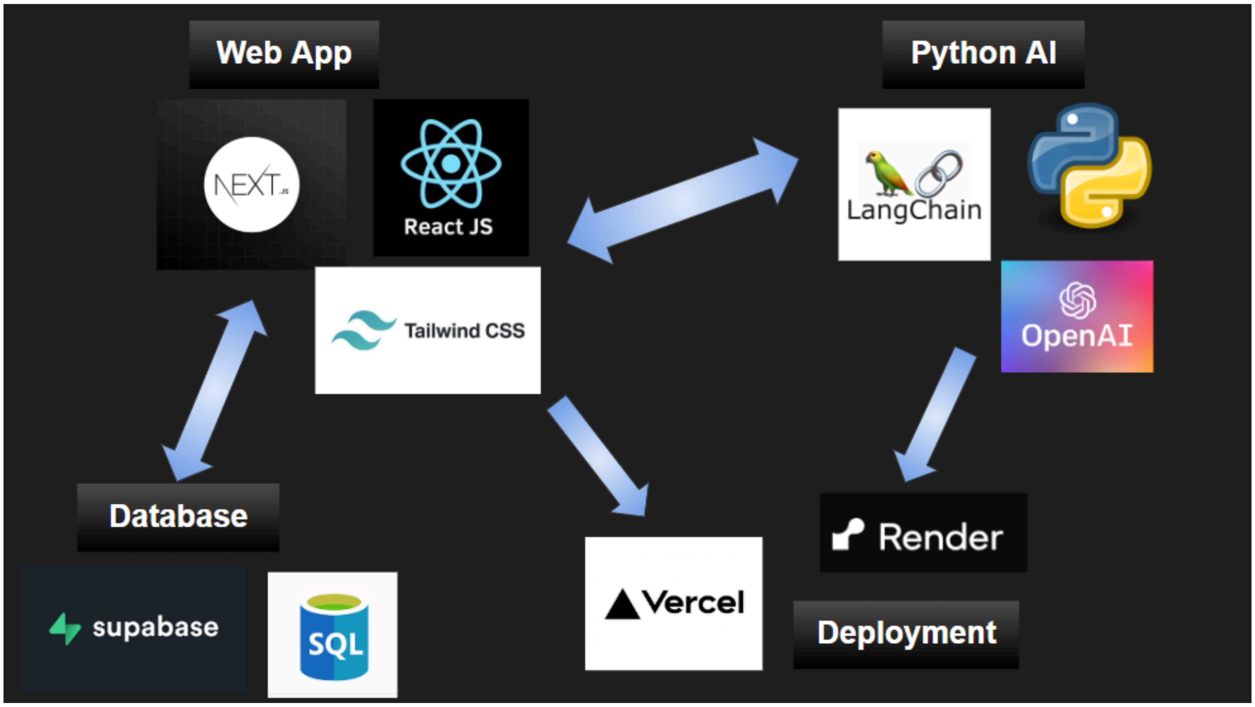


Figure above shows the 4 top-level components of AI Tutor: ***Web App, Python AI, Database, and Deployment****. The arrows indicate communication lines between components. Web App is deployed on Vercel, Python API is deployed on Render*

Frontend Design

The frontend of the AI Tutor is built using **Next.js**, a modern React framework that supports server-side rendering, routing, and API integration. Styling is done using **Tailwind CSS**, which allows for modern, aesthetic and response design.

The application is divided into dynamic routes for each tutor mode (e.g., /freeChat, /kidsChat, /professionalChat, etc.). Each route renders a chat interface that interacts with the backend via REST API calls.

Key UI components include:

- A chat interface with styled message bubbles for both user and AI messages
- An input box with a send button and loading spinner
- Markdown rendering support for professional responses (including LaTeX and code blocks)
- An account page that displays usage statistics and earned badges
- A usage progress bar that updates in real time based on API calls

Authentication is handled through next-auth, which connects with Supabase to manage secure login sessions. Protected routes and API limits are enforced client-side and validated server-side.

Frontend Folder Highlights *(more info in File Structures section)***:**

- **app/**: Root application folder
- **lib/**: Static informational files, such as Supabase Initialization and profile badge info and conditions
- **app/components/**: Reusable UI parts like MarkdownRenderer and the navigation bar
- **app/(protected)/**: Files that require user authentication (log in) to access, including chat and tutor pages. Also features an animated loading screen that checks if the Python API server is online before proceeding to the page
- **app/api/**: Internal API routes (not to be confused with Python API routes). Includes many Supabase database functionalities and nextAuth authentication.

Backend Design

The backend is implemented using **FastAPI** with **LangChain** for AI coordination. Each tutoring interaction is composed of dedicated LangChain **chains**— components that encapsulate prompts, memory, and language model logic. The backend is designed for clarity, reusability, and conversational flow.

Each mode uses different chains depending on the desired user experience. For example, **Casual Mode** and **Kids Mode** use multiple chains in sequence, including:

- **Intro Chain** – Introduces the topic and starts the lesson
- **Response Chain** – Handles ongoing conversation during the session
- **Quiz Generation Chain** – Creates five personalized multiple choice quiz questions
- **Quiz Grading Chain** – Evaluates user answers and returns a score (e.g., “Grade: 80%”)
- **Quiz Feedback Chain** – Gives customized follow-up analysis based on performance
- **Continuation Chain** – Resumes the lesson after the quiz, adjusting based on quiz results

Other modes like **Professional** or **Free Chat** simplify the chain usage by having a single chain that is merely a response engine. They are not constructing a lesson or generating quizzes, so a single response chain suffices in these modes.

PDF Mode uses a ConversationalRetrievalChain, which:

- Converts uploaded PDFs into text using PyMuPDF **(Load)**
- Splits the text into chunks **(Split)**
- Embeds the chunks using OpenAIEmbeddings **(Embed)**
- Stores them in a FAISS vector database **(Vector Store)**
- Answers questions based on the uploaded material (using **Retriever**)

Note: PDF Mode follows the Load → Split → Embed → Vector Store → Retriever model that is a signature of **RAG (Retrieval Augmented Generation)** development. This is a major architecture in Generative AI Engineering, and I am glad I had the chance to implement it in my project.

Memory

Casual, Kids, Free Chat, and Professional Modes use a **ConversationSummaryMemory**. This is a specific type of conversation memory in LangChain that stores previous contexts as a summary, and adds to that summary as conversations continue. This is better than logging the entire conversation through a ConversationBufferMemory, as when the conversation gets long, it becomes harder for the LLM to retain context and remember what has been said. With a summary memory, the important parts of the conversation are retained, and the LLM remains able to operate efficiently.

In PDF mode, conversation memory is maintained using **ConversationBufferMemory**, as this mode is less conversational and more geared towards querying a text, so retaining the full memory is a reasonable approach.

Memory is stored per-user via a Python Dictionary, with a user's session ID as the key and their current memory object as the value. This ensures that **every user has their own conversational memory**, and that the user experience for any one user is not affected by any other user.

Memory is not persistent across sessions. They will remain active for as long as you are in the conversation, and as long as the Python API server is active. They are stored in RAM, and not in Supabase (as that would be unfeasible for the scope of this project). Therefore, when you leave a chat page, or are inactive for a significant period of time, the memory is cleared and a new conversation history is started. Whenever you switch modes or topics within the application, a new memory is started, and the old one is cleared.

Frontend-Backend Communication

The frontend sends POST requests to FastAPI endpoints like /freeChat, /intro, /quiz/start, and /quiz/submit. These routes trigger different LangChain chains or logic branches on the backend. The responses are returned as plain text or markdown via **REST API**, then read by the frontend and rendered in the chat interface. The frontend uses **fetch calls** and reads in the data via **JSON**, and the backend uses **FastAPI HTTP requests** to respond to incoming frontend requests.

Database Design

The application uses **Supabase** as its database and authentication provider. Supabase is a PostgreSQL-based backend-as-a-service that supports real-time updates, secure user management, and built-in row-level security.

The database is used to:

- Store user accounts and credentials
- Track quiz history, topics explored, and login count
- Enforce API rate limits per user
- Manage a badge system tied to user milestones

Key Tables:

- **users**: Stores basic profile and authentication data
- **user_stats**: Tracks quizzes taken, topics explored, and total logins
- **api_usage**: Logs daily API request counts per user, per day
- **badges**: Maps users to earned badges

All tables enforce **Row Level Security (RLS)**, ensuring users only access their own records. Badge logic and conditions are handled in the frontend, but badge eligibility is determined using Supabase queries and updated via serverless API routes.

Entity Relationship Diagram (ERD) of AI Tutor Database

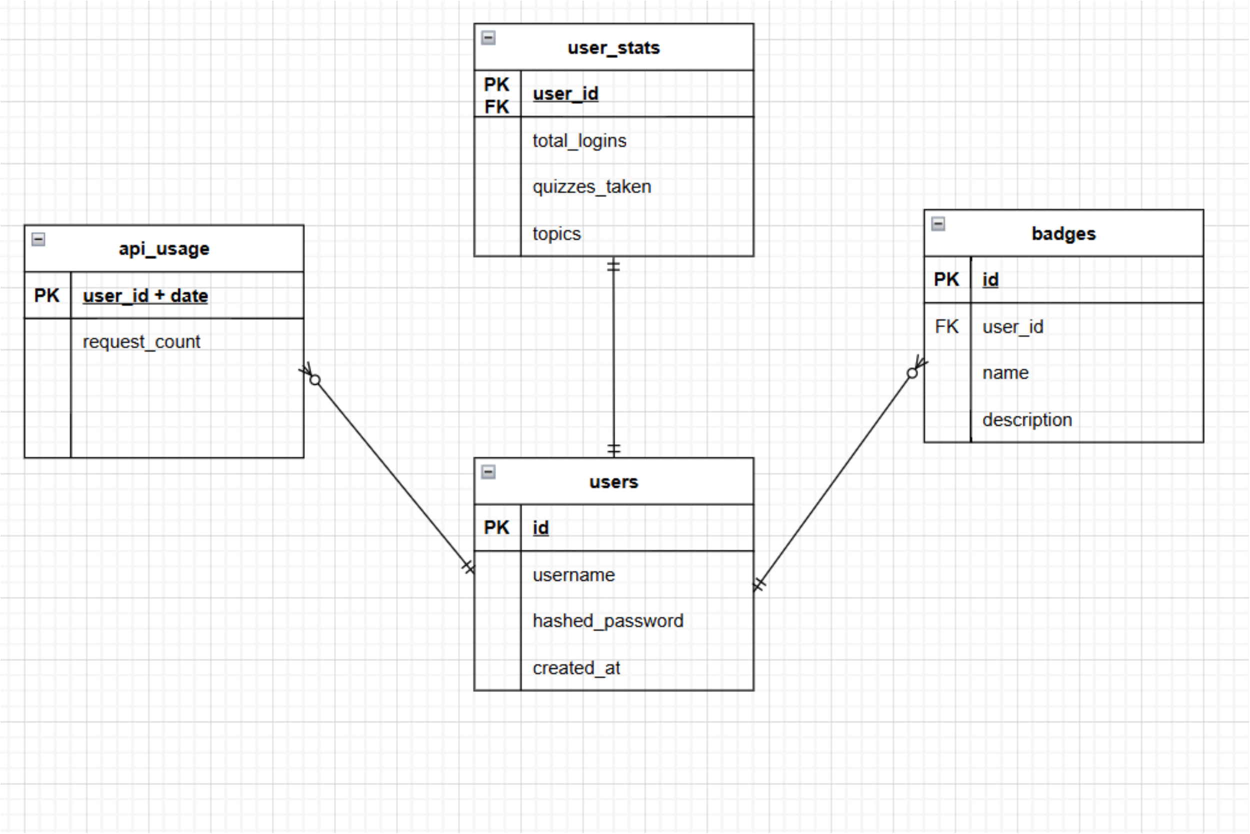


Figure above shows the relationships between the four tables in the AI Tutor Database. Every user has exactly one set of stats, and every set of stats belongs to exactly one user. Users can have multiple entries in the api_usage table, but every entry corresponds to exactly one user. Users can have many badges, but each badge entry is awarded to exactly one user. The user_id key is used across every table to link users to their data.

File Structures

The Frontend (necessarily) contains considerable folder nesting and numerous files, as Next.js organizes its routing through folders. Within the app folder, all visitable pages in the application must be called **page.tsx**, and must be stored in a folder that can be routed to (such as “account” or “signin”). Likewise, all api routes must be called **route.ts** and stored in a folder with a descriptive name. Therefore, the file structure for the Frontend is complex, and I will outline the major components below. I am only including in this section the folders and files that I coded myself (and not folders holding images or configuration and external module code)

Frontend (AI_Tutor_SeniorProject):

(Starting from root directory)

- app/
  - (protected)/ # pages with account-restricted access
    - chat/
      - ChatClient.tsx # Casual mode chat page
      - page.tsx # Loading screen, then chat page
    - freeChat/
      - FreeChatClient.tsx #Free Chat Mode chat page
      - page.tsx # Loading screen, then chat page
    - kidsChat/
      - KidsChatClient.tsx # Kids Mode chat page
      - page.tsx # Loading screen, then chat page
    - kidsTopics/
      - page.tsx # Topic selection screen for kids tutor
    - pdfChat/
      - page.tsx # PDF Mode chat page
    - pdfUpload/
      - page.tsx # PDF Mode file upload page
    - professionalChat/
      - ProChatClient.tsx # Professional Mode chat page
      - page.tsx # Loading screen, then chat page
    - topics/
      - page.tsx # Topic selection screen for casual tutor
    - tutor/
      - page.tsx # Tutor selection page (kids, casual, pro)
    - layout.tsx # authentication and server access check
  - about/
    - page.tsx # about page
  - account/

```
                ●  page.tsx                 # account page (locked if not signed in)
        ◆  api/
                ●  auth/
                        ○  [...nextauth]/
                                ◆  route.ts      # nextAuth handler (reserved folder name)
                        ○  signup/
                                ◆  route.ts      # adds user to database on sign-up
                ●  badges/
                        ○  update/
                                ◆  route.ts      # updates user profile badges if criteria met
                        ○  route.ts              # retrieves user badges from database
                ●  joined/
                        ○  route.ts              # retrieves user start date from database
                ●  stats/
                        ○  login/
                                ◆  route.ts      # retrieves and updates user login data
                        ○  quiz/
                                ◆  route.ts      # retrieves and updates user quiz data
                        ○  topic/
                                ◆  route.ts      # retrieves and updates user topics data
                        ○  route.ts              # retrieves and updates user stats data
                ●  usage/
                        ○  check/
                                ◆  route.ts      # checks and updates user’s API usage
                        ○  route.ts              # retrieves user’s API usage from database
        ◆  components/
                ●  MarkdownRenderer.tsx     # Component to render markdown content
                ●  NavBar.tsx               # Nav bar at top of all screens
                ●  Providers.tsx            # Session wrapper for next-auth
        ◆  signin/
                ●  page.tsx                 # sign in page
        ◆  signup/
                ●  page.tsx                 # sign up page
        ◆  statusCheck/
                ●  page.tsx                 # status check page

        ◆  globals.css                      # base level global css config
        ◆  layout.tsx                       # root layout across entire app
        ◆  page.tsx                         # welcome page (initial landing page)


➔  lib/
        ◆  SupaBaseClient.ts                # Supabase configuration and initialization
        ◆  badges.ts                        # Profile Badge information and criteria
```

- ➔ .env.local # Important private info (keys and urls)

There are many more files and directories that are necessary for the project to run (which can be seen on Github), but these are the fundamental files and directories that I personally coded on the frontend.

Backend (Python-API_SeniorProject):

(All files are in root directory, no sub-folders)

- **main.py**: Entry point and API routes. This is where FastAPI is used, and what is directly being hosted by Render via uvicorn.

- **casualLearning.py**: Casual Learning Mode Prompts, Chains, and Core Functionalities (the actual AI behind Casual Learning Mode)

- **kidsLearning.py**: Kids Mode Prompts, Chains, and Core Functionalities (the actual AI behind Kids Mode)

- **professionalLearning.py**: Professional Mode Prompts, Chains, and Core Functionalities (the actual AI behind Professional Mode)

- **freeChat.py**: Free Chat Prompts, Chains, and Core Functionalities (the actual AI behind Free Chat Mode)

- **pdfLearning.py**: Text Parsing, Language Model Embeddings, Vector Store, Retriever, and Chat Chain for PDF Mode (manages the data preparation and the actual AI queries for PDF Mode)

- **requirements.txt**: Library dependencies to be installed by Render (or local server)

- **render.yaml**: Render config file that specifies which libraries to download and which file is the root/main file to be executed on Render

Data Structures and Classes

Frontend (React + TypeScript)

The frontend of AI Tutor is built using React and TypeScript, with a focus on keeping each tutoring mode (like Casual, Kids, or Professional) separate and easy to manage. The app uses React hooks to handle user input, chat messages, loading states, and quiz logic in each mode.

React Hooks

The main way the frontend manages dynamic behavior is through React's built-in hooks:

- **useState** – Keeps track of changing values like what the user types, the chat history, whether a quiz is active, and if the app is waiting on a response.

- **useEffect** – Runs extra logic when the component loads or updates, like checking how many API requests the user has made or triggering updates to the user's stats.

- **useSession** – Provided by NextAuth, this hook gives access to the user's login session so that the app can check if they're authenticated before letting them access certain pages.

Each chat mode has its own React component and its own set of hooks, so the logic stays isolated. This makes it easier to test and change one mode without affecting the others.

Hooks are not shared across modes. Instead, each chat page controls its own state, memory, and interaction flow. This keeps the app well organized and customizable, and allows us to apply very specific behavior per mode.

Component Structure

The app uses Next.js App Router, which means each page is built as a separate component based on its route. Every tutoring mode has its own page and chat component that controls how the AI interaction works.

Some important parts of the frontend structure include:

- **Message Objects** – Every message sent by the user or the AI is stored as an object that includes who sent it and what the message says. These are shown in the chat bubble

UI.

- **Chat Clients** – These are the main components for each mode (like ChatClient, KidsChatClient). They:
    - Track the full conversation
    - Handle user input
    - Show loading animations
    - Send messages to the backend and display the AI’s reply
    - Trigger quizzes and other flow changes
- **Markdown Renderer** – A special component that formats the AI’s response if it includes code blocks, math equations, or bold/italic text. This is especially useful in Professional Mode.
- **Account Page Components** – These components show stats like quizzes taken or badges earned. They are updated in real-time based on how the user interacts with the app.
- **Loading and Status Screens** – Protected routes (like chat pages) show a loading animation before checking whether the backend is online. If the Render server is asleep, the component will re-ping the server until the server is awake and it receives a response. This is done through the layout.tsx of the protected route folder.

Overall, the frontend is built to follow the strengths of React and Next.js. It uses hooks to keep the UI in sync with user behavior and isolates each part of the app so that it’s easy to manage and expand.

Backend (Python + FastAPI + LangChain)

The backend is written in Python using FastAPI and LangChain, and is responsible for handling all of the AI logic, quizzes, memory, and PDF processing. Two key pieces make up the backend: LangChain chains and the FastAPI app.

LangChain Chains

The core building blocks of the backend are LangChain’s chain objects, especially LLMChain. Each chain is an aspect of the actual “AI Tutor” that does one specific job—like giving an introduction, answering a question, creating a quiz, or giving feedback.

Each tutoring mode (Casual, Kids, etc.) has its own **dedicated Python file** (casualLearning.py, kidsLearning.py, etc.) where all its chains and prompts are defined. This keeps mode-specific logic organized, making it easier to update or expand without affecting the rest of the system.

Chains are set up with:

- A custom prompt template
- The GPT-4o-mini model
- A memory system that remembers key parts of the conversation

These chains are created at the start of a user’s session and stored temporarily in a Python dictionary that keeps track of each user by their session ID. This lets the app quickly find the right chains for the right user and tutoring mode.

Two types of memory are used depending on the mode:

- **ConversationSummaryMemory** – Summarizes the session and helps the model retain key points over longer chats (used in most modes).
- **ConversationBufferMemory** – Stores the entire conversation word-for-word, mostly used in PDF Mode where full context is important.

All chains are short-lived. When the user switches modes or is inactive for too long, the memory is cleared and the chains are re-initialized.

PDF Mode Vectorstore

In PDF Mode, the backend uses a different architecture known as **RAG (Retrieval-Augmented Generation)**. When a user uploads a PDF:

- The file is parsed using PyMuPDF
- The text is split into overlapping chunks

- Embeddings are created using OpenAI’s embedding model
- A FAISS index is built from those chunks
- A ConversationalRetrievalChain is created that allows the user to ask questions based on the file

Each user gets their **own temporary vectorstore**, and this is only held in memory during the session. If the user clears or leaves, the file and index are removed.

FastAPI App

The backend runs as a single FastAPI application, defined in main.py. It handles incoming requests from the frontend and routes them to the right chains.

Key things to know:

- Each tutoring mode has its own endpoint (like /casual, /kids, etc.)
- Routes are **asynchronous**, so they can handle multiple users at once
- **CORS (Cross-Origin Resource Sharing)** is enabled to let the frontend (on Vercel) and backend (on Render) communicate safely

The FastAPI app acts as the central controller that routes requests to the correct LangChain chains and returns the result. It doesn’t hold onto any data long-term—it just manages sessions in memory and resets them when users leave or switch modes. Together, **LangChain chains act like task-specific AI helpers**, while **FastAPI handles all the communication** between the frontend and backend, keeping the app responsive, well organized, and easy to maintain.

Program Source Code

All of the program's source code can be found on Github (links on cover page). I will also email with my submission a zip file containing all of the source code (Nextjs, Python, and Supabase), so you can download the code there too.

Test Plan and Results

The AI Tutor application was tested using a combination of live-use and scenario-based testing to validate functionality across all core features. Testing focused on ensuring correct behavior across the frontend UI, backend API routes, AI response quality, quiz accuracy, and database updates. Below is a breakdown of the testing strategy and observed results.

Test Environment

- **Frontend**: Tested locally at localhost:3000 and on Vercel deployed site
- **Backend**: Tested at localhost:8000 and on Render deployed server
- **Browsers**: Microsoft Edge and Safari
- **Devices**: PC (Laptop) and Mobile (iPhone)

Test Categories

Below are categories of testing and their corresponding results.

✅ = Fully Functional ⚠️ = Mostly/Partially Functional ❌ = Not Functional

1. Sign in functionality

- Goal: Users can navigate to sign in and sign up pages, can create an account, and log in to their existing accounts.

- Test:
    - Can navigate to login ✅
    - Can navigate to sign up✅
    - Can create an account✅
    - Can sign in to existing account✅
    - Incorrect login credentials are rejected ✅

2. Authentication and Navigation

- Goal: Pages can all be navigated to properly, and only authenticated users can access chat pages.

- Test:
    - All page.tsx files are accessible through the app✅
    - UI is clean and intuitive ✅
    - Only authenticated users can access tutor functionality✅
    - Unauthenticated users are redirected to signin✅
    - Selection screens (tutor, subject, stc.) route to correct destination ✅

3. Chat Functionality (Free, Casual, Kids, Pro)

- Goal: Ensure the AI responds correctly in each mode

- Test:
    - Casual Mode:
        - Starts with descriptive intro✅
        - Engages with user and facilitates interest✅
        - Outputs are formatted correctly (Markdown if needed) ✅
        - When finished with quiz, continues the lesson✅
        - If the user performed well on the quiz (above 75%), the tutor advances to new material✅
        - If the user performed poorly on the quiz (below 75%), the tutor reinforces previous material✅
    - Kids Mode:
        - Starts with descriptive intro✅
        - Always uses child-friendly language and tone to address its audience✅
        - Engages with user and facilitates interest✅
        - Outputs are formatted correctly (Markdown if needed) ✅
        - When finished with quiz, continues the lesson✅

        - If the user performed well on the quiz (above 75%), the tutor advances to new material✅
        - If the user performed poorly on the quiz (below 75%), the tutor reinforces previous material✅
    - Professional Mode:
        - Can help user with advanced topics such as math and coding ✅
        - When the user asks for code, the code is formatted correctly, and is able to be copy and pasted successfully✅
        - When the user asks for mathematical help, equations are formatted properly using LaTeX ⚠️
    - Free Chat:
        - Engages with user and facilitates interest✅
        - Outputs are formatted correctly (Markdown if needed) ✅

4. Quiz Generation and Grading (Kids, Casual)

- Goal: Confirm AI generates topic-relevant quizzes and grades fairly

- Test:
    - New quiz is generated whenever “Take Quiz” button is pressed ✅
    - Quiz has 5 multiple choice questions ✅
    - Questions are numbered ⚠️
    - Responses can be recorded ✅
    - When quiz is submitted, grade and feedback are outputted✅
    - Grade is factually correct and accurate ⚠️
    - Grade is clear, with ratio of correct to incorrect, and a final percentage score ✅
    - Feedback is constructive and useful, and reinforces lesson✅
    - Feedback matches grade✅
    - All information is relevant and relates to the previous conversation and topic ✅
    - Can generate multiple quizzes per conversation✅

5. PDF Mode

- Goal: Verify file upload, text processing, and AI response relevance

- Test:
    - File can be uploaded from frontend✅
    - File data is sent to backend✅
    - File is processed and is able to be accessed by AI ⚠️

  - AI can answer questions based on document provided ⚠️
  - AI answers are relevant and directly related to the text ✅
  - AI can read and use a diverse set of PDF inputs ⚠️
  - AI can retain conversation memory and continually respond to user ✅

6. Progress Tracking and Badges

- Goal: Ensure Supabase logs activity and awards badges

- Test:
  - Users are stored properly in database✅
  - Daily Logins are recorded✅
  - Quizzes taken are recorded✅
  - New Topics Explored are recorded✅
  - Badges are awarded when conditions are met✅
  - All stats and badges are displayed correctly on user profile✅

7. API Limits

- Goal: Enforce daily request limits per user

- Test:
  - Every user has an API Limit✅
  - The API Limit is displayed on user profile✅
  - The API usage updates every time the user uses chat functionality✅
  - When the user maxes out their API usage, api calls are stopped and placeholder messages are shown instead, telling the user they are out of messages ✅

- *Method: API Limit was temporarily reduced to 10 messages for testing*

Testing Results

Based on my testing results, the project works entirely as expected and as specified in my proposal.

Listed below are the areas that were mostly/partially functional and an explanation of the problems associated

- **Professional Tutor Math**: The mathematical functions inside the AI output can sometimes be incorrectly translated from LaTeX. For major math functions in the output, this is very rare, and outputs look great. However, for smaller functions and variable references inside sentences, the MarkdownRenderer component has some trouble rendering them properly, and they can appear unrendered and hard to read.

- **Quiz Numbers**: Questions in the quiz should be numbered 1-5. Usually, this is rendered correctly. However, because the quizzes are dynamically AI generated, the LLM is ultimately responsible for including the numbers, and although it was explicitly told to include them, it can sometimes forget to include them. This is a minor bug, as the question numbers are merely for user convenience, but are nevertheless worth mentioning.

- **Quiz Grading**: The LLM is responsible for grading the quiz. The majority of the time, the grades are factually correct and accurate. However, this is not a 100% guarantee, as the LLM is ultimately responsible for handling the grading, and can make mistakes in its analysis. Upon occasion, correct answers are falsely graded as incorrect.

- **PDF Tutor**: The PDF Tutor is far from perfect, and is an experimental feature. It can handle clear questions and text-only pdf inputs relatively well, but unstructured data such as images can confuse it, and it will often output “I don’t know” to questions it should be able to answer. Since this is not designed to be a core feature of the app, this is not a major concern, but in future development of this app, the PDF Mode will need some fine-tuning to be able to generalize well and handle a wide range of questions and scenarios.

✅ Overall, the app performs excellently under all testing, and performs all core features as expected according to my proposal.

Summary and Conclusion

The AI Tutor project represents the consolidation of my experience as a Computer Science major and combines multiple areas of computer science into a single full-stack application. Through this project, I developed a web-based educational tool that uses artificial intelligence to deliver personalized tutoring sessions, complete with adaptive quizzes, user tracking, and multiple learning modes.

This project provided an opportunity to apply and build upon my skills in modern web development, backend API design, natural language processing, database integration, and user experience design. Working with LangChain and OpenAI's GPT-4o-mini allowed me to explore creative prompt engineering techniques and conversation memory handling. On the frontend, using Next.js and Tailwind CSS gave me experience building an aesthetic, responsive, and intuitive user interface. Supabase allowed me to understand the process of working with and managing real data, and integrating user authentication and external data storage in an application. This experience was integral to my growth as a developer and allowed me to experiment with what is possible in today’s day and age.

In the end, I was able to produce a working, deployable AI educational tool that I am proud of. The application is already fit for personal and educational use, and with future improvements, it could even evolve into a commercial product or serve as the conceptual foundation for one. Most importantly, this project gave me practical, hands-on experience integrating AI into a real-world application — experience that will be valuable as I pursue a career in AI engineering. Taking on the mission of building this project has proved to be beyond worthwhile, and I am thankful for this opportunity to build something novel and within my domain of interest. I am proud of myself for making my vision for this project become a reality, and I have gained confidence in my ability to turn ideas into concrete results, and consequently turn words into action.

Bibliography

External Sources

1. OpenAI. *GPT-4o and API Documentation*.
   https://platform.openai.com/docs

2. LangChain. *LangChain Documentation*.
   https://docs.langchain.com

3. Supabase. *Open Source Firebase Alternative*.
   https://supabase.com/docs

4. Next.js. *React Framework for Production*.
   https://nextjs.org/docs

5. NextAuth. *NextAuth Documentation.*
   Introduction | NextAuth.js

6. TypeScript. The *TypeScript Handbook.*
   TypeScript: Handbook - The TypeScript Handbook

7. Tailwind CSS. *Utility-First CSS Framework*.
   https://tailwindcss.com/docs

8. Vercel. *Frontend Deployment Platform*.
   https://vercel.com

9. Render. *Cloud Hosting for FastAPI*.
   https://render.com

10. FastAPI. *High Performance Web Framework for APIs*.
    https://fastapi.tiangolo.com

11. PyMuPDF. *PDF Parser and Analyzer for Python*.
    https://pymupdf.readthedocs.io

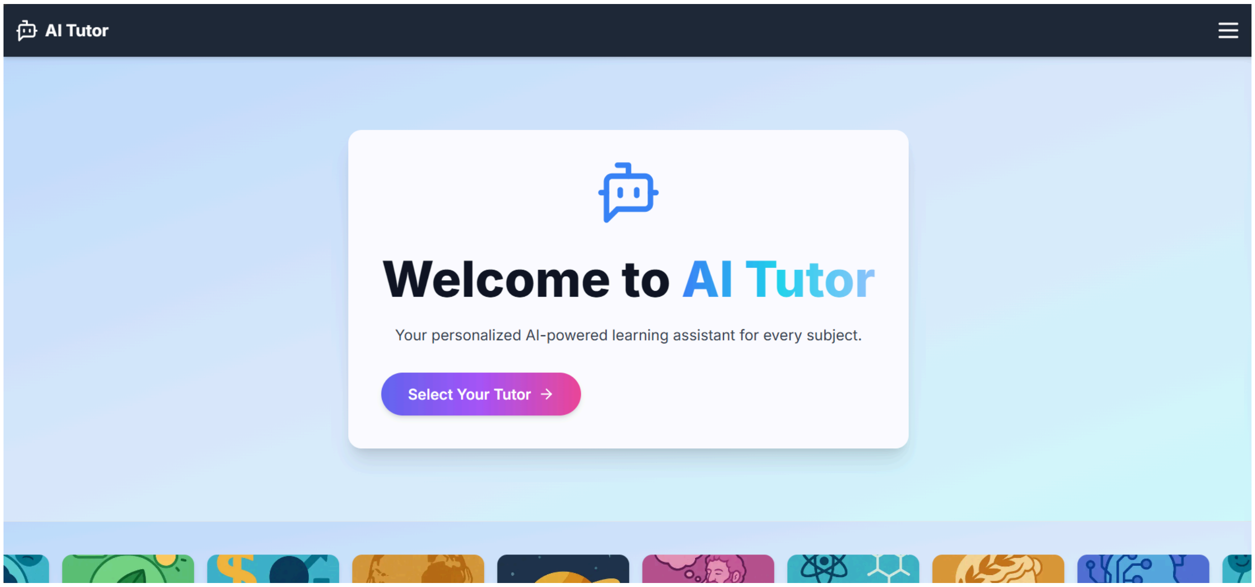
Ramapo Courses

- CMPS-369 Web Application Development
- CMPS-364 Database Design
- CMPS-285 Mobile Development
- CMPS-320 Machine Learning

External Courses & Projects

- IBM AI Engineering Professional Certificate
  Coursera Certification
- Machine Learning Specialization
  Coursera Specialization
- Personal Portfolio Website
  Site: Elijah Campbell - Portfolio
  Github: ECampbell37/portfolioWebsite
- LangChain for LLM Application Development
  Coursera Course

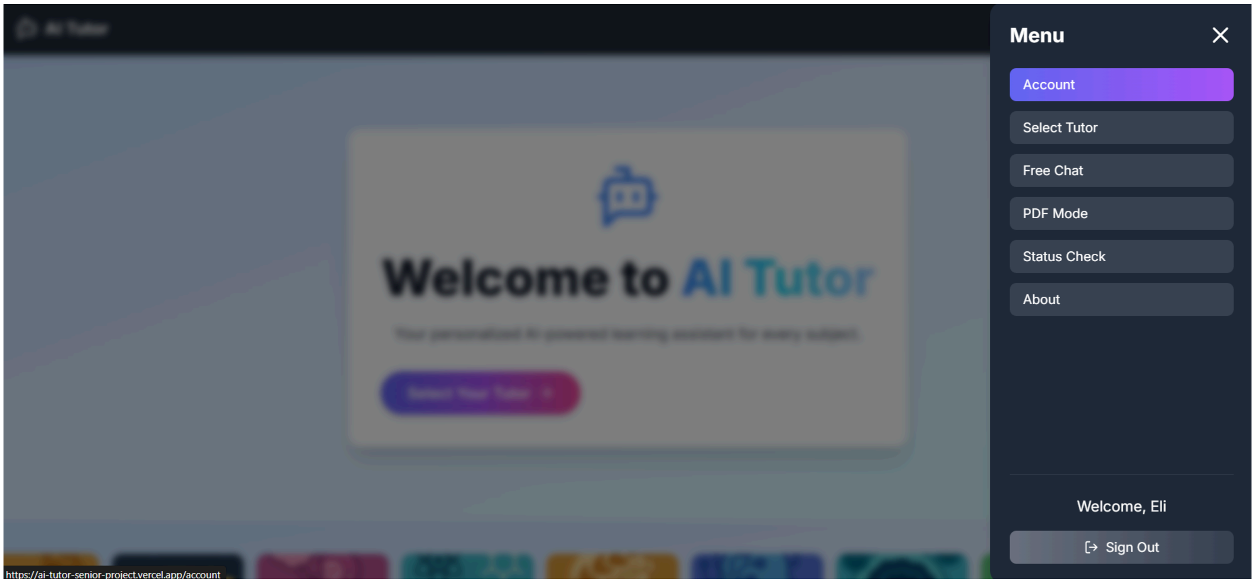
Screenshots



Welcome to AI Tutor

Your personalized AI-powered learning assistant for every subject.

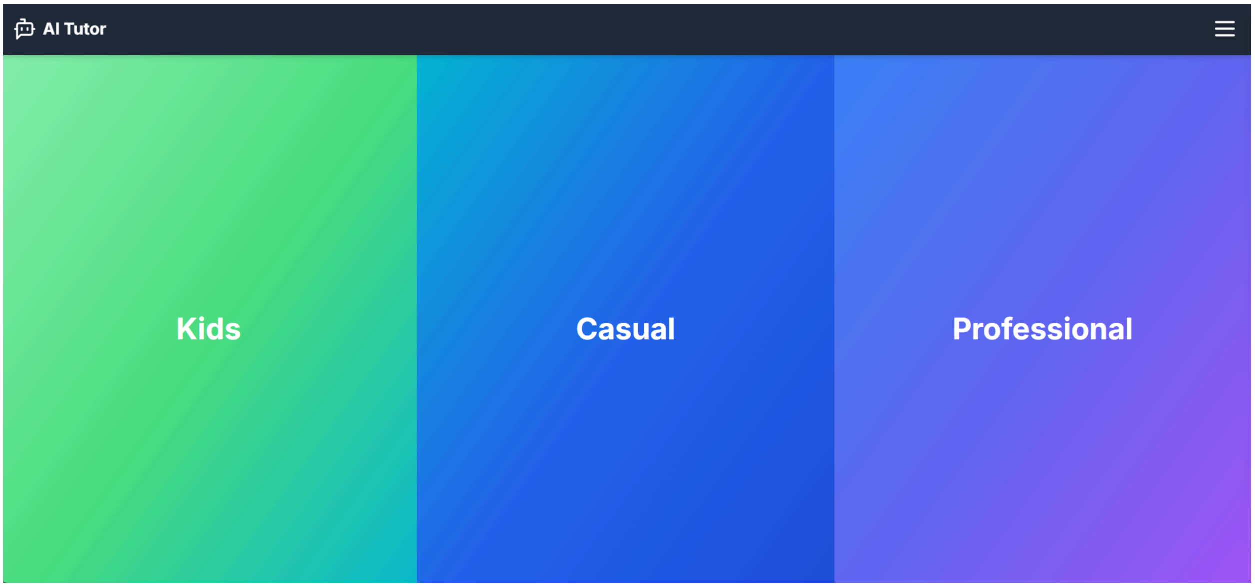
Home screen



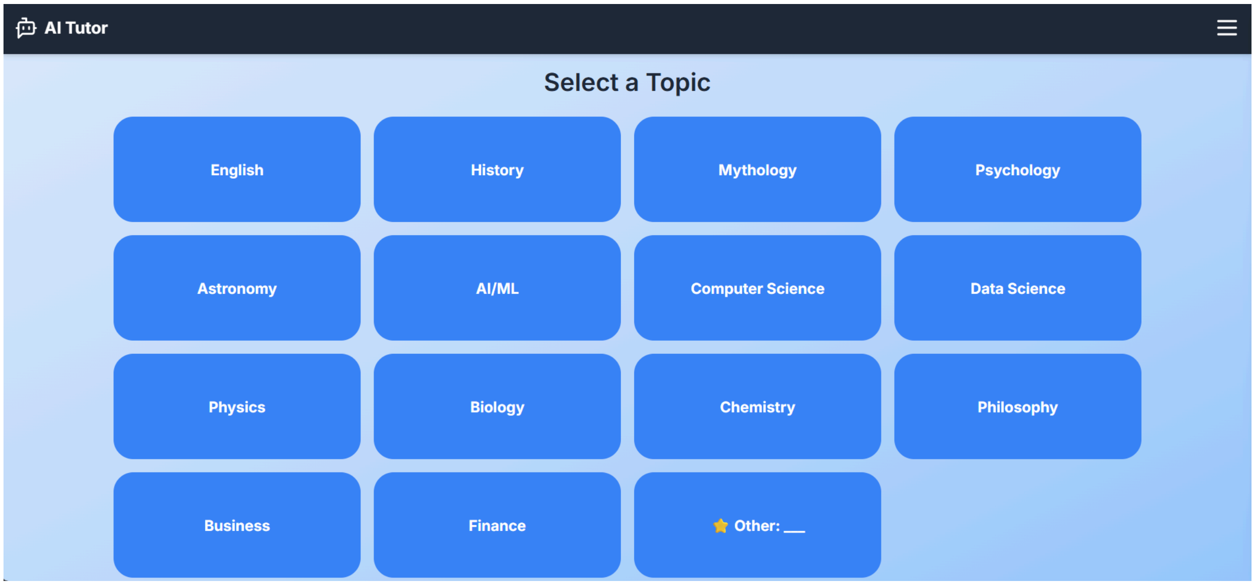
Welcome, Eli

https://ai-tutor-senior-project.vercel.app/account

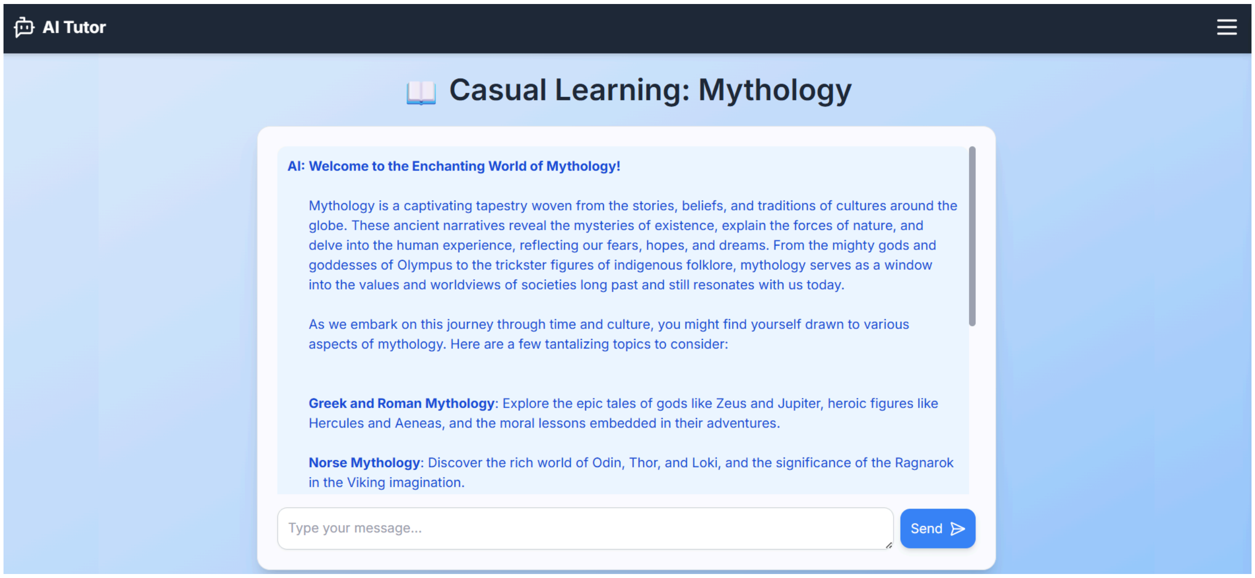
Navigation Menu



Tutor Selection Page



Topic Selection Page for Casual Learning



📖 Casual Learning: Mythology

AI: Welcome to the Enchanting World of Mythology!

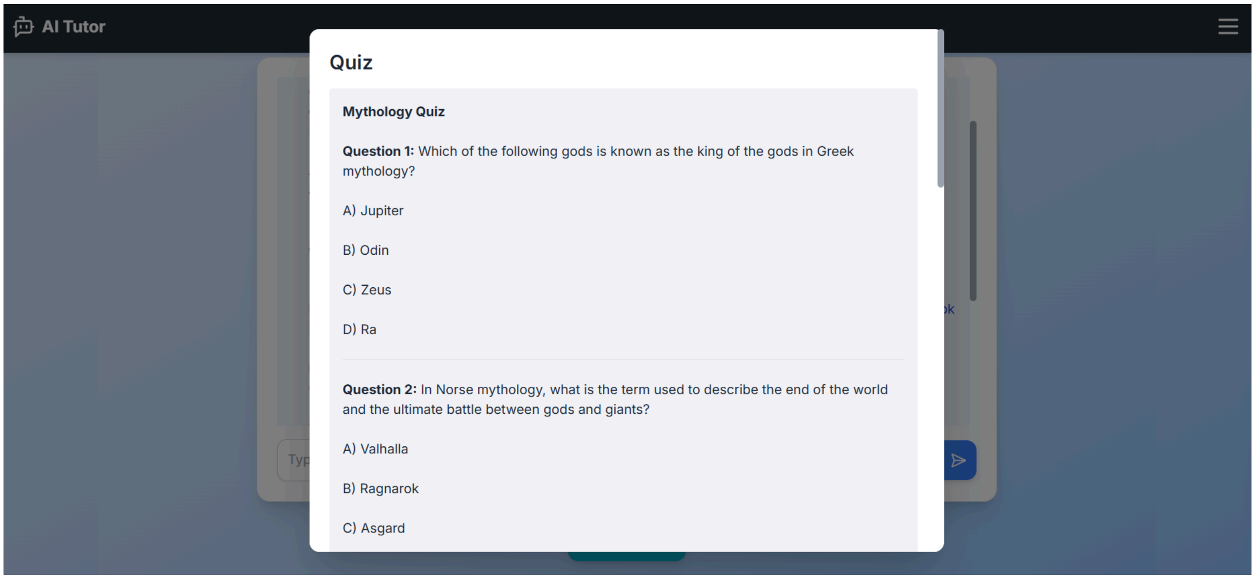
Mythology is a captivating tapestry woven from the stories, beliefs, and traditions of cultures around the globe. These ancient narratives reveal the mysteries of existence, explain the forces of nature, and delve into the human experience, reflecting our fears, hopes, and dreams. From the mighty gods and goddesses of Olympus to the trickster figures of indigenous folklore, mythology serves as a window into the values and worldviews of societies long past and still resonates with us today.

As we embark on this journey through time and culture, you might find yourself drawn to various aspects of mythology. Here are a few tantalizing topics to consider:

Greek and Roman Mythology: Explore the epic tales of gods like Zeus and Jupiter, heroic figures like Hercules and Aeneas, and the moral lessons embedded in their adventures.

Norse Mythology: Discover the rich world of Odin, Thor, and Loki, and the significance of the Ragnarok in the Viking imagination.

Casual Learning Chat Interface



Quiz

Mythology Quiz

Question 1: Which of the following gods is known as the king of the gods in Greek mythology?

A) Jupiter

B) Odin

C) Zeus

D) Ra

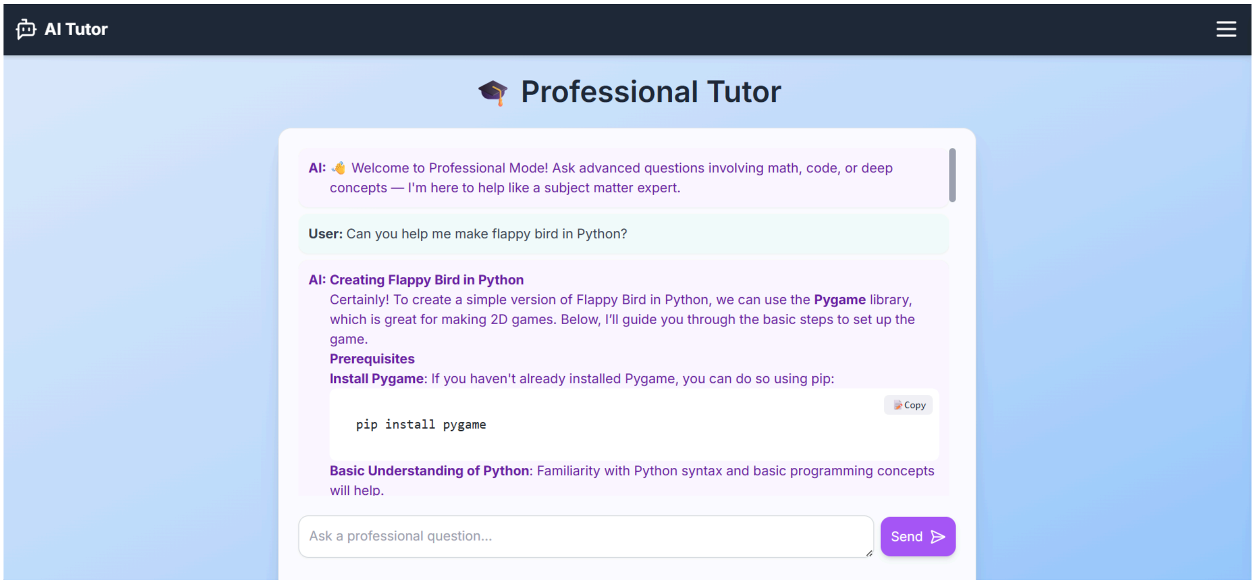
Question 2: In Norse mythology, what is the term used to describe the end of the world and the ultimate battle between gods and giants?

A) Valhalla

B) Ragnarok

C) Asgard

Quiz Interface



AI: 👋 Welcome to Professional Mode! Ask advanced questions involving math, code, or deep concepts — I'm here to help like a subject matter expert.

User: Can you help me make flappy bird in Python?

AI: Creating Flappy Bird in Python

Certainly! To create a simple version of Flappy Bird in Python, we can use the **Pygame** library, which is great for making 2D games. Below, I'll guide you through the basic steps to set up the game.

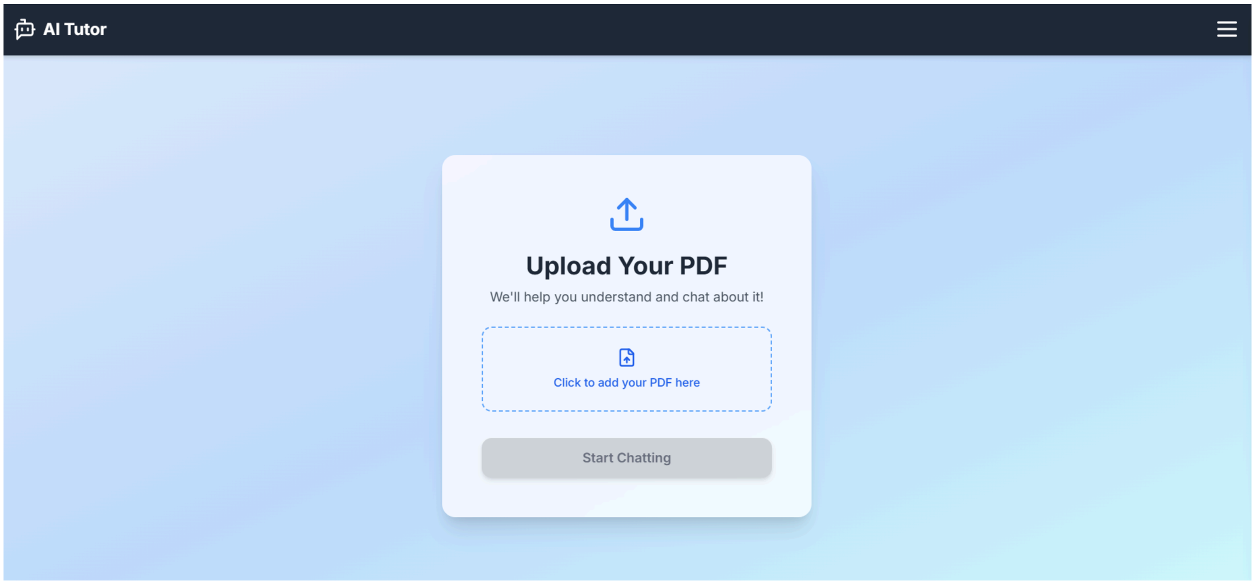
Prerequisites

Install Pygame: If you haven't already installed Pygame, you can do so using pip:

```
pip install pygame
```

Basic Understanding of Python: Familiarity with Python syntax and basic programming concepts will help.

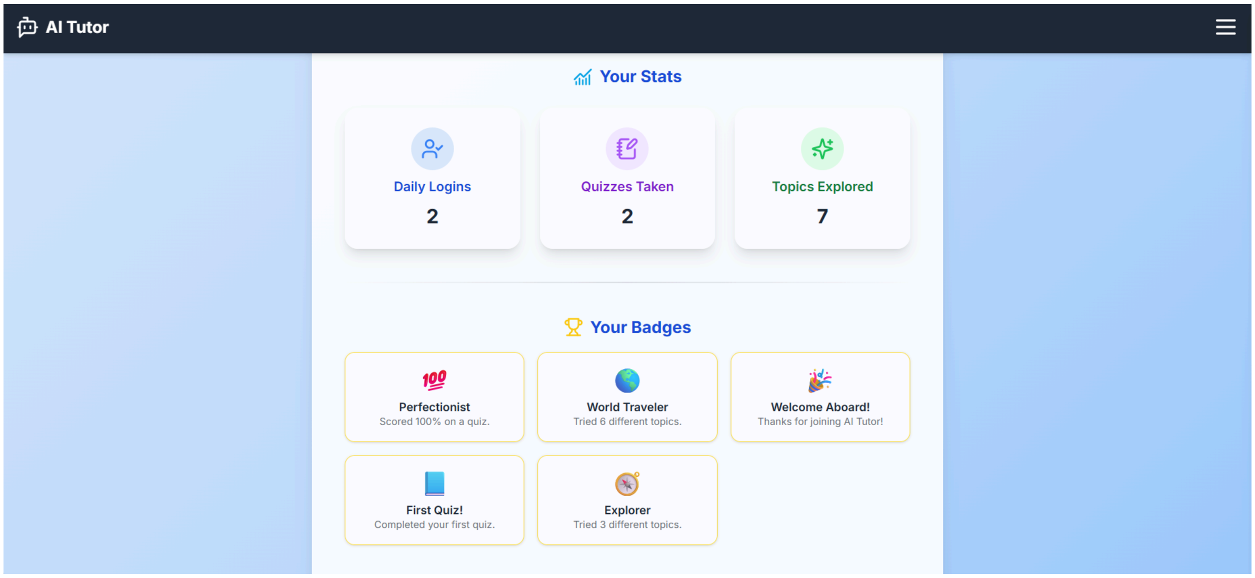
Professional Mode Interface



Upload Your PDF

We'll help you understand and chat about it!

PDF Upload Page



Your Stats

Daily Logins 2

Quizzes Taken 2

Topics Explored 7

Your Badges

💯 **Perfectionist** Scored 100% on a quiz.

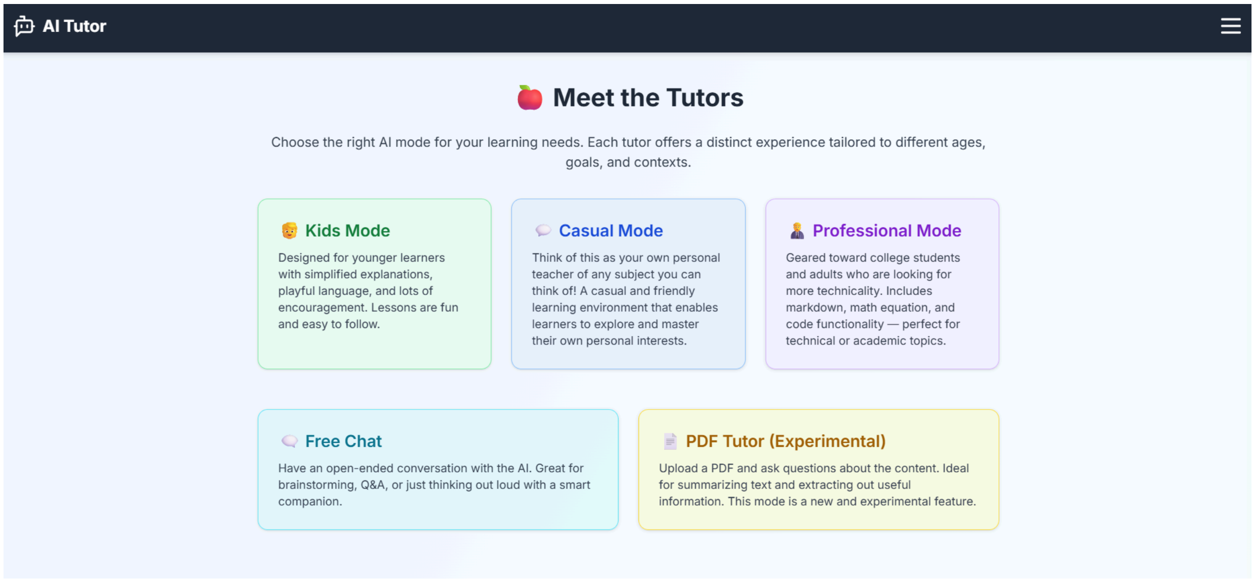
🌎 **World Traveler** Tried 6 different topics.

🎉 **Welcome Aboard!** Thanks for joining AI Tutor!

📘 **First Quiz!** Completed your first quiz.

🧭 **Explorer** Tried 3 different topics.

Profile Stats and Badges



🍎 Meet the Tutors

Choose the right AI mode for your learning needs. Each tutor offers a distinct experience tailored to different ages, goals, and contexts.

🧒 Kids Mode

Designed for younger learners with simplified explanations, playful language, and lots of encouragement. Lessons are fun and easy to follow.

💭 Casual Mode

Think of this as your own personal teacher of any subject you can think of! A casual and friendly learning environment that enables learners to explore and master their own personal interests.

👨‍💼 Professional Mode

Geared toward college students and adults who are looking for more technicality. Includes markdown, math equation, and code functionality — perfect for technical or academic topics.

🗨️ Free Chat

Have an open-ended conversation with the AI. Great for brainstorming, Q&A, or just thinking out loud with a smart companion.

📄 PDF Tutor (Experimental)

Upload a PDF and ask questions about the content. Ideal for summarizing text and extracting out useful information. This mode is a new and experimental feature.

About Page, Explanation of Tutor types

Video Demonstration

Youtube link for demo: https://www.youtube.com/watch?v=dWFv_ES8W_8

Quick note about what is covered

- Starts on welcome screen
- **I checked the connection at the very beginning. This is optional.** However, I prefer to do that before I use the tutor so I know that the server is awake. If you don't do this, you may have a longer loading screen when trying to use the tutor (as the server may be asleep)
- I show sign in and sign up pages, and log myself in
- I show my profile, with my stats and badges. Note the API usage number, it will change as I use the app
- I show Free Chat Mode
- I demo the "Other" functionality in Casual Mode, and choose "LangChain" as the topic
- I demo a mythology test in Casual Mode
- I demo kids mode (works the same as casual, just different context and tone)
- I demo professional mode, and ask it to make me a game in C++ and improve it
- Professional Mode features code copying, so code can be copy and pasted from the tutor for easy use. I demo this functionality.
- I show the about page
- I demo file uploading and PDF mode. I use the intro doc for Senior Project, as that is a good example of what this mode is intended for
- The tutor winds up giving a reasonably good summary of the Senior Project Requirements, and even tells me what documentation is needed.

In summary, all modes are functional, the UI is clean and intuitive, and the project as a whole functions as intended. If you would like to test any additional functionality or edge-cases, you can try them out yourself here! Just make sure you **sign up** with your own username and password.