Sprint Three

CS 4400 | AI Tutor

Spencer Thompson & Landon Towers

November 17, 2025

Table of Contents

1. Team Organization

- **Sprint Leader:** Spencer Thompson

1.1. Team 7

- Spencer Thompson
- Landon Towers

1.2. Meeting Schedule

- **Professor:** Thursdays at 11:15 AM
- **Sponsor:** Mondays at 2:00 PM
- **Team:** Ad-hoc, scheduled weekly

2. Project Description

The AI Tutor is a generative AI-powered learning assistant designed to support students at Utah Valley University (UVU). It provides a personalized and interactive learning experience by integrating with the Canvas Learning Management System (LMS). The application consists of a web-based frontend and a robust backend, working together to deliver a seamless and intuitive user experience.

The frontend is a modern chat interface built with Svelte. It allows students to ask questions and receive answers from an AI tutor in a conversational manner. The interface supports rich text formatting, including markdown, code snippets with syntax highlighting, and mathematical equations using LaTeX. This ensures that the AI can provide clear and easy-to-understand explanations for a wide range of subjects.

The backend is the core of the AI Tutor, developed using Python and the FastAPI framework. It orchestrates the entire system, managing user authentication, data processing, and communication with external services. The backend uses a MongoDB database to store user data, chat history, and course information.

One of the key features of the AI Tutor is its “smart chat” functionality. By securely accessing a student’s data from Canvas—including their enrolled courses, assignments, grades, and activity stream—the AI can provide context-aware and personalized assistance. For example, a student can ask for a summary of their upcoming assignments, clarification on a specific course topic, or help with a difficult concept from a lecture. The AI can also be customized by the user with a bio, to further personalize the experience.

To answer a wide range of questions, the AI Tutor is equipped with several tools. It can read the content of webpages, which is useful for providing explanations based on external resources. It can also execute Python code, allowing it to function as a calculator or to demonstrate programming concepts.

The AI Tutor is designed to be a safe and reliable learning environment. It includes a moderation service that filters out inappropriate content, and it uses a secure authentication system based on JSON Web Tokens (JWT) to protect user data. The application is also designed to be scalable and maintainable, with a containerized architecture using Docker.

In summary, the AI Tutor is a comprehensive and innovative learning platform that combines the power of generative AI with the rich data available in the Canvas LMS. It aims to provide UVU students with a powerful and accessible tool to enhance their learning experience and academic success.

3. Meeting Logs

3.1. Week 10

| What have you done since last team meeting?

- We managed to fix some issues with initializing a fresh instance of the database for development. This was something that was a pain point for a long time.
- We got the local development environment working!
- Additionally, a lot of bugs regarding the development environment and docker containers have been fixed.

| What obstacles are you encountering?

- Designing the infrastructure, has been somewhat difficult.

| What do you plan to accomplish by the next team meeting?

- We will need to have deployed our code to production.
- Getting the local environment working properly.

| Contributions

Team: 7	Sprint: 1	Date: 10/26/2025	Team Score: 100%
Name:	**Contribution (%):**	**Signature:**	**Individual Score:**
Spencer Thompson	50%	Spencer Thompson	100%
Landon Towers	50%	Landon Towers	100%

| Notes

- N/A

3.2. Week 11

| What have you done since last team meeting?

- We got the staging environment working! In addition, we added a bunch of new domains to test various different builds of the tutor on.
- We successfully deployed a new version of the AI Tutor with a variety of fixes as well as migrating to the new GPT-5 model.
- The new deployed Tutor also more accurately follows the behavior that the Tech Management Department wants it to.

| What obstacles are you encountering?

- Communication is difficult, particularly when discussing more technical aspects of the codebase.
- There are several different departments all connected to this project, all with different feedback, requirements, and wants. Providing the features, paperwork, and deliverables for all parties involved is incredibly difficult.

| What do you plan to accomplish by the next team meeting?

- Have official tests written and running.

| Contributions

Team: 7	Sprint: 1	Date: 11/2/2025	Team Score: 100%
Name:	**Contribution (%):**	**Signature:**	**Individual Score:**
Spencer Thompson	50%	Spencer Thompson	100%
Landon Towers	50%	Landon Towers	100%

| Notes

- It may be beneficial for our team to have a meeting to more specifically discuss how the project responsibilities will be split between team members.

3.3. Week 12

| What have you done since last team meeting?

- Landon wrote a suite of tests to properly test the AI Tutor.
- We met with the Tech Management department about some new requested features, and have the last set of features to be added planned.
- The bug regarding the Tutor not being able to see courses is fixed.

| What obstacles are you encountering?

- Determining a fair split of the workload is a tough problem to solve.
- There are several different departments all connected to this project, all with different feedback, requirements, and wants. Providing the features, paperwork, and deliverables for all parties involved is incredibly difficult.

| What do you plan to accomplish by the next team meeting?

- Have the ability to select which “Role” of the tutor to use as a button.
- Have automated tests running.

| Contributions

Team: 7	Sprint: 1	Date: 11/9/2025	Team Score: 100%
Name:	**Contribution (%):**	**Signature:**	**Individual Score:**
Spencer Thompson	50%	Spencer Thompson	100%
Landon Towers	50%	Landon Towers	100%

| Notes

- N/A

3.4. Week 13

| What have you done since last team meeting?

- The academic paper that the Tech Management Department is writing for the AI Tutor finished its first draft. So, I read that paper to verify it, and provide analytics data for it.
- Part of the analytics that we checked is lifetime total users, which is **652**, which is quite cool.
- We are about to merge the tests written by Landon into the staging branch, and merge our current set of features and changes into main.

| What obstacles are you encountering?

- The analytics containers that we are using seem to be broken. This is an important part of the project that needs to be fixed.

| What do you plan to accomplish by the next team meeting?

- Finish pull requests and merges.
- Fixed analytics
- Added UX buttons to select which “role” of the tutor to use.
- Have automated tests running.

| Contributions

Team: 7	Sprint: 1	Date: 11/16/2025	Team Score: 100%
Name:	**Contribution (%):**	**Signature:**	**Individual Score:**
Spencer Thompson	50%	Spencer Thompson	100%
Landon Towers	50%	Landon Towers	100%

| Notes

- N/A

4. Timeline

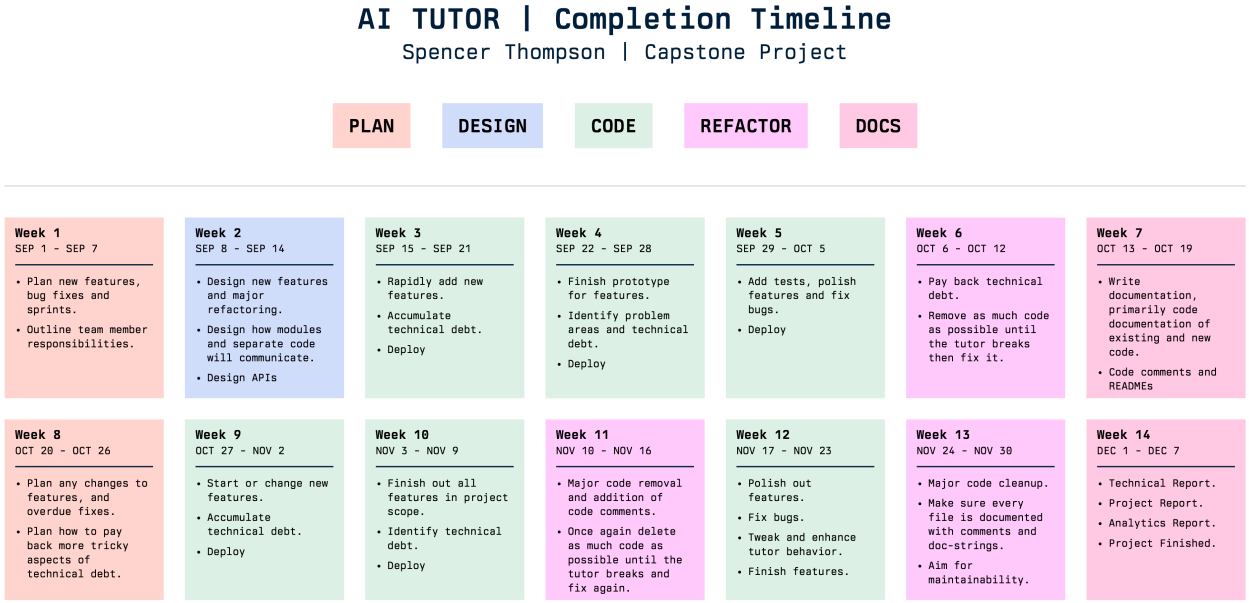


Figure 1: Completion Timeline

4.1. Backlogs

ID	Title	Type	Priority	Status	Owner	Sprint
ANA.1	Fix Analytics Server	Bug	1	Backlog	Spencer	3
ANA.2	Get Telemetry Data	Chore	2	In Progress	Spencer	3
BUG.1	Fix API Key Checking Error	Bug	1	Complete	Spencer	2
DEV.1	Fix Development Environment	Bug	2	Complete	Spencer	1
STG.1	Staging Server	Tests	2	Backlog	Landon	2
REQ.1	Revise FR / NFR Requirements	Tests	1	Backlog	Landon	2
REQ.2	Choose Which Requirements to Test	Tests	1	Backlog	Landon	2
TST.1	First Test Commit	Tests	1	Backlog	Landon	2
TST.2	Functional Requirement Tests	Tests	2	Backlog	Landon	3
TST.3	Non-Functional Requirement Tests	Tests	2	Backlog	Landon	3
TST.4	Automated Tests	Tests	3	In Progress	Landon	4
TST.5	Collect Data on Testing Metrics	Tests	2	In Progress	Landon	4
FTR.1	Migrate to GPT-5	Feature	2	Complete	Spencer	1
FTR.1	Button to choose Tutor Role	Feature	3	In Progress	Spencer	3
FTR.2	Migrate to GPT-5.1	Feature	2	In Progress	Spencer	3
DPL.1	Deploy New Features	Chore	1	Complete	Spencer	1
DPL.2	Deploy New Features	Chore	1	Complete	Spencer	2
DPL.3	Deploy New Features	Chore	1	Complete	Spencer	3
DOC.1	Add Cover Page to Sprints	Chore	1	Complete	Spencer	2
DOC.2	Fix Architecture Diagram	Chore	2	Backlog	Spencer	2

4.2. Sprints

5. Software Requirement Specification

5.1. Functional Requirements

5.1.1. User Management

- **F1.1:** Users must be able to log in to the system using their UVU credentials via Canvas OAuth2.
- **F1.2:** The system must use JSON Web Tokens (JWT) for authenticating API requests after the initial login.
- **F1.3:** Users shall be able to set a custom bio in their profile. The content of this bio will be included in the system prompt sent to the AI to influence its responses.
- **F1.4:** The system shall provide an administrative dashboard that displays the following real-time analytics: total number of users, number of active users, and total messages sent.

5.1.2. Chat Interface

- **F2.1:** The system shall provide a web-based chat interface for users to interact with the AI Tutor.
- **F2.2:** The chat interface must display the conversation history, with user and AI messages clearly distinguished.
- **F2.3:** The system shall stream messages to the client in real-time using websockets or a similar technology.
- **F2.4:** The chat interface shall render AI responses in GitHub-flavored markdown, supporting tables, lists, bold, italics, and other standard formatting.
- **F2.5:** The system must render mathematical equations formatted using LaTeX syntax.
- **F2.6:** The system must provide syntax highlighting for code snippets in Python, JavaScript, Java, C++, and SQL.

5.1.3. AI Tutor

- **F3.1:** The AI Tutor shall generate responses to user queries using a large language model (LLM) specified in the system configuration (e.g., GPT-4, Claude 3).
- **F3.2:** The system shall provide a “smart chat” feature that injects context from the user’s Canvas data into the LLM prompt to generate personalized responses.
- **F3.3:** The AI Tutor shall be able to retrieve the user’s enrolled courses, upcoming assignments, and recent grades from the Canvas API.
- **F3.4:** The AI Tutor shall have the ability to use the following tools to perform specific tasks:
  - **F3.4.1:** A web scraper tool that can read the full text content of a webpage given a URL.
  - **F3.4.2:** A Python code interpreter that can execute sandboxed Python code to perform calculations or demonstrate programming concepts.

- **F3.5:** All user input and AI-generated responses must be passed through a content moderation filter. Any content flagged as inappropriate (e.g., hate speech, violence) shall be blocked and logged.

5.2. Non-functional Requirements

5.2.1. Security

- **NF1.1:** All network communication between the client and server must be encrypted using TLS 1.2 or higher.
- **NF1.2:** The system must be protected against the OWASP Top 10 web vulnerabilities, which includes Cross-Site Scripting (XSS) and NoSQL Injection.
- **NF1.3:** Access to the backend API must be restricted to authenticated users with valid JWTs. Direct access via API keys is prohibited.
- **NF1.4:** All user data, including Canvas information and chat history, must be encrypted at rest in the database using AES-256 encryption.

5.2.2. Performance

- **NF2.1:** The user interface shall have a response time of less than 200ms for all user interactions (e.g., button clicks, page loads).
- **NF2.2:** The first token of a streamed AI response shall be delivered to the client in under 2 seconds, on average.
- **NF2.3:** The system shall initially support 100 concurrent users with an average API response time of under 500ms.

5.2.3. Usability

- **NF3.1:** A new user must be able to successfully send a message in “smart chat” mode within 60 seconds of their first login without requiring documentation.
- **NF3.2:** All error messages displayed to the user must include a unique error code and a clear, human-readable explanation of the problem.
- **NF3.3:** The AI Tutor’s responses shall achieve a Flesch-Kincaid grade level score between 8 and 12 to ensure they are understandable to a broad audience.

5.2.4. Scalability

- **NF4.1:** The system shall be horizontally scalable. An increase in container instances must result in a proportional increase in user capacity.
- **NF4.2:** The application shall be fully containerized using Docker, with all services defined in a `docker-compose.yml` file for automated deployment and scaling.

5.2.5. Reliability

- **NF5.1:** The system shall have a service availability of 99.9% (uptime).
- **NF5.2:** The system shall have a Mean Time To Recovery (MTTR) of less than 15 minutes.
- **NF5.3:** The content moderation filter shall have a false negative rate of less than 1% for clearly inappropriate content.

5.2.6. Maintainability

- **NF6.1:** All Python code must adhere to the PEP 8 style guide, enforced by an automated linter.
- **NF6.2:** The application shall have a modular design, with a clear separation of concerns between the data, business logic, and presentation layers.
- **NF6.3:** All new code committed to the main branch must have a minimum of 80% unit test coverage.

6. Addressing Feedback

7. Design

- **Authenticate with UVU ID:** The user logs into the system. The system initiates an OAuth2 flow with the external Canvas LMS, which handles the actual authentication using the user's UVU credentials.
- **Fetch User Profile:** The user retrieves their current profile information from the system, such as their name and custom bio, which is then displayed in the application's settings interface.
- **Update User Profile:** The user modifies and saves their profile information (specifically their bio) to personalize the AI's tone and responses.
- **List Conversations:** The user's client application requests and displays a list of all their past chat sessions, allowing them to select a previous conversation for review.
- **Retrieve Conversation:** After a user selects a conversation from the list, the client application retrieves the full message history for that specific chat session from the server.
- **Initiate Smart Chat:** The user starts a new conversation in "smart chat" mode. This is a distinct action that tells the backend to inject context from the user's Canvas data into the AI's prompt.
- **Send Chat Message:** The user sends a query to the AI Tutor. This is the most common action. This use case can be extended with specialized tools if the query requires them.
- **Access Course Info:** When a "Smart Chat" is active, the system automatically fetches the user's course information, assignments, and other relevant data from the Canvas API to provide context for the AI's response. This is a system action triggered by the user's chat message.
- **Fetch Web Content:** As an extension of "Send Chat Message," if the user's query contains a URL, the AI can use this tool to access the content of that webpage to inform its answer.
- **Execute Python Code:** As an extension of "Send Chat Message," the AI can use this tool to write and run Python code in a secure, sandboxed environment to perform calculations, run algorithms, or demonstrate programming concepts.
- **View System Analytics:** The Administrator accesses a separate analytics dashboard (Plausible) to view aggregated, anonymized usage data and system performance metrics.

7.1. Architecture Diagram

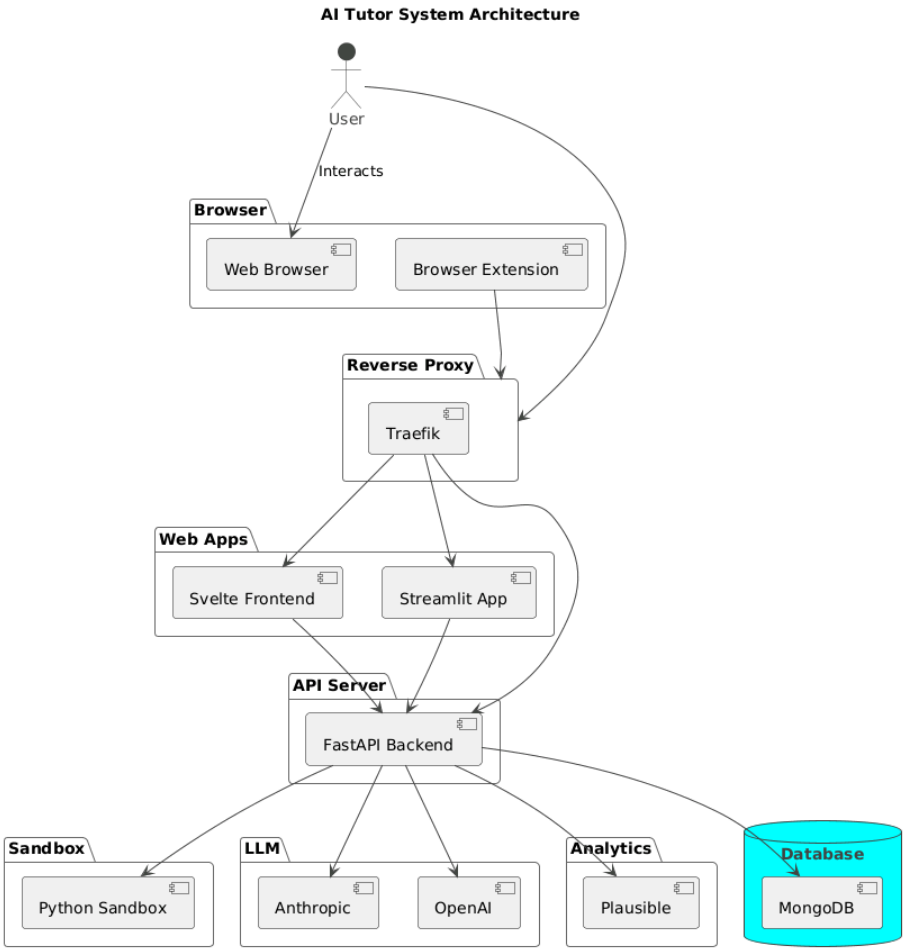


Figure 2: Architecture Diagram

7.2. Use Case Diagram

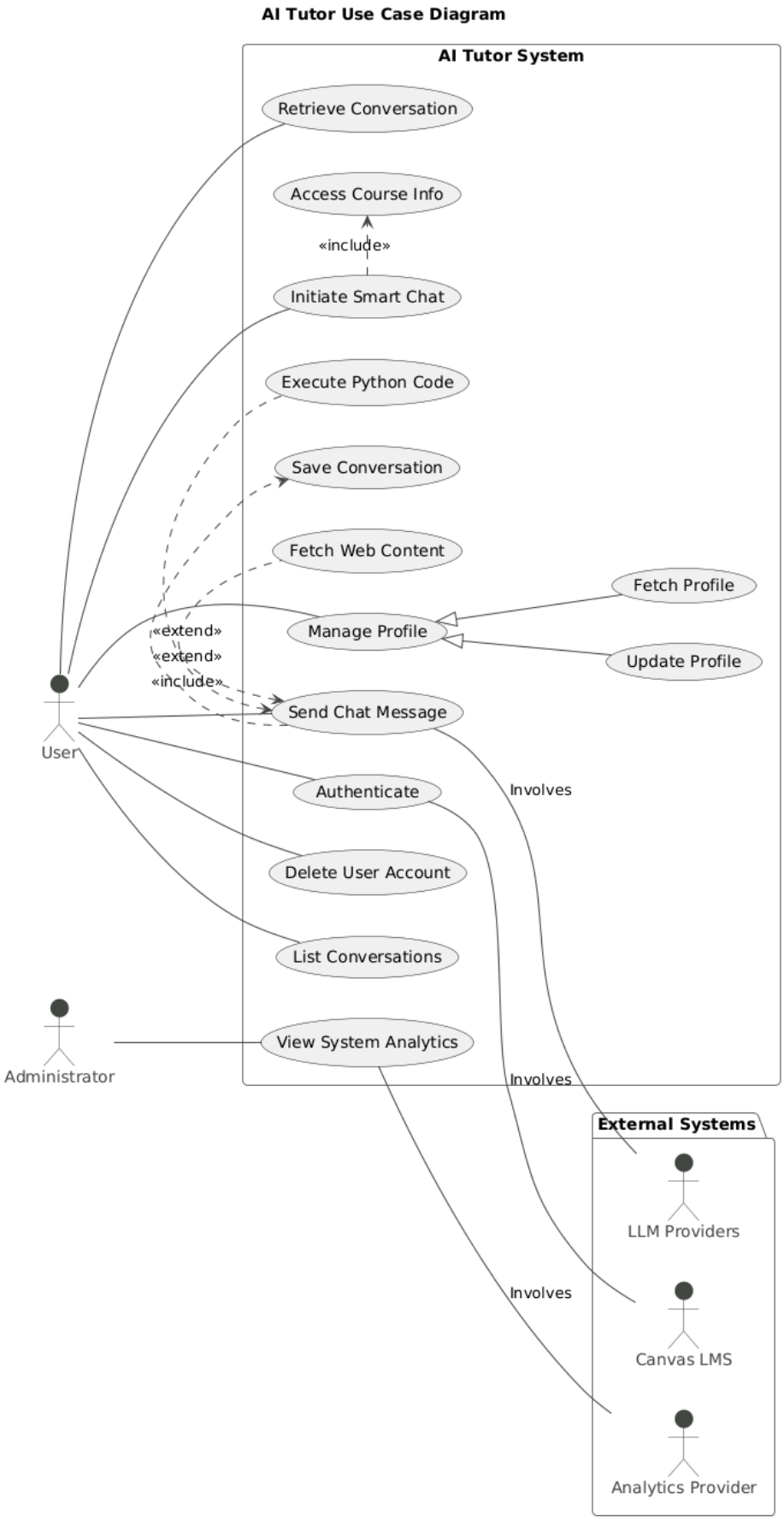


Figure 3: Use Case Diagram

7.3. Class Diagram

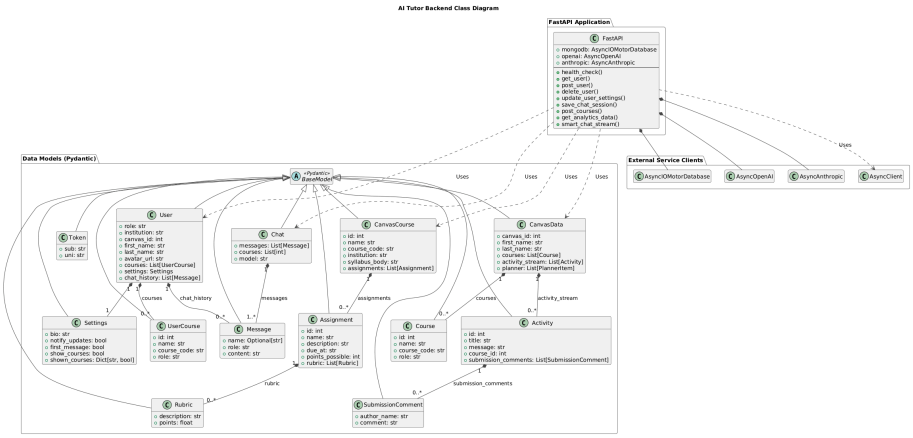


Figure 4: Class Diagram

7.4. Other Diagrams

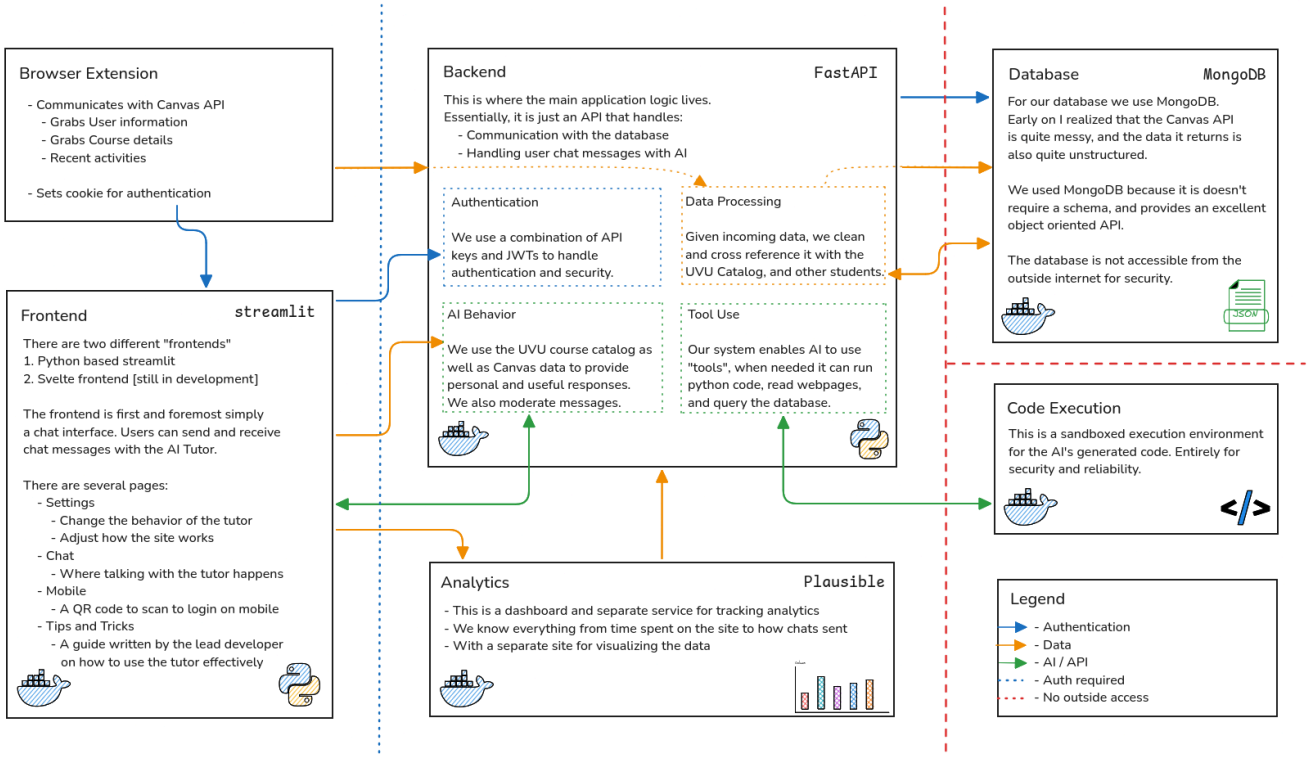


Figure 5: Architecture Diagram

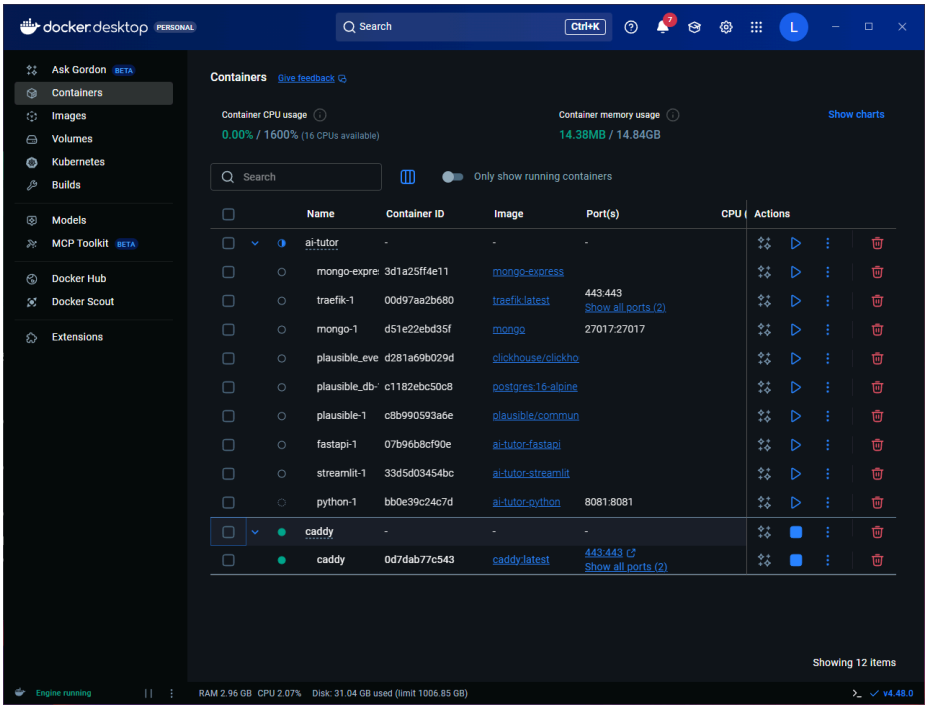


Figure 6: AI Tutor Running in Docker

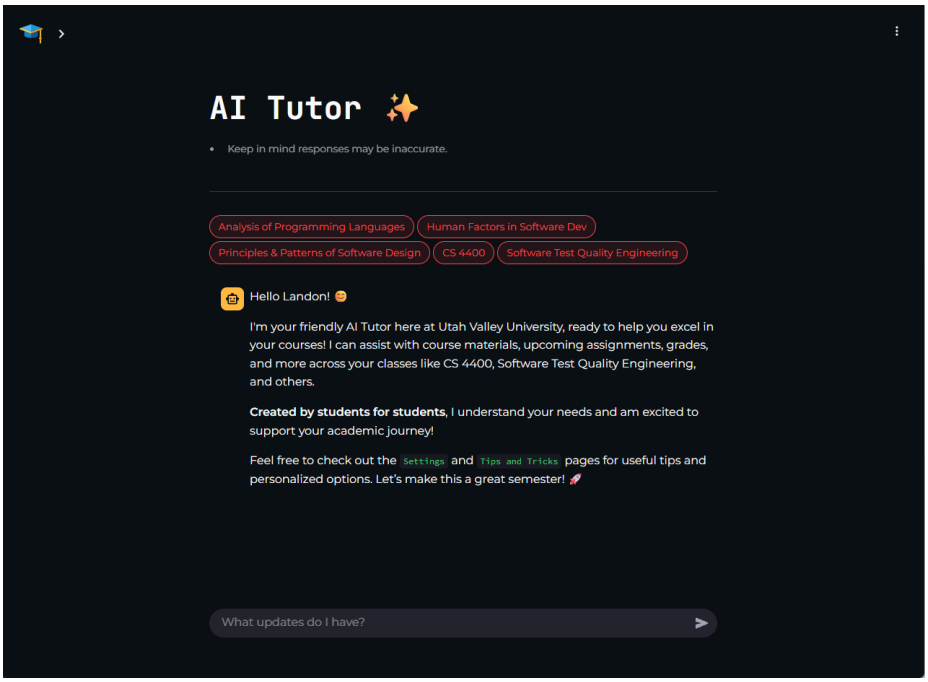


Figure 7: AI Tutor Main Page

8. Demo

8.1. Video

- The Demo Video is in Google Drive here:
  - https://drive.google.com/drive/folders/1IO0jYRLZXh_n9kpROqcBzJVIzEC9H6kJ?usp=drive_link

8.2. Code

```
async def openai_responses_api_iter(
    messages: List[Message],
    descriptions: str = "",
    context: dict = dict(),
    recursive=False,
):
    """
    New Responses API compatible with current code.
    Currently only uses GPT-5 and variants.
    """
    time_start = time.perf_counter()

    # filter out the "name" parameter
    messages = [{k: dict(m)[k] for k in ("role", "content")} for m in messages]
    first_message = messages[-1]

    mod = await moderate(first_message["content"])  # NOTE: this is a bit messy
    if mod:
        yield mod
        return

    time_after_moderation = time.perf_counter()

    course_descriptions = f"## Courses\n\nThese are the student's courses: \n{descriptions}" if descriptions else ""

    if user := context.get("user"):
        bio = user.get("bio")
        if bio is None:
            bio = ""
    else:
        bio = ""

    agent_router = await openai.responses.create(
        model="gpt-5-nano",
        input=messages,
        tools=[t.get("tool") for t in agent_tools.values()],
        instructions=f"""
        Given an input message, determine which agent would most effectively help the user.
        Keep in mind you are providing these parameters for yourself in a future response.

        {course_descriptions}
        """,
        reasoning={"effort": "minimal"},
        tool_choice="required",
        store=False,  # retrieve later, defaults to true
    )

    route = ""
    for output in agent_router.output:
        if output.type == "function_call":
            route = json.loads(output.arguments)
            # TODO: Get tokens used

    if route["agent"] == "Tutor":
        print(f"Using Agent: {route['agent']}")
        system_message = general_system_message(bio, descriptions) + tutor_system_message()
    else:
        system_message = general_system_message(bio, descriptions)

    print(f"Using Reasoning: {route['reasoning']}")
    print(f"Using Verbosity: {route['verbosity']}")
    print()

    stream = await openai.responses.create(
        model="gpt-5-nano",
        tools=[t.get("tool") for t in tools.values()],
        # tools=[t.get("tool") for t in tools.values()] + [{"type": "web_search"}],
        tool_choice="auto",
        input=messages,
        stream=True,
        instructions=system_message,
        reasoning={"effort": route["reasoning"]},
        text={"verbosity": route["verbosity"]},
        safety_identifier="Spencer",  # should be hashed string, incase they do something crazy
        store=False,  # retrieve later, defaults to true
    )

    async for event in stream:
        # print(event.type)
        if event.type == "response.created":
            time_response_created = time.perf_counter()

        if event.type == "response.output_text.delta":
            # print(event.delta, flush=True, end="")
            yield event.delta

        if event.type == "response.function_call_arguments.done":
            print("Calling tool")
            print(event.arguments)

        if event.type == "response.completed":
            time_response_completed = time.perf_counter()

            tokens_used = {
                "input_tokens": event.response.usage.input_tokens,
                "output_tokens": event.response.usage.output_tokens,
                "reasoning_tokens": event.response.usage.output_tokens_details.reasoning_tokens,
            }

    print()
    print()
    print(f"Start to end: {time_response_completed - time_start}")
    print(f"created to end: {time_response_completed - time_response_created}")

    # TODO: add recursive call?
    # Previously this was done on tool calls only
```

Listing 1: Code to Migrate to GPT-5

9. Testing

- Our strategies and procedures to ensure the AI Tutor application is reliable, functional, and secure. The plan covers multiple levels of testing, from individual components to the system as a whole.

9.1. Scope

This plan applies to all components of the AI Tutor project, including:

- **Backend:** The Python FastAPI application, including all API endpoints, business logic, and interactions with the database and external services (Canvas, LLMs).
- **Frontend:** The Svelte-based web interface, including UI components, state management, and user interactions.
- **Infrastructure:** The Docker-based containerization and deployment configuration.

9.1.1. Unit Testing

- **Objective:** Verify individual functions and components work correctly in isolation.
- **Backend (Python):** Each function in the FastAPI application will have corresponding unit tests. Mocking will be used to isolate components from external dependencies like the database or the Canvas API.
  - **Tools:** `pytest`

9.1.2. Integration Testing

- **Objective:** To verify that different parts of the application work together as intended.
- **Backend:** Tests will focus on the interaction between API endpoints and the MongoDB database. For example, ensuring that creating a user via an endpoint correctly stores the user in the database.
  - **Tools:** `pytest`, `TestClient` for FastAPI
- **Frontend-Backend:** Tests will verify that the frontend can successfully make API calls to the backend and handle the responses correctly. This will involve running both the frontend and backend servers in a test environment.
  - **Tools:** `Playwright` or `Cypress`

9.1.3. End-to-End (E2E) Testing

- **Objective:** To simulate real user scenarios from start to finish, ensuring the entire application flow is working correctly.
- **Scenarios:** Key user journeys will be tested, such as:
  1. User logs in via Canvas, asks a question in Smart Chat, and receives a context-aware answer.
  2. User updates their bio, starts a new chat, and verifies the AI's tone has changed.
  3. User asks a question that requires executing Python code.
- **Tools:** `Playwright`

9.1.4. Security Testing

- **Objective:** To identify and mitigate potential security vulnerabilities.
- **Methods:**
  - Regular dependency scanning to find known vulnerabilities in third-party packages.
  - Manual penetration testing of key endpoints, focusing on authentication (JWT handling) and input validation to prevent injection attacks.
  - Ensuring all sensitive data is handled securely and API keys are not exposed.

9.2. Roles and Responsibilities

- **All Team Members:** Responsible for writing unit tests for the code they develop.
- **Sprint Lead:** Responsible for overseeing the testing process, running integration and E2E tests before a release, and managing bug reports.

9.3. Metrics and Quality Assurance

- **Code Coverage:** Aim for a minimum of 80% unit test coverage for the backend Python code, measured using `pytest-cov`.
- **Bug Tracking:** All bugs will be logged as GitHub Issues. Each bug report will include a description, steps to reproduce, priority, and severity.
- **Pass/Fail Criteria:** For a sprint to be considered complete, all unit and integration tests must pass, and there should be no outstanding critical or high-priority bugs.

9.4. Test Cases

- This section provides a sample of test cases derived from the testing plan. The `Test Date` is set to the creation date and `Pass/Fail` status is pending execution.

9.4.1. Unit Test Cases (Backend)

Test Case #	Description	Test Date	Inputs	Expected Outputs	Pass/Fail
UT-BE-01	**User Model Validation:** Tests that the `User` Pydantic model in `models.py` successfully validates a correct user object.	2025-10-05	`{"sub": "123", "name": "Test User"}`	The `User` model is instantiated without errors.	Pending
UT-BE-02	**Invalid User Model:** Tests that the `User` model raises a `ValidationError` if required fields are missing.	2025-10-05	`{"sub": "123"}` (missing `name`)	A `pydantic.ValidationError` is raised.	Pending
UT-BE-03	**AI Response Formatting:** Tests a utility function in `ai.py` that formats a raw LLM response into the correct JSON structure for the frontend.	2025-10-05	Raw text string from a mocked LLM.	A valid JSON object with `type` and `content` fields is returned.	Pending

9.4.2. Unit Test Cases (Frontend)

Test Case #	Description	Test Date	Inputs	Expected Outputs	Pass/Fail
UT-FE-01	**Message Component Rendering:** Verifies that the `Message.svelte` component correctly displays the message text and author.	2025-10-05	`props = { author: "AI", text: "Hello!" }`	The component renders a `div` containing the text "Hello!" and indicates it is from the "AI".	Pending
UT-FE-02	**Send Button Event:** Verifies that clicking the "Send" button in the `ChatInput.svelte` component dispatches a `send` event.	2025-10-05	User clicks the send button.	A `send` event is dispatched with the input field's text content as the payload.	Pending

9.4.3. Integration Test Cases

Test Case #	Description	Test Date	Inputs	Expected Outputs	Pass/Fail
IT-BE-01	**Get User Profile:** Verifies the `/api/users/me` endpoint.	2025-10-05	`GET` request to `/api/users/me` with a valid JWT for a user stored in the test database.	A `200 OK` response is returned with a JSON body containing the correct user's profile data.	Pending
IT-BE-02	**Create Conversation:** Verifies the `/api/conversations/` endpoint.	2025-10-05	`POST` request to `/api/conversations/` with a valid JWT and a title.	A `201 Created` response is returned, and a new conversation document is created in the database for that user.	Pending

9.4.4. End-to-End Test Cases

Test Case #	Description	Test Date	Inputs	Expected Outputs	Pass/Fail
E2E-01	**Full Login and Chat Flow:** Simulates a user logging in, sending a message, and receiving a response.	2025-10-05	1. Navigate to home page. 2. Click "Login". 3. Complete mock Canvas login. 4. Type "Hello" into chat. 5. Click "Send".	1. User is redirected to Canvas. 2. User is redirected back to the app. 3. The message "Hello" appears in the chat window. 4. An AI response is streamed into the chat window.	Pending

9.4.5. Security Test Cases

Test Case #	Description	Test Date	Inputs	Expected Outputs	Pass/Fail
ST-01	**Unauthorized API Access:** Attempt to access an authenticated endpoint without proper credentials.	2025-10-05	`GET` request to `/api/users/me` with no `Authorization` header.	The API returns a `401 Unauthorized` or `403 Forbidden` status code.	Pending
ST-02	**XSS in Chat Input:** Attempt to inject a script into the chat.	2025-10-05	User types `<script>alert('XSS')</script>` into the chat input and sends.	The message is displayed as plain text in the chat window, and no alert dialog appears. The script is sanitized.	Pending

- Bug Report
  - **API Key Check Failure:** After a merged pull request, the backend code was broken due to the `check_api_key` function breaking on the change. This was reverted and fixed.
  - **AI Tutor Behavior:** Not necessarily a discrete bug, but the LLM is not producing responses in accordance with the desires of the Tech Management Department, our project sponsor.

    - **LLM Tool Code Execution Container**: The docker container that is used to sandbox code generated by the tutor to execute is not working properly and breaking often.
    - **Course Name**: In the Canvas API, the `course_name` parameter can be changed by users, and we need to migrate from `course_name` to `course_code` so that we have more predictable behavior from the LLM. This parameter is used as a pseudo-primary key by MongoDB.
- Assessment Report

  This section will contain an analysis of the testing results, including metrics trends and an overall assessment of application quality. This report will be generated after the initial testing cycles are complete.

9.5. Preventive QA

- **Static Code Analysis:** Automated tools will be used to analyze the source code without executing it. This helps to identify potential vulnerabilities, bugs, and code smells early in the development process.
    - **Tools:** `ruff`, `bandit`
- **Code Reviews:** All new code will be reviewed by at least one other team member before it is merged into the main branch. This helps to ensure code quality, consistency, and knowledge sharing among the team.
- **Coding Standards:** The team will adhere to a consistent set of coding standards (e.g., PEP 8 for Python) to ensure that the codebase is readable and maintainable.

9.6. Design and Maintenance Metrics

9.6.1. Design Metrics

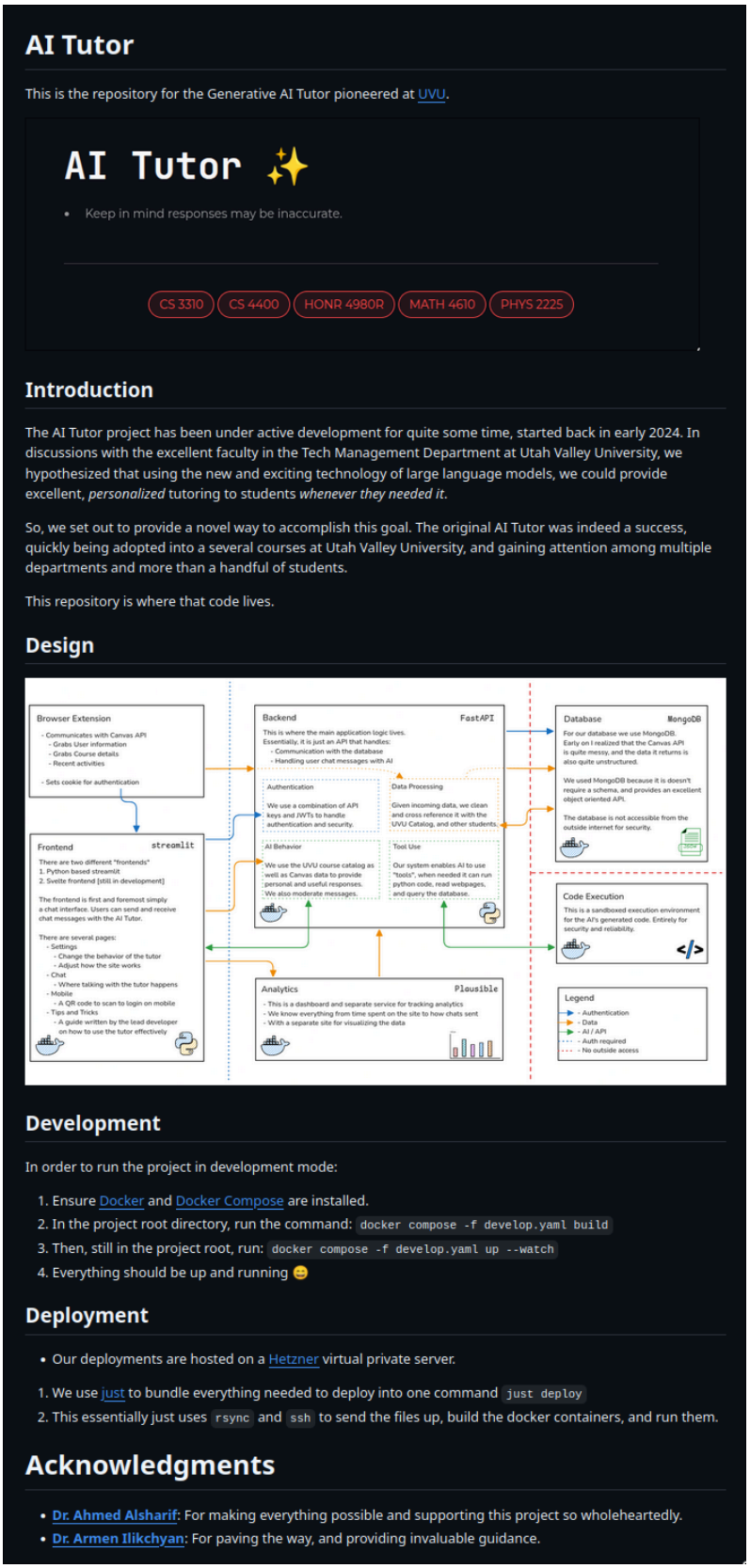
- **Cyclomatic Complexity:** This metric measures the complexity of a function's decision-making logic. A high cyclomatic complexity can indicate that a function is difficult to test and maintain. The target is to keep the cyclomatic complexity of all functions below 10.
- **Coupling:** This metric measures the degree of interdependence between modules. Low coupling is desirable, as it makes it easier to modify one module without affecting others.
- **Cohesion:** This metric measures how closely related the responsibilities of a single module are. High cohesion is desirable, as it indicates that a module has a well-defined purpose.

9.6.2. Maintenance Metrics

- **Mean Time To Repair (MTTR):** This metric measures the average time it takes to fix a bug, from the time it is reported to the time a fix is deployed. The target MTTR for critical bugs is less than 24 hours.
- **Mean Time Between Failures (MTBF):** This metric measures the average time between system failures. A high MTBF indicates a reliable system. The target MTBF is greater than 1000 hours.
- **Code Churn:** This metric measures the number of times a file is modified. A high code churn can indicate that a file is a "hotspot" in the codebase and may be a candidate for refactoring.

9.7. Quality Assurance Metrics

10. README



AI Tutor

This is the repository for the Generative AI Tutor pioneered at UVU.

AI Tutor ✨

- Keep in mind responses may be inaccurate.

CS 3310 CS 4400 HONR 4980R MATH 4610 PHYS 2225

Introduction

The AI Tutor project has been under active development for quite some time, started back in early 2024. In discussions with the excellent faculty in the Tech Management Department at Utah Valley University, we hypothesized that using the new and exciting technology of large language models, we could provide excellent, *personalized* tutoring to students *whenever they needed it*.

So, we set out to provide a novel way to accomplish this goal. The original AI Tutor was indeed a success, quickly being adopted into a several courses at Utah Valley University, and gaining attention among multiple departments and more than a handful of students.

This repository is where that code lives.

Design

Development

In order to run the project in development mode:

1. Ensure Docker and Docker Compose are installed.
2. In the project root directory, run the command: `docker compose -f develop.yaml build`
3. Then, still in the project root, run: `docker compose -f develop.yaml up --watch`
4. Everything should be up and running 😄

Deployment

- Our deployments are hosted on a Hetzner virtual private server.

1. We use just to bundle everything needed to deploy into one command `just deploy`
2. This essentially just uses `rsync` and `ssh` to send the files up, build the docker containers, and run them.

Acknowledgments

- **Dr. Ahmed Alsharif**: For making everything possible and supporting this project so wholeheartedly.
- **Dr. Armen Ilikchyan**: For paving the way, and providing invaluable guidance.

Figure 8: Screenshot of the `README.md` from Github