Sprint Two

CS 4400 | AI Tutor

Spencer Thompson & Landon Towers

October 29, 2025

Table of Contents

1. Meeting Logs

1.1. Week 7

| What have you done since last team meeting?

- We were able to meet with the tech management department to find their concerns as well as nail down the features they want the tutor to have.

| What obstacles are you encountering?

- Given that this project is related to several different departments and people at UVU, it is difficult to do everything we need to keep everyone happy.

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

- Deploy some new code.

| Contributions

Team: 7	Sprint: 1	Date: 10/5/2025	Team Score: 100%
Name:	**Contribution (%):**	**Signature:**	**Individual Score:**
Spencer Thompson	50%	Spencer Thompson	100%
Landon Towers	50%	Landon Towers	100%

| Notes

- N/A

1.2. Week 8

What have you done since last team meeting?

- We got a new virtual machine for staging the environment.
- We also managed to plan out some next steps regarding features such as:
  - Migrating to GPT-5
  - Fixing some issues with database initialization
  - Moving to a combination of Sqlite and MongoDB

What obstacles are you encountering?

- We are having issues running the local development environment on Landon's computer. This could be an issue with the existing code, so I will look into it.

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

- Hopefully get the local development environment working on everyone's computers.

Contributions

Team: 7	Sprint: 1	Date: 10/12/2025	Team Score: 100%
Name:	**Contribution (%):**	**Signature:**	**Individual Score:**
Spencer Thompson	50%	Spencer Thompson	100%
Landon Towers	50%	Landon Towers	100%

Notes

- N/A

1.3. Week 9

| What have you done since last team meeting?

- I fixed a lot of issues regarding the local development environment. We are using docker and docker compose for development and deployment, so I just cloned down the repository and fixed a lot of issues regarding the local environment.
- I successfully managed to migrate from `gpt-4o` to `gpt-5` which was a rather large refactor/addition.

| What obstacles are you encountering?

- Getting the local development environment is still not working for Landon which is quite frustrating.

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

- Getting the MongoDB database to initialize properly for development/staging and deployment.
- Getting the local environment working properly.

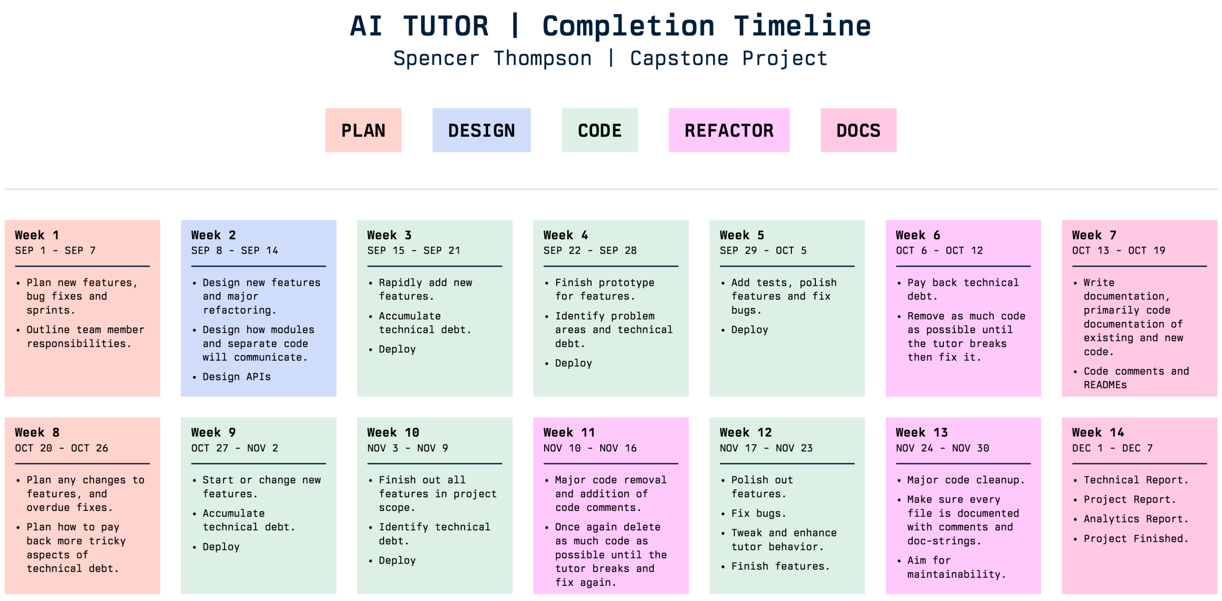
| Contributions

Team: 7	Sprint: 1	Date: 10/19/2025	Team Score: 100%
Name:	**Contribution (%):**	**Signature:**	**Individual Score:**
Spencer Thompson	50%	Spencer Thompson	100%
Landon Towers	50%	Landon Towers	100%

| Notes

- N/A

2. Backlogs and Sprints



AI TUTOR | Completion Timeline

Spencer Thompson | Capstone Project

PLAN DESIGN CODE REFACTOR DOCS

Week 1
SEP 1 - SEP 7

- Plan new features, bug fixes and sprints.
- Outline team member responsibilities.

Week 2
SEP 8 - SEP 14

- Design new features and major refactoring.
- Design how modules and separate code will communicate.
- Design APIs

Week 3
SEP 15 - SEP 21

- Rapidly add new features.
- Accumulate technical debt.
- Deploy

Week 4
SEP 22 - SEP 28

- Finish prototype for features.
- Identify problem areas and technical debt.
- Deploy

Week 5
SEP 29 - OCT 5

- Add tests, polish features and fix bugs.
- Deploy

Week 6
OCT 6 - OCT 12

- Pay back technical debt.
- Remove as much code as possible until the tutor breaks then fix it.

Week 7
OCT 13 - OCT 19

- Write documentation, primarily code documentation of existing and new code.
- Code comments and READMEs

Week 8
OCT 20 - OCT 26

- Plan any changes to features, and overdue fixes.
- Plan how to pay back more tricky aspects of technical debt.

Week 9
OCT 27 - NOV 2

- Start or change new features.
- Accumulate technical debt.
- Deploy

Week 10
NOV 3 - NOV 9

- Finish out all features in project scope.
- Identify technical debt.
- Deploy

Week 11
NOV 10 - NOV 16

- Major code removal and addition of code comments.
- Once again delete as much code as possible until the tutor breaks and fix again.

Week 12
NOV 17 - NOV 23

- Polish out features.
- Fix bugs.
- Tweak and enhance tutor behavior.
- Finish features.

Week 13
NOV 24 - NOV 30

- Major code cleanup.
- Make sure every file is documented with comments and doc-strings.
- Aim for maintainability.

Week 14
DEC 1 - DEC 7

- Technical Report.
- Project Report.
- Analytics Report.
- Project Finished.

Figure 1: Completion Timeline

3. Software Requirement Specification

3.1. Functional Requirements

3.1.1. 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.

3.1.2. 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.

3.1.3. 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).
- **F3.3:** 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.
- **F3.4:** The AI Tutor shall be able to retrieve the user's enrolled courses, upcoming assignments, and recent grades from the Canvas API.
- **F3.5:** The AI Tutor shall have the ability to use the following tools to perform specific tasks:
  - **F3.5.1:** A web scraper tool that can read the full text content of a webpage given a URL.
  - **F3.5.2:** A Python code interpreter that can execute sandboxed Python code to perform calculations or demonstrate programming concepts.
- **F3.6:** 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.

3.2. Non-functional Requirements

3.2.1. 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.

3.2.2. 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.

3.2.3. 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.

3.2.4. 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.

3.2.5. 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.

3.2.6. 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.

4. Addressing Feedback

5. Design

5.1. Architecture Diagram

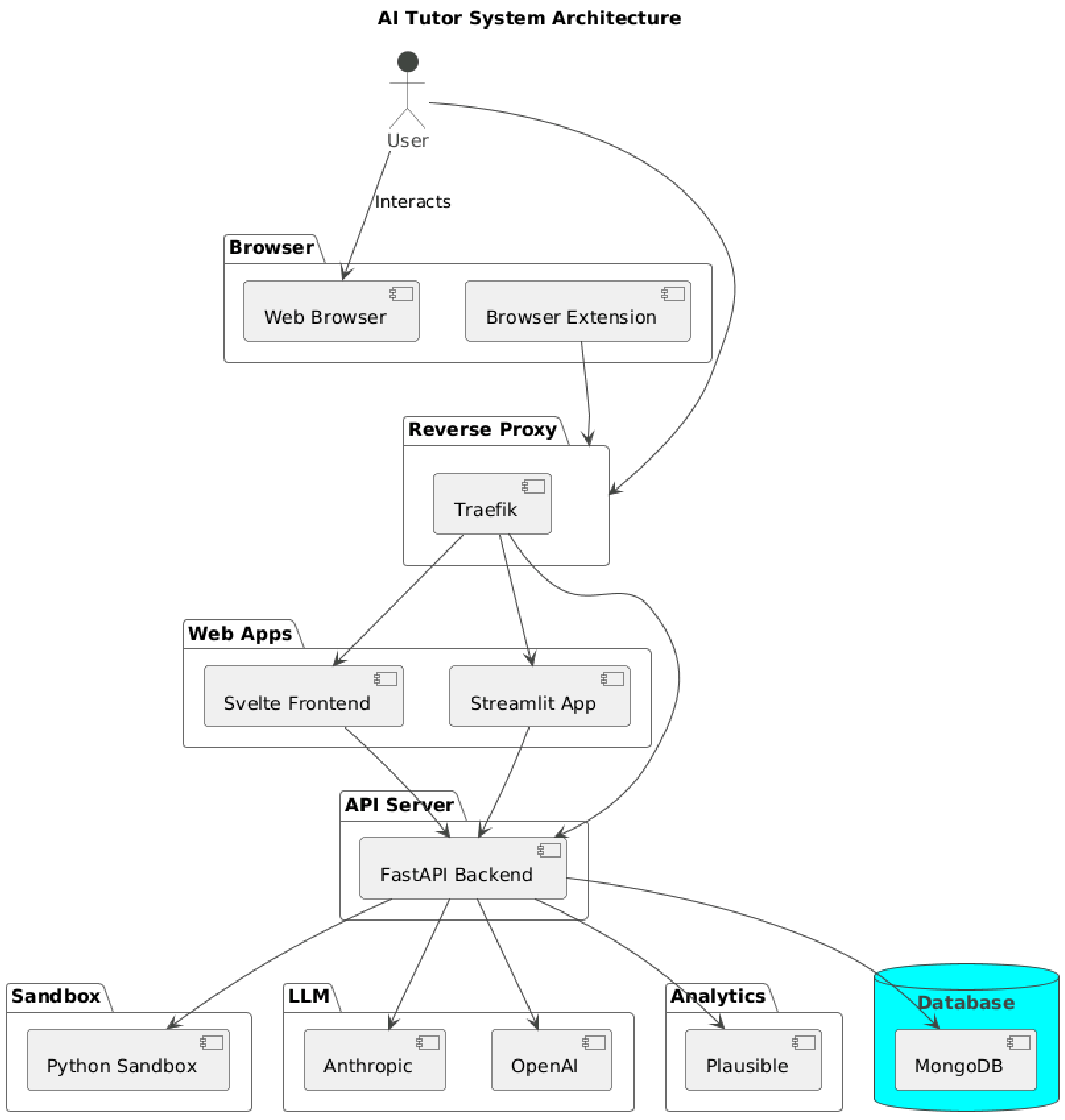


Figure 2: Architecture Diagram

5.2. Use Case Diagram

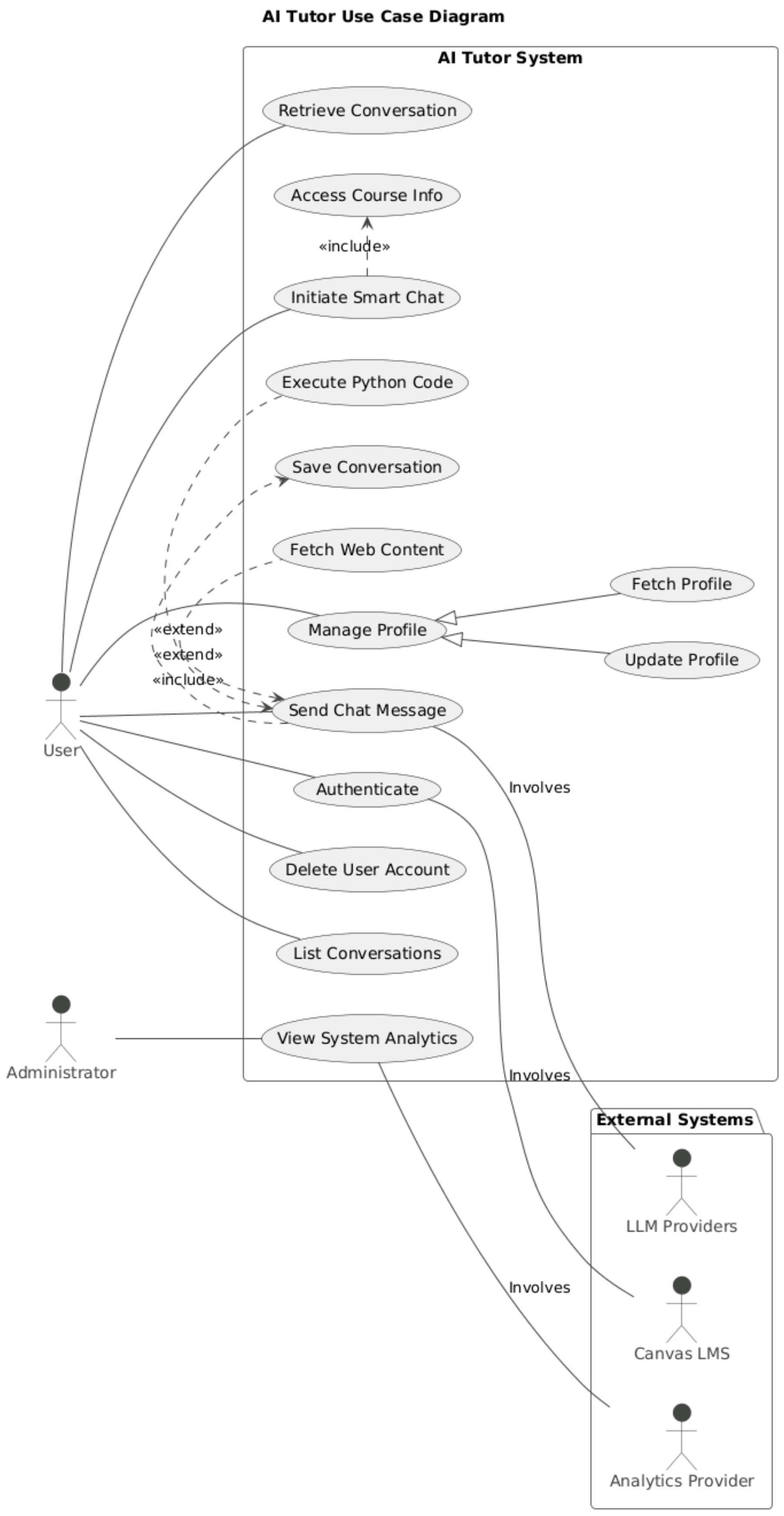


Figure 3: Use Case Diagram

5.3. Class Diagram

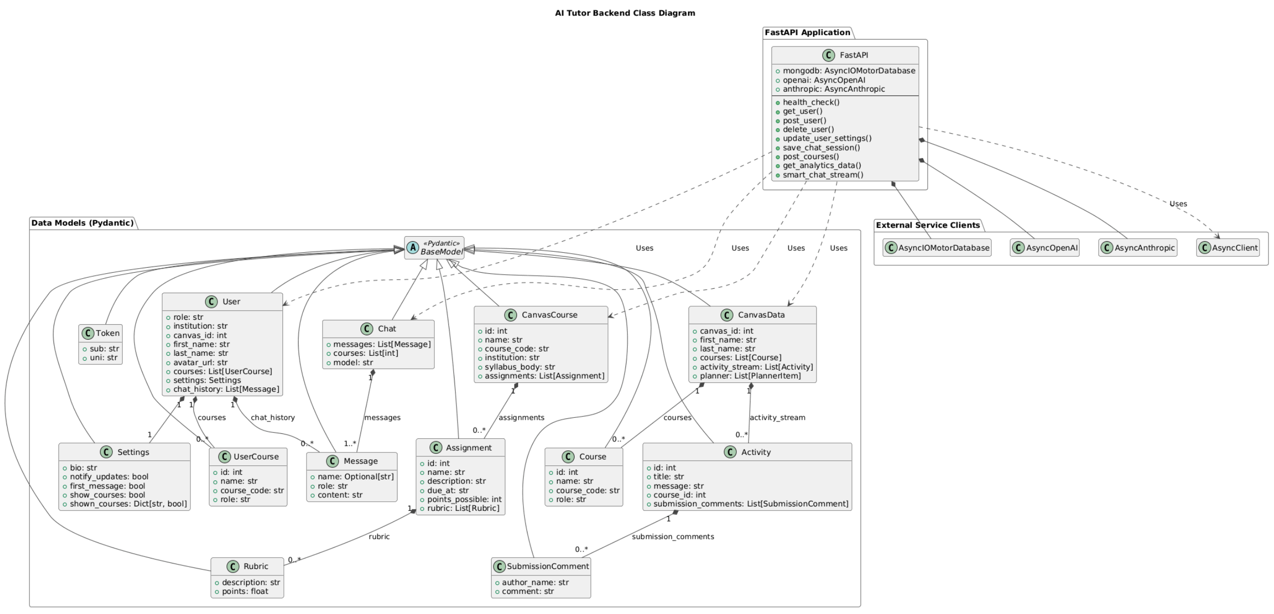
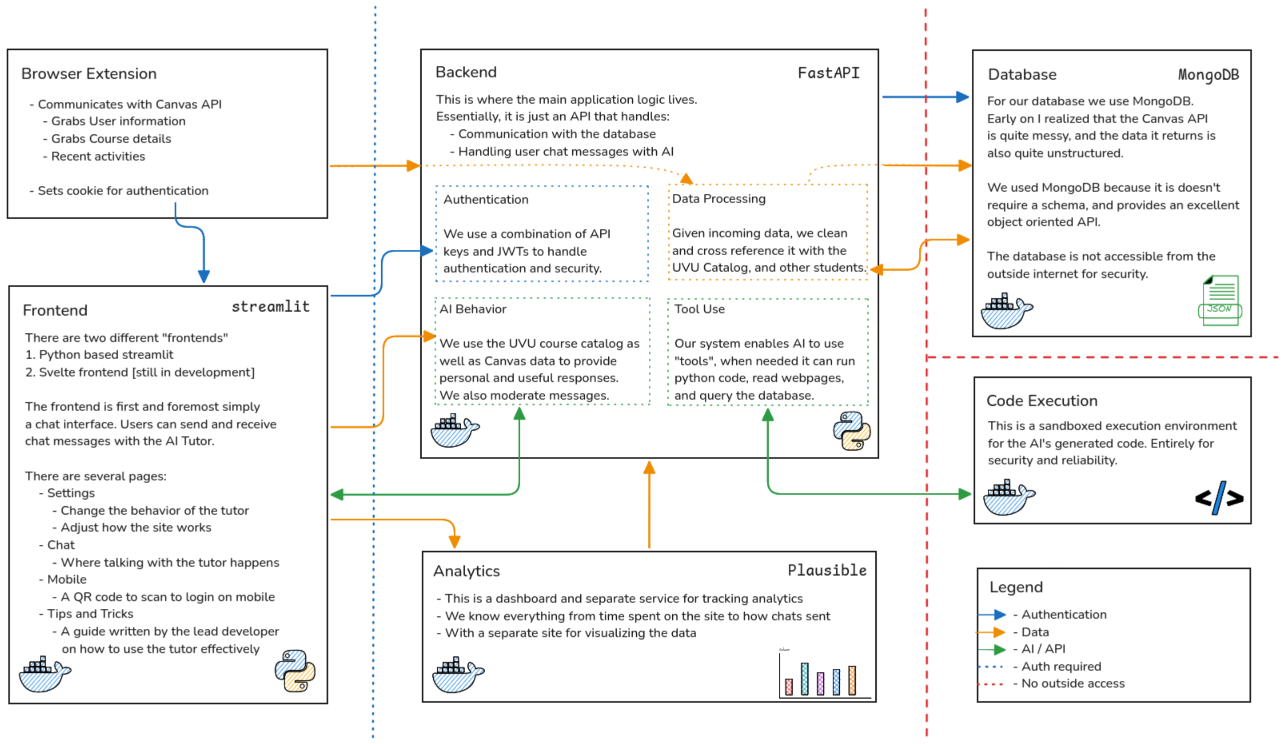


Figure 4: Class Diagram

5.4. Other Diagrams


Browser Extension

- Communicates with Canvas API
  - Grabs User information
  - Grabs Course details
  - Recent activities
- Sets cookie for authentication

Frontend streamlit

There are two different "frontends"
1. Python based streamlit
2. Svelte frontend [still in development]

The frontend is first and foremost simply a chat interface. Users can send and receive chat messages with the AI Tutor.

There are several pages:
- Settings
  - Change the behavior of the tutor
  - Adjust how the site works
- Chat
  - Where talking with the tutor happens
- Mobile
  - A QR code to scan to login on mobile
- Tips and Tricks
  - A guide written by the lead developer on how to use the tutor effectively

Backend FastAPI

This is where the main application logic lives.
Essentially, it is just an API that handles:
- Communication with the database
- Handling user chat messages with AI

Authentication

We use a combination of API keys and JWTs to handle authentication and security.

Data Processing

Given incoming data, we clean and cross reference it with the UVU Catalog, and other students.

AI Behavior

We use the UVU course catalog as well as Canvas data to provide personal and useful responses.
We also moderate messages.

Tool Use

Our system enables AI to use "tools", when needed it can run python code, read webpages, and query the database.

Analytics Plausible

- This is a dashboard and separate service for tracking analytics
- We know everything from time spent on the site to how chats sent
- With a separate site for visualizing the data

Database MongoDB

For our database we use MongoDB.
Early on I realized that the Canvas API is quite messy, and the data it returns is also quite unstructured.

We used MongoDB because it is doesn't require a schema, and provides an excellent object oriented API.

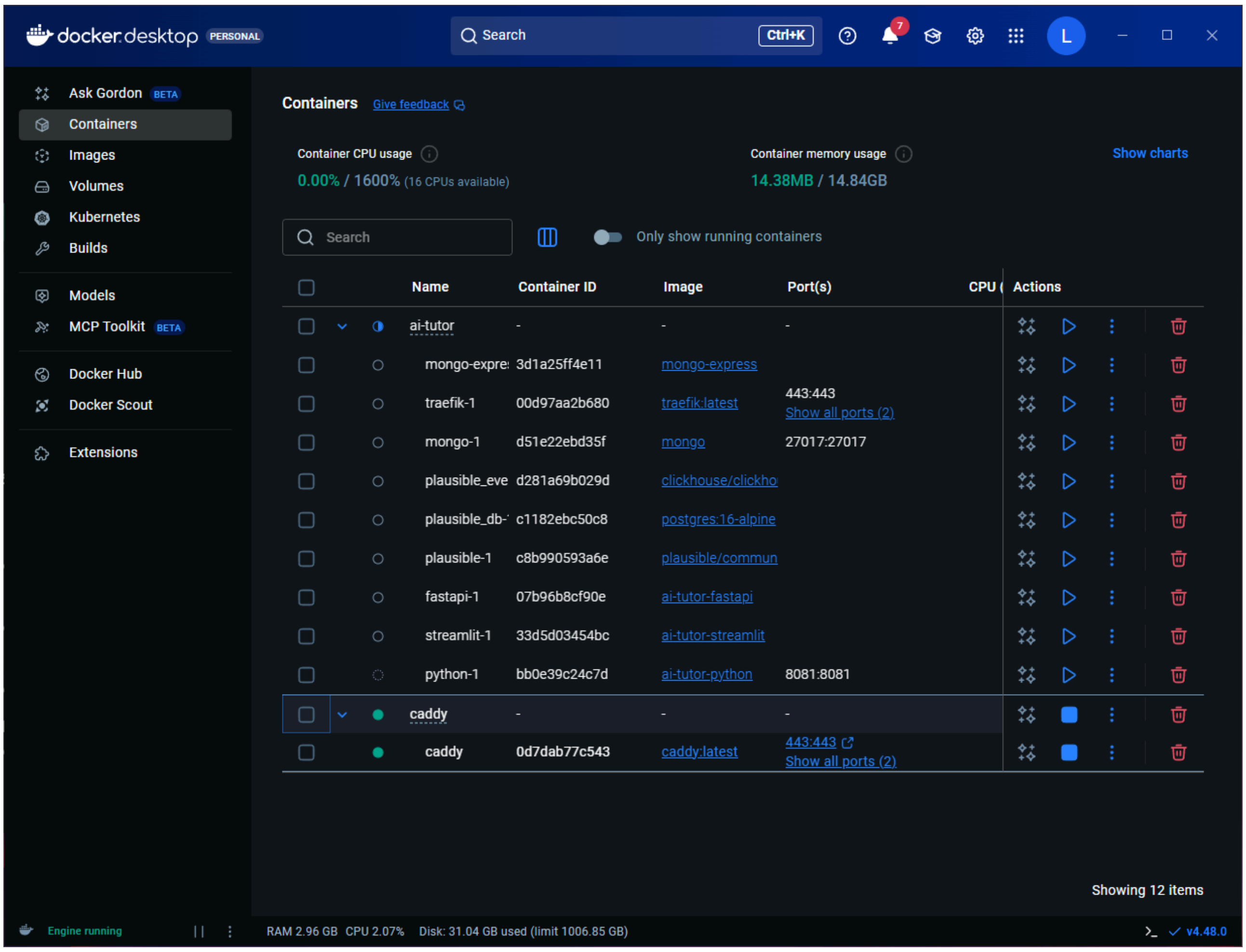
The database is not accessible from the outside internet for security.

Code Execution

This is a sandboxed execution environment for the AI's generated code. Entirely for security and reliability.

Legend
- Authentication
- Data
- AI / API
- Auth required
- No outside access

Figure 5: Architecture Diagram



Containers

Container CPU usage
0.00% / 1600% (16 CPUs available)

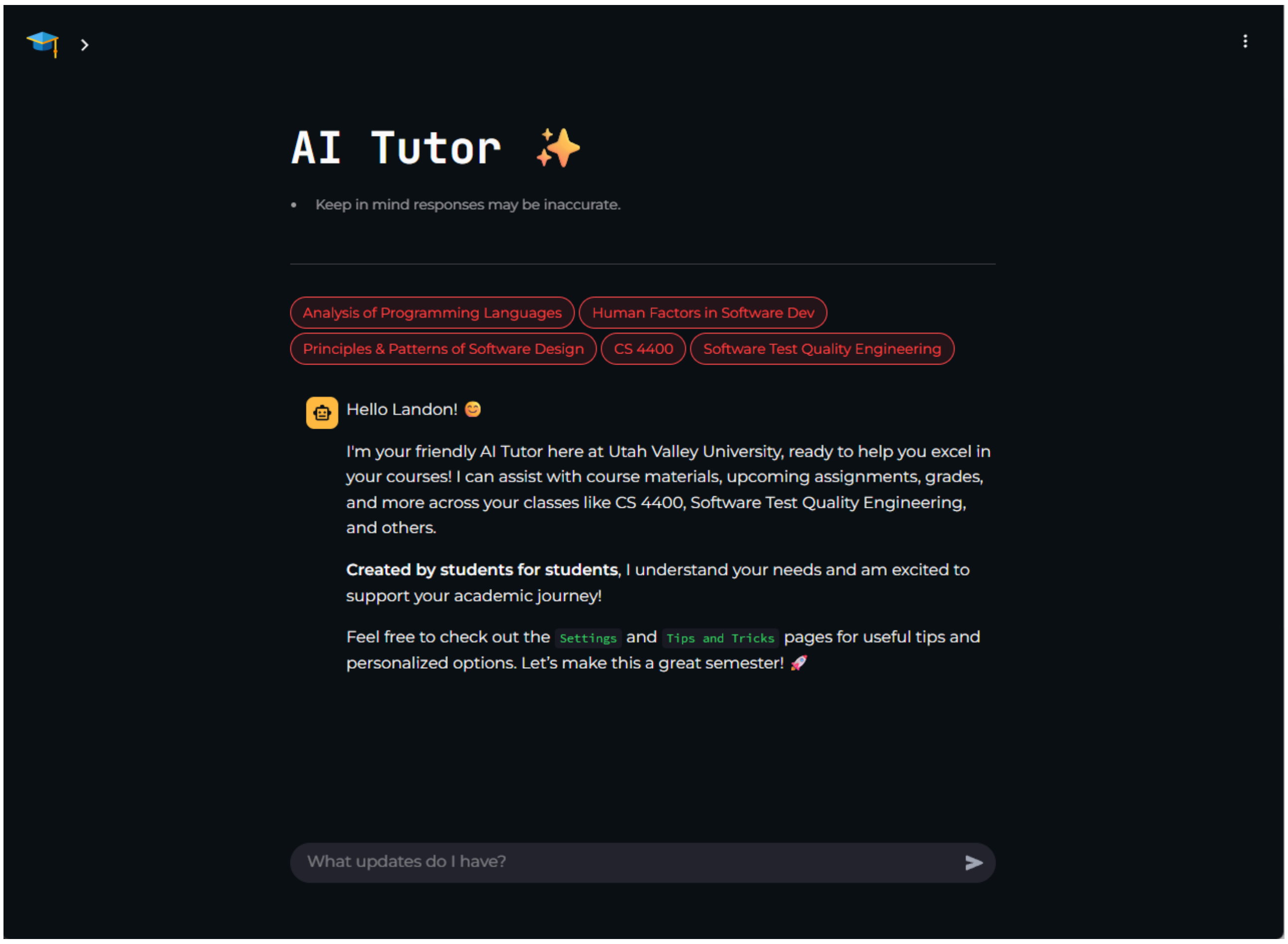
Container memory usage
14.38MB / 14.84GB

Name	Container ID	Image	Port(s)	CPU (
ai-tutor	-	-	-	
mongo-expre:	3d1a25ff4e11	mongo-express		
traefik-1	00d97aa2b680	traefik:latest	443:443 Show all ports (2)	
mongo-1	d51e22ebd35f	mongo	27017:27017	
plausible_eve	d281a69b029d	clickhouse/clickho		
plausible_db-'	c1182ebc50c8	postgres:16-alpine		
plausible-1	c8b990593a6e	plausible/commun		
fastapi-1	07b96b8cf90e	ai-tutor-fastapi		
streamlit-1	33d5d03454bc	ai-tutor-streamlit		
python-1	bb0e39c24c7d	ai-tutor-python	8081:8081	
caddy	-	-	-	
caddy	0d7dab77c543	caddy:latest	443:443 Show all ports (2)	

Showing 12 items

Engine running RAM 2.96 GB CPU 2.07% Disk: 31.04 GB used (limit 1006.85 GB) v4.48.0

Figure 6: AI Tutor Running in Docker



AI Tutor ✨

- Keep in mind responses may be inaccurate.

Analysis of Programming Languages
Human Factors in Software Dev
Principles & Patterns of Software Design
CS 4400
Software Test Quality Engineering

Hello Landon! 😊

I'm your friendly AI Tutor here at Utah Valley University, ready to help you excel in your courses! I can assist with course materials, upcoming assignments, grades, and more across your classes like CS 4400, Software Test Quality Engineering, and others.

Created by students for students, I understand your needs and am excited to support your academic journey!

Feel free to check out the `Settings` and `Tips and Tricks` pages for useful tips and personalized options. Let's make this a great semester! 🚀

Figure 7: AI Tutor Main Page

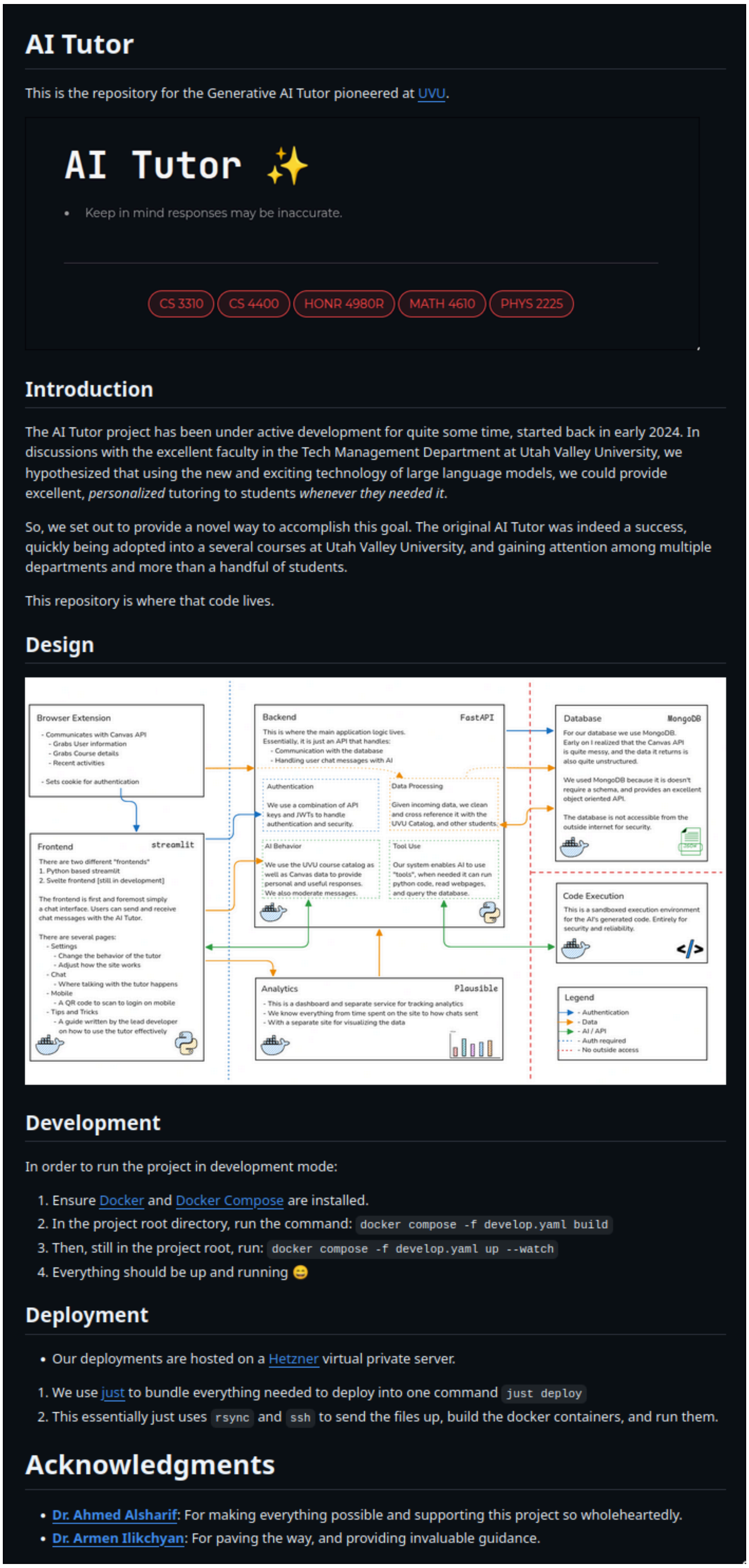
6. Testing

6.1. Testing Plan

6.2. Test Cases

6.3. Quality Assurance Metrics

7. README.md



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

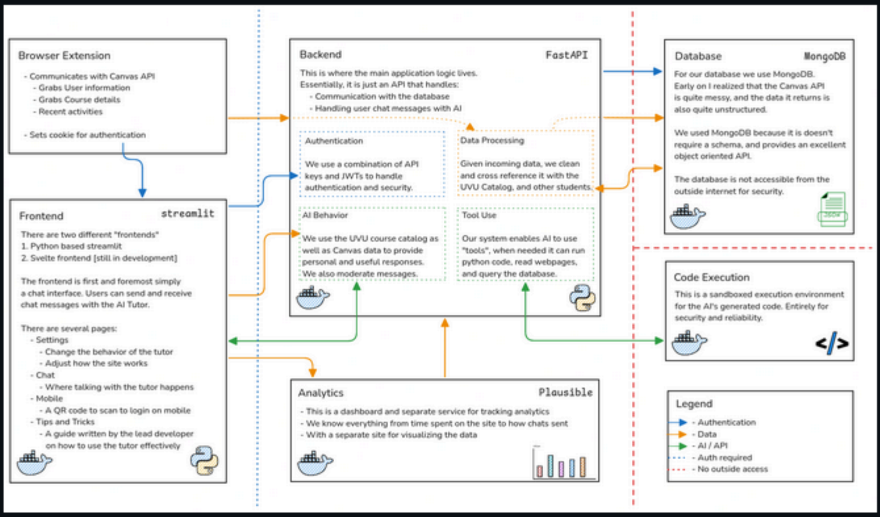
Introduction

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



- Communicates with Canvas API
  - Grabs User information
  - Grabs Course details
  - Recent activities
- Sets cookie for authentication

There are two different "frontends"
1. Python based streamlit
2. Svelte frontend [still in development]

The frontend is first and foremost simply a chat interface. Users can send and receive chat messages with the AI Tutor.

There are several pages:
- Settings
  - Change the behavior of the tutor
  - Adjust how the site works
- Chat
  - Where talking with the tutor happens
- Mobile
  - A QR code to scan to login on mobile
- Tips and Tricks
  - A guide written by the lead developer on how to use the tutor effectively

This is where the main application logic lives.
Essentially, it is just an API that handles:
- Communication with the database
- Handling user chat messages with AI

We use a combination of API keys and JWTs to handle authentication and security.

Given incoming data, we clean and cross reference it with the UVU Catalog, and other students.

We use the UVU course catalog as well as Canvas data to provide personal and useful responses. We also moderate messages.

Our system enables AI to use "tools", when needed it can run python code, read webpages, and query the database.

- This is a dashboard and separate service for tracking analytics
- We know everything from time spent on the site to how chats sent
- With a separate site for visualizing the data

For our database we use MongoDB. Early on I realized that the Canvas API is quite messy, and the data it returns is also quite unstructured.

We used MongoDB because it is doesn't require a schema, and provides an excellent object oriented API.

The database is not accessible from the outside internet for security.

This is a sandboxed execution environment for the AI's generated code. Entirely for security and reliability.

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