AI Tutor

Technical Report

Spencer Thompson

Table of Contents

1. Introduction

This project, the AI Tutor, is a project that has now been under development for quite some time. We have seen some exciting success regarding utilizing artificial intelligence in real world application.

This report outlines Why, How, and What this project is all about, with a focus on the technical design, implementation and solution.

1.1. Background

As previously mentioned, the AI Tutor project started back in early 2024. In discussions with the excellent faculty in 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 24/7.

So we set out to develop a simple application to accomplish this goal. The original AI Tutor was indeed a success. Quickly the project gained attention among multiple departments and more than a handful of students. Although, given the rather breakneck pace of development, there was a rather rapid accumulation of technical debt. Some of the core features that we wanted had become quite difficult.

After the original scope of the project had been completed, we still had a desire to have a system that was better suited for both students and professors. As the lead developer, I had the feeling that a fresh start might be better than attempting to pay down a significant amount of technical debt. Therefore, this report is particularly focused on:

- The evolution of the project as a whole.
- The features that gives the AI Tutor an advantage over **every other alternative**.
- The design and implementation of the tutor, as well as the challenges faced along the way.

1.1.1. Context

A rather interesting piece of context to keep in mind while reading this report is that: during the duration of this project, every single piece of technology has changed **drastically**. Oftentimes the APIs or services that we were utilizing evolved or changed overnight. This could possibly be attributed to the incredible amount of development and hype around the use of generative AI.

The point being, many other competitors both at our own university and others, were quickly building and iterating on similar ideas. Our team built and deployed **two** complete iterations, while other teams have yet to deploy their projects.

1.1.2. Purpose

From the beginning our project was focused on providing a rather niche ideal. We all had this idea of a tutor or assistant that had intimate knowledge of a student's courses.

1.2. Goals

Primarily we wanted:

- 24/7 access to students to assist with coursework.
- Personalized responses to student questions that were unique to *each* student.

- Answer questions about the syllabus, upcoming assignments, grades and more.

The hope and idea being that, this could be an incredibly valuable resource for students, faculty and the university as a whole.

1.3. Limitations

During the duration of this project, there were really two main issues.

1. Incredibly Busy Semester.
2. Canvas API Key Permissions.

The first problem is simply that the developer team and I had incredibly difficult and loaded semesters. In addition, we had several other job opportunities open up. Given that graduation is just over the horizon, it is absolutely true that this somewhat hindered our development.

Regardless though, this first problem pales in comparison to the other. Our university had a competing team that was also working on an AI assistant. We diligently attempted to work with this other team, over the course of a year, but they were obstinate, they insisted that they wanted to hire outside talent.

Part of this problem was that they were unwilling to give us an API Key. This could also have been university permissions issue.

Not having a proper API Key was a truly difficult challenge. Especially given that our team had already deployed to active courses, while they had not.

---

2. Design

The overall design of the project can be a bit confusing at first. Really there are **five** major distinct pieces.

- **Browser Extension**
  - For user authentication and communication with the Canvas API.
- **Frontend**
  - The chat interface for using the AI Tutor.
- **Backend**
  - Where all the data processing and external API calls happen.
- **Analytics**
  - The service that monitors and displays the user Telemetry data.
- **Database**
  - Where we store everything we need.

2.1. Data

The order in which each piece will be explained is the rough order that they function. As a whole, the system is essentially two data pipelines with a unified user interface. Essentially these two pipelines are:

- Analytics
  - Telemetry and usage data gathered from students.
- Canvas
  - Student data regarding assignments, courses, submissions, etc.

2.1.1. Pipeline

While the analytics data is interesting and deserves its own time in the spotlight, it is not the focus of the project.

The Canvas data is really where things get interesting.

- I knew that we needed to grab data from the Canvas LMS, which is Open Source. After thoroughly reading the documentation and looking into the source code, I knew I was on the right track.
- We also needed to process that data and cross reference it with our own users.
- Lastly, we needed a way to securely store and retrieve our mix of user data and course data.

$$\text{Canvas} \rightarrow \text{Our Code} \rightarrow \text{Storage}$$

2.1.2. Analytics

From the beginning, we also knew that we wanted to get some data to attempt to answer the question:

Does AI Tutoring help students?

In addition to the data pipeline, we also needed a smaller pipeline to get user telemetry data to dive into answering this question.

2.2. AI Behavior

From the first iteration of the AI Tutor, we knew that the behavior of the AI Tutor itself was particularly important. It was difficult just to get the tutor not to send super long messages, let alone get it to answer questions with a high accuracy. We needed something extra to get the results that we wanted and needed.

---

3. Implementation

In the first iteration of the AI Tutor was relatively stateless. It did not have a backend, and it did not have a database.

This was quite limiting in term of functionality, and therefore, this time around we had a complete redesign of the entire system. Initially we just had a frontend with *too much* functionality, and we hosted that on Google Cloud.

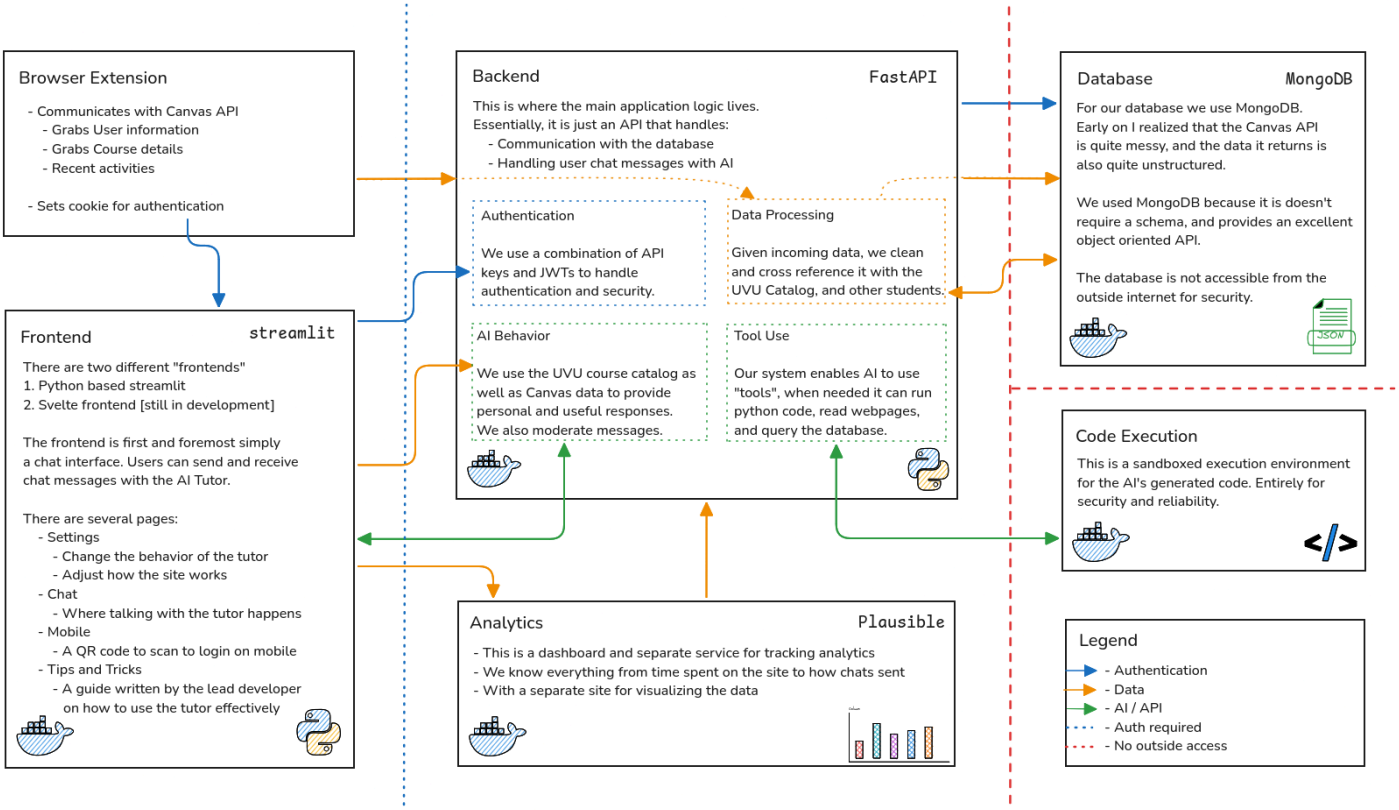

Figure 1: AI Tutor Infrastructure Diagram

3.1. Infrastructure

This time around we wanted to have something more robust with the ability to add one or two entire dimensions of features. Given that we initially hosted on Google Cloud, and didn't really use any of the features of the Google Cloud except for the compute engine, we decided to use something more cost effective and simple.

We settled on using a virtual private server from Hetzner. This decision probably saved us $40 to $50 dollars a month, I would estimate that we saved an order of magnitude of money by switching to Hetzner.

Something that we didn't change from the first AI Tutor was that we used Docker and Docker compose. Docker has made everything much easier to deploy as well as iterate quickly. Using **Traefik** also made this process of development and deployment faster and less painful.

3.2. Security

This bit of implementation was perhaps one of the most clever and interesting pieces of the project as a whole. Coming back to the limitations that I mentioned previously, we did not have access to a developer API key. We had expended a tremendous amount of effort to do this, but with no success. There was a point where hope was starting to be lost, although, perhaps it was pure stubbornness or competition with the official UVU team, I **had** to find a solution.

I had already created a personal API key, and I toyed with the idea of having each user create an API key, but knew that it would be too much an impediment to the user experience.

So, I decided to get creative. In the process of trying to make something incredibly secure, yet give us access to the data we needed, I realized that we could use the *JavaScript Console* in the browser. This shouldn't be a surprise to anyone in web development, but I was curious if I could make an API request from the console.

Assuming it wouldn't work I gave it a try. To my surprise I got back exactly what I had requested from the API.

At this moment, I was surprised that there was no protection against this kind of request, but I knew that most likely there were CORS (Cross-Origin Resource Sharing) limits / restrictions. These are usually included by default on many web applications so that you have to explicitly allow a URL or service to access information.

So I got to thinking, is this something that we could make work? I realized that a browser extension, can indeed run arbitrary JavaScript to interact with a website as long as the user gives permission. **As long as** the CORS policy allowed browser extensions, which I doubted heavily for security reasons, than this could totally work.

I whipped up a quick little browser extension and tested it out. Sure enough it worked like a charm.

- This was a **truly incredible moment.**

Given this discovery, I decided that we would use a browser extension to provide pseudo-access to the Canvas API. For authentication, we just had the browser extension set a cookie on the site as a JSON web token. This simplified and almost removed security as a concern because we could just use the current users existing login as our login.

Further still, our database, and sensitive data is not accessible from the outside web, so in order to get data, a request would need a valid JSON web token as well as an API key that I created.

I can confidently say that our site is incredibly secure.

3.3. Frontend

Something important to our team is that we wanted to have a really solid user experience. Given the constraint that we had a small team, with experience focused on data, this was an interesting challenge.

The first iteration of the project used a library for creating simple websites called Streamlit. This library made it incredibly easy to create and deploy very simple websites.

Another option that we considered is to use a different library, Svelte, which was more in depth.

3.3.1. Chat Interface

In the end we settled on both, initially though we focused on Streamlit, and created a beta written in Svelte.

At first Streamlit was excellent, there was already a lot of easy ways to build AI chat applications, although after two or three months, it became apparent that Streamlit didn't have everything that we needed. We managed to implement some rather hacky and interesting solution to this problem with Streamlit, but still we wanted a better solution.

We then decided to dive into the Svelte option. We didn't get quite to a position that we were satisfied with, but it was very cool nonetheless. Svelte gave us the freedom we really wanted, but also required a lot more effort to design and implement everything we needed.

3.3.2. Browser Extension

The browser extension that I mentioned earlier is the key step of not only authentication but also the data pipeline for Canvas / course data. For Google Chrome and Chromium based web browsers we also configured the extension to provide a side bar on **any site** to access the tutor. This feature turned out to be truly a favorite among users which was cool, even though it was a small addition. We simply just embedded the streamlit frontend as an Iframe within the side panel.

3.4. Backend

I knew that at its heart, this project needed a brain, or central area to take care of the heavy lifting.

After some research FastAPI looked like an excellent option. I just wanted something that we could use to offload the functionality, and create a barrier of sorts for security and clean design. FastAPI worked excellent for this.

Essentially, FastAPI just lets you as a developer write functions with decorators, and the function becomes the API (http) endpoint. There are great support, examples, and documentation.

3.4.1. AI

This is an **AI** Tutor after all, and we had some chaos this time around. Primarily, I am talking about providers. Originally we had a grant from OpenAI that we were using for credits to provide chat completions.

Although, one day recently after deployment, our credits were not working. Within a couple hours, I had completely switched us over to using the Anthropic API. Even though we didn't stay with Anthropic, I do think it is interesting to note that their AI behavior seemed to be more emotionally intelligent than OpenAI.

3.4.2. Tools

The real magic of the AI though, is the tool use / function calling. This functionality is what really made this version of the AI special and unique. This features essentially allows the AI to choose a *tool*, which is essentially just a defined JSON schema. Then when generating a response, the tool call itself can be extracted and used in code.

The tools that we gave to our AI Tutor are:

- Grades
  - Show grades of assignments and courses overall.
- Course Questions
  - Access to information about upcoming assignments, recently submitted assignments, discussions, announcements, and more.
- Catalog
  - Access to the UVU course catalog with information about prerequisites, credit hours and more.
- Read Webpages
  - If the user pasted a URL or URLs, the AI could read those pages for the user.
- Execute Python Code
  - If calculation or precision was needed the AI could execute python code in a sandboxed environment.

These tools really made the AI feel and be truly useful to users.

3.4.3. Telemetry

As I have mentioned previously, we needed to collect some telemetry in order to answer some research questions. As I had experience with Plausible Analytics, I setup the self hosted version and collected data about how our users use the site. One of the reasons I like Plausible is that it is privacy conscious.

Figure 2: Plausible Analytics Dashboard

3.5. Database

The last portion of the implementation is the database. After seeing the data that was coming back from the Canvas API, I just felt uncomfortable dealing with the inconsistencies of that data. SQL didn't necessarily feel quite right.

So, we decided to go with MongoDB, a document based database. The real reasons why this decision was made are:

- We had a lot of JSON data, which fit well into Mongo.
- Saving chats would be incredibly easy.
- It would allow very quick prototyping and iteration.

This turned out tremendously well.

---

4. Challenges

This project has been incredibly interesting and satisfying, partly due to the unique problems and creative solutions these problems required.

4.1. Permissions

The main pain point of the development of the Tutor was the resistance we got from the official university IT department. They did not want to provide us an API key, as mentioned previously, due to security and competition.

This lead to quite a bit of hopelessness in the beginning of the project. While we did find unique solutions to these issues it was consistently difficult nonetheless.

4.2. API Inconsistency

Another interesting problem that we encountered was the different APIs that we encountered along the way. The two that I had the most trouble with were:

- **Canvas**
- Plausible Analytics

Regarding **Canvas**, this one was tricky and frustrating because there was just so much inconsistency. Reading the documentation hinted that, over time this software had so many features added that there was no clear or well defined structure. This may seem like a naive statement, and perhaps to some extent that is true. At a basic level, the very simplified schema of the Canvas Instructure database probably looked something along the lines of:

- **Courses** have **Users** and **Users** have **Courses**
  - (many to many)
- **Courses** have:
  - **Assignments**, **Discussions**, **Submissions**, **Announcements**, etc.
  - (one to many)

Now, on its own this would have been fine. Given that **Courses** and **Users** is many to many it does get complex. But this would be manageable, except that there are so many more little tables in their database. Further still, many of these tables don't have restrictions like the classic (and lifesaving) SQL:

NOT NULL

So often, there were fields returned by the API call that were `NULL` and with no defined schema that I could find, this caused endless bugs.

One other rather interesting quirk of the Canvas API is that Courses have a `course_name` and `course_code` field. I only much later found out that the `course_name` field could be changed by the user which was frustrating to say the least.

4.2.1. Analytics

I wanted to add one more little interesting bit about working with APIs. As I have mentioned, we used **Plausible Analytics** for our telemetry. I initially *assumed* that their API would be nice and neat, given that that piece of software is open source and written in Elixir, a popular functional programming language.

What I did not expect, was that when making queries to the API to get custom analytics data, really this was just essentially a Cartesian product of everything I was asking for. When dealing with high dimensional data, this is helpful, although the implementation just seemed quite frustrating.

4.3. Messy Data

- Canvas
- My own Design

4.3.1. Live Service

- Developing / Adding features for an application in use is difficult

---

5. Solutions

5.1. Workarounds

- Browser Extension

5.2. Clever Tricks

5.2.1. Authentication

- JWT / Cookie

---

6. Conclusion

6.1. Findings

6.2. Implications