AI Tutor

Technical Report

Spencer Thompson

Table of Contents

1. Introduction 2
   1.1. Background 2
      1.1.1. Context 2
      1.1.2. Purpose 2
   1.2. Goals 3
   1.3. Limitations 3
2. Design 3
   2.1. Data 4
      2.1.1. Pipeline 4
      2.1.2. Analytics 4
   2.2. AI Behavior 4
3. Implementation 4
   3.1. Infrastructure 5
   3.2. Security 5
   3.3. Frontend 6
      3.3.1. Chat Interface 6
      3.3.2. Browser Extension 7
   3.4. Backend 7
      3.4.1. AI 7
      3.4.2. Tools 7
      3.4.3. Telemetry 8
   3.5. Database 8
4. Challenges 9
   4.1. Permissions 9
   4.2. API Inconsistency 9
      4.2.1. Analytics 9
   4.3. Messy Data 10
      4.3.1. Live Service 10
5. Conclusion 10

1. Introduction

The AI Tutor project has been under development for quite some time. We have seen some exciting success regarding the utilization of artificial intelligence regarding real world problems. There is a lot of public and professional focus on AI and this makes it hard to determine what is mere hype and what is real. We can confidently say that the AI Tutor provides both genuine utility and real innovative cost savings.

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

1.1. Background

As previously mentioned, 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. Although, given the rather breakneck pace of development, there was a rapid accumulation of technical debt. Some of the core features that we wanted had become more complex than we wanted.

After the original scope of the project had been completed, we still had a desire to build a system that was better suited for both students and faculty. As the lead developer, I knew that a fresh start might be better than attempting to refactor away 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

An 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 **fundamentally changed**. Oftentimes APIs or services that we relied on evolved or changed drastically overnight. This could be attributed to the incredible amount of development and hype around generative AI.

This is all to say that, many other competitors both at our **own university** and others, were actively building and iterating on similar ideas. Our team built and deployed **two** complete iterations, while other teams have yet to deploy their applications.

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 students' courses, and felt more real to chat with than simply using *another chatbot*.

1.2. Goals

Primarily we wanted to create a system that:

- Gave students 24/7 access to truly great tutoring.
- Provided personalized responses to student questions that were unique to *each* student.
- Answered questions about the syllabus, upcoming assignments, grades, and more.

Our hope and core idea being: this system 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. The developer team, as students, all had surprisingly busy semester course load.
2. The university did not want to provide proper Canvas API key permissions for our team.

The first problem is simply that the developer team and I each had particularly challenging and busy semesters. Additionally, we each multiple other job opportunities open up, which both added to the sheer load of work, and caused us to think more about the future. Given that graduation was just over the horizon, it was difficult at times to have a streamlined development process.

Regardless, this first problem pales in comparison to the other. Our university had a competing team of developers that were also actively working on their own AI assistant. We earnestly attempted to collaborate with this other team over the course of a year, but they insisted in hiring outside talent.

This problem was exacerbated by their unwillingness to provide our team with a proper Canvas API Key. This could also have been university policy regarding permissions.

Not having a proper API Key was truly a difficult challenge. This was especially frustrating given that our team had **already** deployed the first iteration of the AI Tutor to active university courses, while they had not.

---

2. Design

The overall design of the project can be a bit confusing to understand at first. There are **five** major distinct modules that provide the core functionality:

- **Browser Extension**
  - For user authentication and communication with the Canvas API.
- **User Interface**
  - The frontend 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 user telemetry data.
- **Database**

    - Where we store everything.

2.1. Data

The order in which each piece will be explained is the rough order that they function together. 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.
- **Course data from Canvas**
    - Student data regarding assignments, courses, submissions, etc.

2.1.1. Pipeline

While the analytics data is interesting and deserves its own explanation, it is not the focus of this project.

The course data from Canvas is where the unique functionality of the Tutor comes from.

- We knew that we needed to grab data from the Canvas LMS, which is Open Source on Github. After thoroughly reading the documentation and looking into the source code, we knew we were 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 collect 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. This would allow us to dive into answering the question above.

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 to make the Tutor send concise messages, it would often keep generating output and essentially *ramble*. Further, responses tended to stray from the question or even worse, make up or *hallucinate* responses that were not correct. We knew we needed something more to get the kind of results that we wanted and needed.

---

3. Implementation

In the first iteration of the AI Tutor, our code was relatively stateless. It did not have a true backend, and it did not have a database to store user data.

This was quite limiting in term of functionality. Therefore, we decided on a complete redesign of the entire system. Initially, we only had a frontend with perhaps *too much* functionality that was hosted on Google Cloud.

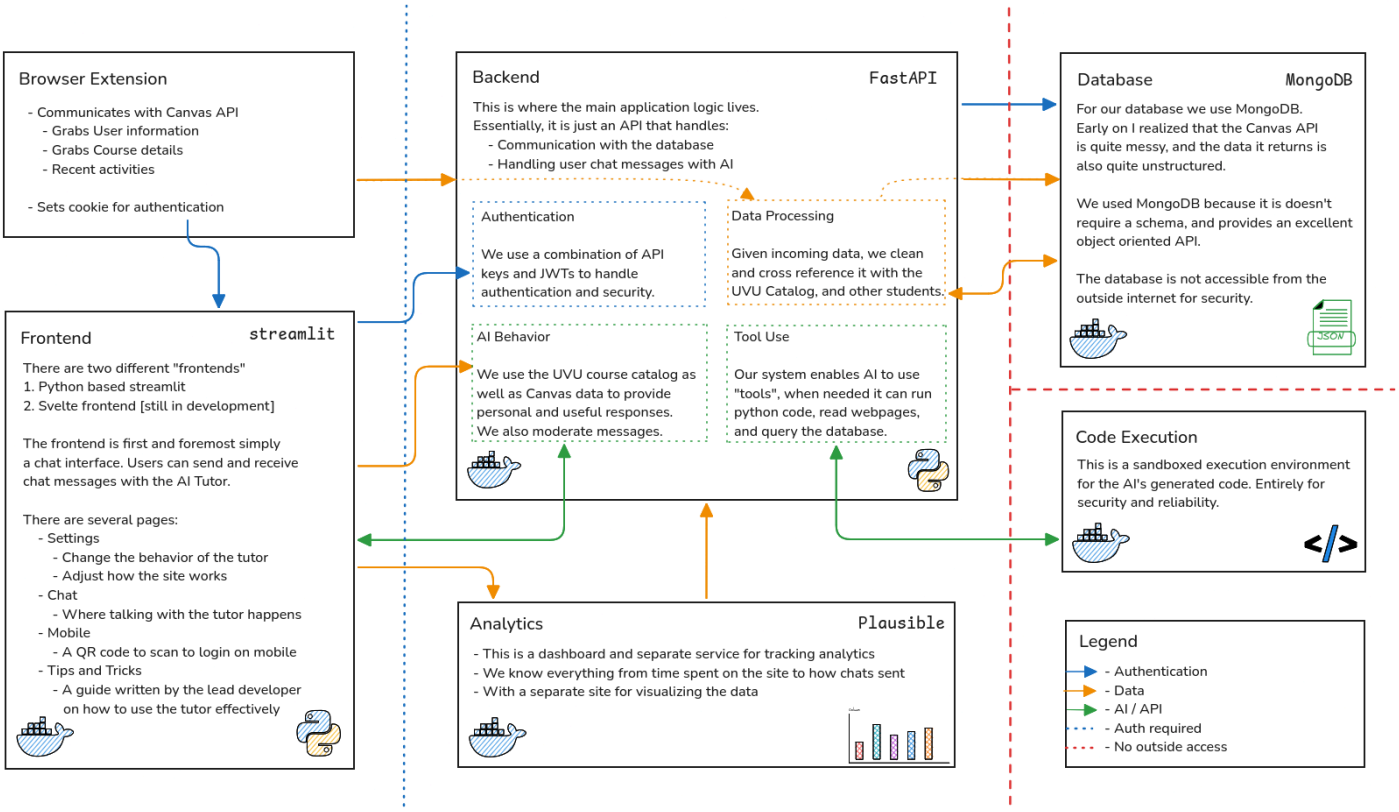

Figure 1: AI Tutor Infrastructure Diagram

3.1. Infrastructure

We wanted to create something more robust with the ability to add one or two entire dimensions of features. Given that we initially hosted on Google Cloud, we realized that we didn't utilize hardly any features of 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 turned out to save us $40 to $50 dollars a month, I would estimate that we saved an order of magnitude by switching to Hetzner.

Something that we didn't change from the first AI Tutor was using Docker and Docker compose. Docker made everything much easier to develop quickly and deploy effectively. We also used **Traefik**, which made this process of development and deployment much faster and less painful.

3.2. Security

Our implementation of security was perhaps one of the most clever and unique pieces of this project as a whole. Given our limitations mentioned previously, we did not have access to a real developer API key. We truly expended a tremendous amount of effort to obtain this API key, but with no success. At one point, we had started to lose hope. Although, perhaps due to pure stubbornness or competition with the official university team, we **had** to find a solution.

We had already created personal API keys, and had toyed with the idea of having each user create an API key, but we all knew this would be an impediment to the user experience.

So, we decided to get creative. In the process of trying to make something incredibly secure and provide access to the data we needed, we realized that we could use the *JavaScript Console* in the browser to run arbitrary code. This shouldn't be a surprise to anyone in web development, but we were curious if we could make an API request to the Canvas API from the console. Assuming it wouldn't work we gave it a try. To our surprise we got back exactly what we had requested from the API.

At this moment, we were surprised that there was no protections against this kind of request, but knew that most likely there were **CORS** (Cross-Origin Resource Sharing) policy restrictions. These are usually included **by default** on many web frameworks so that URLs must be explicitly allowed to access information.

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

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

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

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

Further still, our database, and sensitive data would not be accessible from the outside web. This by extension enforced a request to have a valid JWT (JSON web token) as well as an API key that we created, to get any data.

I can confidently say that our site is incredibly secure.

3.3. Frontend

A fundamentally important aspect of the Tutor was 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 was to use a different framework, Svelte, which provided more features and control.

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 an excellent choice. It already provided many simple methods to build a chat applications. After two or three months though, it became apparent that Streamlit didn't quite have everything that we needed. We managed to implement some rather *hacky* and interesting solutions to our problems with Streamlit, but still, we wanted a better solution.

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

3.3.2. Browser Extension

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

3.4. Backend

We 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. We wanted something that we could use to offload functionality, as well as, create a barrier of sorts for security and provided us with cleaner design. FastAPI was excellent for this.

Essentially, FastAPI just provides developers a method to write simple functions with decorators, and the function becomes the HTTP endpoints. There are great support, examples, and documentation, which allowed us to iterate quickly.

3.4.1. AI

This is an **AI** Tutor after all, and we had some bumps along the road this time around. Primarily, referring to AI 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, we had completely switched over to using the Anthropic API. Even though we didn't stay with Anthropic, 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 truly made this version of the AI Tutor special and unique. This feature 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.
- **Course 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 mentioned previously, we needed to collect telemetry data in order to answer a variety of research questions. Having experience with Plausible Analytics, we setup the self hosted version and collected data about how our users use the site. One of the reasons we liked 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, we 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 decision turned out tremendously well for the project.

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, and while we did find unique solutions with the browser extension to these issues, it was consistently difficult nonetheless.

4.2. API Inconsistency

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

- **Canvas**
- **Plausible Analytics**

Regarding **Canvas**, this was tricky and frustrating because there was just so much inconsistency. Reading the documentation hinted that, over time Canvas 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 a many to many relationship it does get complex. 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 that were `NULL` and with no defined schema that I could find, this caused endless bugs.

One rather interesting quirk of the Canvas API is that Courses have a `course_name` and `course_code` field. Only much later did we discover that the `course_name` field could be changed by the user which was frustrating to say the least.

We had used `course_name` for a significant portion of our code almost as a pseudo primary key.

4.2.1. Analytics

I wanted to add one more little interesting bit regarding these APIs. As mentioned above, we used **Plausible Analytics** for our telemetry. We initially *assumed* that

their API would be nice and neat, given that this piece of software is open source and written in Elixir, a popular functional programming language.

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

4.3. Messy Data

As the project continued, we started to be grateful for design decisions that we had made earlier in development and became painful aware of early mistakes as well.

All things considered, our team did an excellent job of design. Most issues we faced mostly revolved around external APIs that we were using.

Probably the biggest issue that we ran into due to our own design was the database. It was both an excellent choice but it also presented some issues. The main issue in question is that **Joins are complicated in MongoDB**.

Whenever we wanted to have a document in mongo be **"joined"** on another document, it was messy. Regardless, we did manage to make it work, and an insight for the future is that we will probably use both a document and relational database in tandem, in the future.

4.3.1. Live Service

As a quick comment, this project taught us something incredibly valuable:

- Developing an application with active users is **difficult**.

As a team, we have worked on projects that are active, or perhaps have users to some extent, but this was something new altogether. If we created bugs, we immediately heard from professors and students, and having issues in the production code was a much more serious problem than we could have imagined.

We really had to be careful and deliberate when working on features or refactoring. Additionally, we learned that when adding fields to a database or feature; new and old users' data would cause issues. What would often occur is that when adding new features, we would have to be clever or have database migrations to avoid `null` values, which would wreak havoc in the code.

---

5. Conclusion

All in all, this project has been an incredible learning experience and a huge success. We managed to get 300+ users, and provide a valuable and cutting edge tool for our university.