

Spring 2025

CMPT 782 – Special Topics in Computer Science (Web Development)

Project – Phase 2

Fact-Checking for AI Chatbots: A Semi-Automated Human-in-the-Loop Approach

Submitted by:

Meshaal Al-Saffar (200607511)

Submitted to:

Dr. Abdelkarim Erradi

Date: 18 May 2025

1.1 Overview & Requirements

This project represents Phase 2 of a previous initiative, building upon the foundational work to enhance AI trustworthiness and information accuracy. In this phase, two critical aspects are additionally implemented: the development of a data layer using **Prisma** to facilitate structured storage and retrieval of information, and the incorporation of security mechanisms using **JWT** to safeguard user data and ensure the integrity of the verification process. These additions aim to further strengthen the reliability and functionality of the semi-automated human-in-the-loop fact-checking tool, making it more adaptable to organizational needs and secure for end-users.

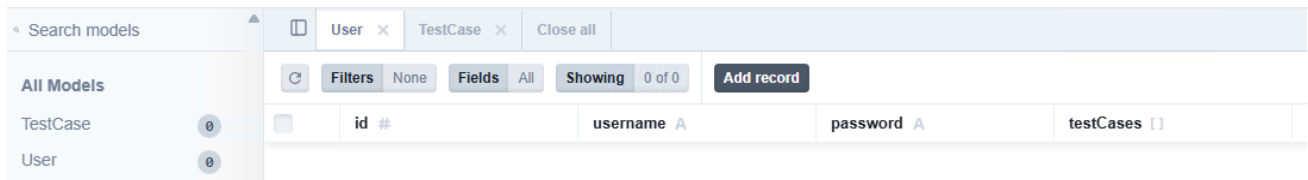
1.2 Limitations & Future Work

It is important to note that the focus of this phase was predominantly on the web development aspect of the system. This excluded the integration of actual AI chatbot responses, which were considered beyond the scope of the current work. This approach aimed to lay a strong foundation for the system's infrastructure, enabling the potential integration of AI-driven functionalities in subsequent phases. This integration remains a promising opportunity for future development, as it could lead to a standalone system with potential for commercialization.

1.3 Code Upgrades Summary

- Installed Prisma CLI as a dev dependency:
`npm install prisma --save-dev`
- Initialized Prisma with SQLite as the datasource:
`npx prisma init --datasource-provider sqlite`
- Checked if Prisma Studio runs:
`npx prisma studio`
- Installed Prisma Client:
`npm install @prisma/client`
- Defined the Prisma data model:
User and TestCase, with a one-to-many relationship (one user can have many test cases, each test case belongs to one user). Refer to file `/prisma/schema.prisma`

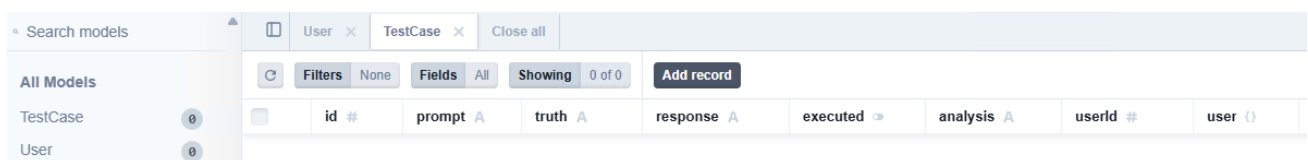
- Ran the first migration:
`npx prisma migrate dev --name init`
- Checked the new data model in Prisma Studio:
`npx prisma studio`
Refer to the figures below



The screenshot shows the Prisma Studio interface with the 'User' model selected. The table has columns: id #, username A, password A, and testCases [].

id #	username A	password A	testCases []
------	------------	------------	--------------

Figure: User data model

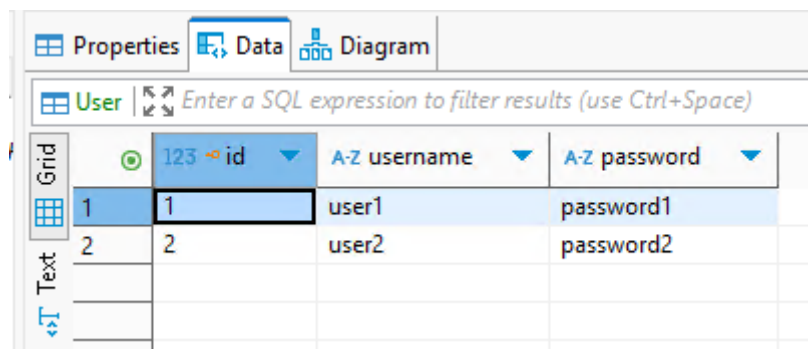


The screenshot shows the Prisma Studio interface with the 'TestCase' model selected. The table has columns: id #, prompt A, truth A, response A, executed, analysis A, userId #, and user {}.

id #	prompt A	truth A	response A	executed	analysis A	userId #	user {}
------	----------	---------	------------	----------	------------	----------	---------

Figure: TestCase data model

- Seeding the database with users.
Refer to file `prisma/seed.js`
Ran the command `node prisma/seed.js`
Refer for figure below to see the data



The screenshot shows the Prisma Studio interface with the 'User' model selected. The table has columns: id, username, and password. The data is as follows:

id	username	password
1	user1	password1
2	user2	password2

Figure: Authorized users

- Updated API routes to use Prisma Client instead of reading/writing JSON files.
- Updated the Frontend to use the API
- Many other codes updates
- Upgraded authentication to JWT
- Rewired the previous functionality to the authenticated user in JWT

1.4 Server-Side Actions / API

Table: API Endpoints

Method	URL	Description
GET	/api/testCases	Retrieves all test cases or a specific test case by ID from the data store.
POST	/api/testCases	Adds a new test case to the data store.
PATCH	/api/testCases	Updates one or more test cases in the data store. Supports "edit" or "simulate" operations.
DELETE	/api/testCases	Deletes a specific test case by its ID from the data store.
POST	/api/auth	Authenticates a user by verifying their username and password.

Important note about the PATCH method of the /api/testCases handler, an attribute `operationType` with possible 2 values ("edit" or "simulate") have been used to distinguish the operations:

1. Updating a Single Test Case (Edit Operation):
2. Updating Multiple Test Cases (Batch Execution):