CSCI 370 Final Report

CSM Fierro

Alejandro Estrada Silva
Angel Lechuga Gonzalez
Donovan Keohane
Tiana Nguyen
Michelle Torres

December 10, 2023

CSCI 370 Fall 2023

Mr. Tree Lindemann-Michael

Table 1: Revision history

Revision	Date	Comments
New	August 31,2023	Created document and added Introduction, Requirements , Risk, and Definition of Done.
Rev – 2	September 17,2023	Added system architecture and design.
Rev – 3	October 22,2023	Added Quality testing and ethics
Rev – 4	November 12, 2023	Added Results and future work
Rev – 5	December 1, 2023	Revised suggestions
Rev – 6	December 10, 2023	Finalized Report

Table of Contents

I. Introduction

Modern buildings utilize various sensors and technological systems to improve energy efficiency, accessibility, and other aspects of life. It is common for buildings to employ Building Management Systems (BMS) that are specific to vendors and even buildings in some cases. Currently, the environment does not allow for solutions to be translated between buildings. The National Institute of Standards and Technology estimates that this lack of translatability and interoperability results in a loss of about $15.8 billion dollars annually [1]. It is therefore useful to develop a universal and unified schema that all buildings can use.

With an increased need for semantic graphs in modeling large cyber-physical systems, Dr. Gabe Fierro, an assistant professor at the Colorado School of Mines, founded Brick Schema, an ontology and data model that defines a standard representation of data used by smart buildings. In the development of Brick, Dr. Fierro uses large language model (LLM) embeddings in order to create mappings between existing vocabulary and Brick. However, in order to use this tool, users originally had to write their own python script and upload their vocabularies. This method was not user friendly, especially for users with limited programming experience. If there exists a more user-friendly solution, Brick has the potential to be used by more people who need it. Consequently, the CSM Fierro team consisting of Alejandro Estrada Silva, Angel Lechuga Gonzalez, Donovan Keohane, Tiana Nguyen, and Michelle Torres were tasked to create a web interface to Brick. The goal of this project was to allow users to easily upload their own vocabulary file on a website and download a translation file that results from using Brick. The user is given the option to choose input file types, vocab relations, and output file types. In addition to the mapping results that Brick outputs, the team's goals at the beginning of the project were to develop a visual representation of the mapping and to improve cache embeddings in the vector database.

II. Functional Requirements

1. Frontend
    a. The frontend will encompass the user uploading a vocabulary in a specified format, submitting it for processing, filtering results, and exporting the mapped vocabulary in a given format.
    b. The user should be able to select the scopes which describe their data, and what relationship mapping the backend should produce.
2. Frontend hosting
    a. The frontend will be static and should therefore be served by a lightweight platform.
3. Backend
    a. The existing script from Dr. Fierro must be extended to expose a backend by taking in user inputs for mapping types and filters of the Brick classes.
    b. Input sanitization must be performed for all files supplied through the frontend.
    c. A new endpoint should be created to facilitate the user supplying an existing vector database to load into the backend.
4. Vector Database
    a. The existing vector database solution should be analyzed and replaced with a superior option if it exists
        i. If time allows, our client requested that alternatives for the vector database libraries be investigated.
        ii. The database should be able to work with the current Brick Mapper class and ideally have built-in caching.

III. Non-Functional Requirements

1. Wrap current Brick mapper and infrastructure around web app.
2. Inputs must be in .xls, .csv, or .json format.
3. Output must be in .xls, .csv, or .json.
4. Front end will incorporate REACT.
5. The backend will incorporate Python's Flask.
6. Pinecone will be used as the vectorized database.
7. GitHub will be used as the repository for team code development.
8. Final product must be deployable, using containerization and Docker Compose.

IV. Risks

Risks for this project includes technical risks and skill risks:

A. Technology Risks
   a. Users can input different file types and there may be a risk of their document/format being converted incorrectly.
   b. The capability of users choosing different mapping options could malfunction and limit the user's desired output.
   c. Our solution could crash if it is unable to handle large amounts of data and traffic.
   d. Our team could lose cache embeddings.

B. Skill Risks
   a. Much of the project relies on frontend construction, only two of our members have experience with frontend projects. Our team will have to learn JavaScript along with other software to implement frontend code.
   b. If time allows to improve the LLM embeddings, none of our team has experience with embeddings/tokenization. We will need to get familiar with the subject and ask Dr. Fierro for support.

C. Technical Design Issues
   a. Figuring out how to connect the backend to frontend on one server to reduce security risks
      i. Plans to address this issue include using Nginx to direct the client to different apps based off the route and then secure it using Flask-CORS.

V. Definition of Done

1. Minimal useful features
   a. Accessible web application.
   b. Allow users to upload files in different formats, .xls, .csv, and .json.
   c. Allow users to choose between different mapping options, one-to-one, many-to-one, or one-to-many.
   d. Communication between frontend and backend software.
   e. Output translated file in Brick schema
      i. Allow users to download output files.
   f. Cache embeddings to reduce processing cost.
   g. Containerization of the web application software.
2. Client Test
   a. The client will test if the web application is successful in creating and outputting the mappings of different schemas to Brick.

b. The client will test if the web application is accessible, functional, and simple to use.

The product will be delivered using Docker Compose files to ensure all services run and dependencies match up correctly. The client will receive a working web application.

VI. System Architecture

The web application is designed to have a simple interface to allow for any user to use the translating tool. Our web application makes it easy for the user to upload files, select mapping options, see results, and download files. The order mentioned is the typical workflow of the user on the web application. In general, the user is able to upload files, customize definitions, view results, and download files.

We have identified two different types of users, one being people with a computer science background and understanding of the software and the other being people with little to no computer science background. The web application therefore was designed to be easy to use for both parties with guided help for those with no background in computer science.

In order to accomplish this design, our solution is outlined by the system architecture shown below in Figure 1. Our project is running on two docker containers on a Google Cloud Platform instance that is accessible from the internet. When a user hits the instance, they access an Nginx reverse proxy running in our frontend container with two routes. The first route, the root route, directs to our static React/Chakra UI frontend. Upon submission, the frontend sends a request to the /api route which hits our backend container. The backend consists of a Python flask backend secured by cross-origin resource sharing.

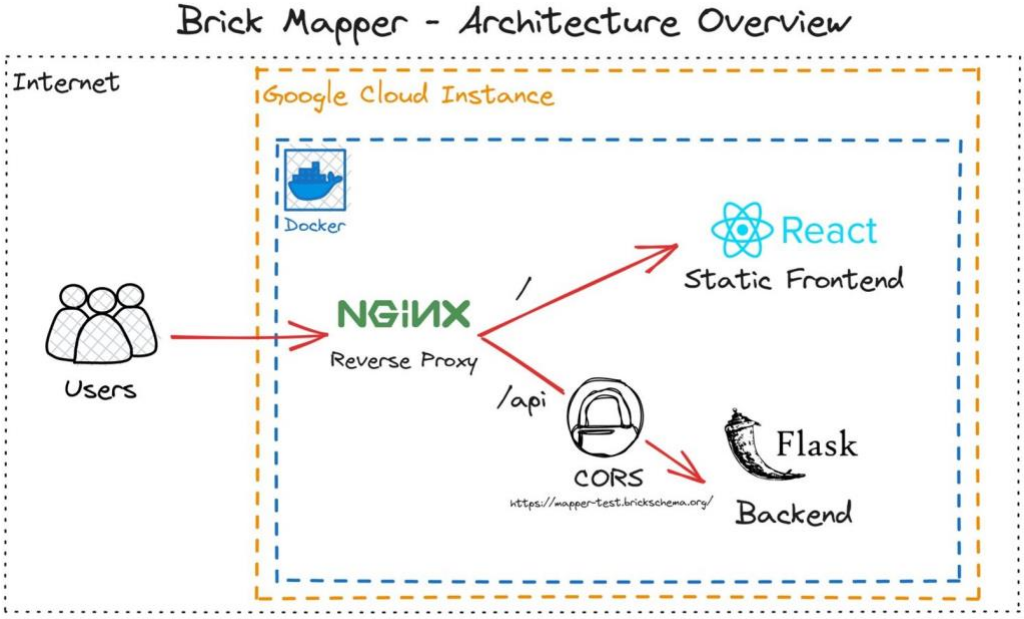

Figure 1: System Architecture Diagram

The system architecture is further highlighted in the initial wireframes of the web app shown below in Figure 2. The wireframes showcase the static React driven frontend. As shown in the wireframes, the frontend should update as the user selects their mapping options and filters. It also highlights the calls to the backend API to produce the mappings

and return to the frontend for the user to view and download. The wireframe also demonstrates how the backend, and the frontend should be running simultaneously in agreement with the system architecture.

Figure 2: Website Wireframe

As shown by both the system architecture and the website wireframe, the system should be able to connect the frontend and the backend API to ensure the user has customization of the mappings as well as to provide the mappings in real time. As mentioned before, the backend and the frontend will be running simultaneously using Docker containerization and will be hosted on a Google Cloud Instance for users to access.

VII. Technical Design

The web application workflow is simple as seen in Figure 3 below, and its functionality can fit into one main web page. Our design directs users to a page where they can submit their own index/schema file that they would like to "translate". Along with the option to upload, users must choose the mapping type and scope they hope for. After choosing their options, their mapping output will be displayed in two different ways. If the user indicates one-to-one mapping, the output will display, and they can decide what file type they would like to download. If the user indicates one-to-many mapping, they will have the option to pick what output best fits what they are looking for and can download in the file type they desire. This functionality occurs within one page to simplify the workflow and ensure users do not get overwhelmed or confused with page layout or workflow.

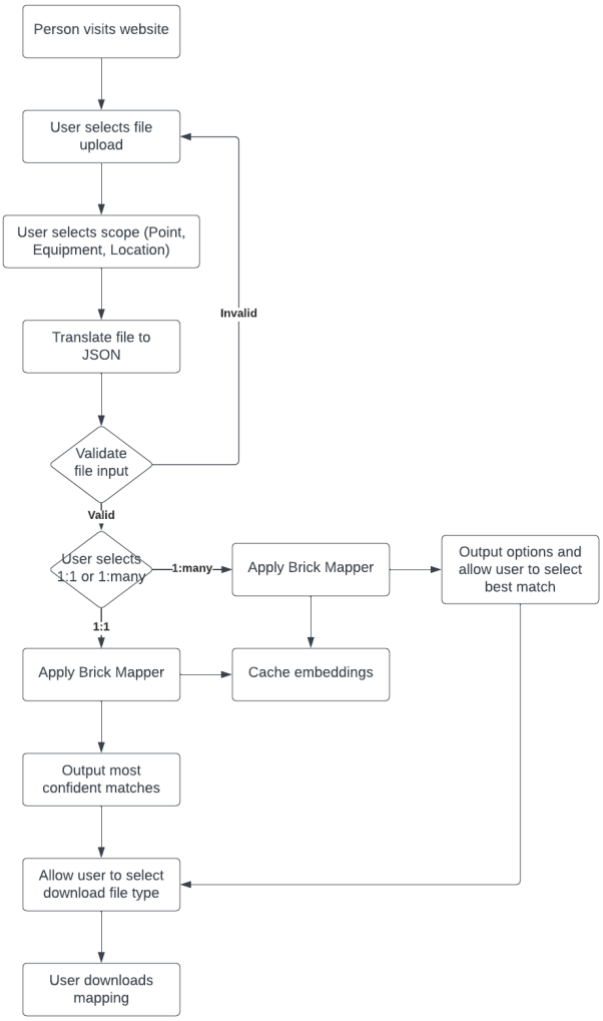

Figure 3: Web Application Workflow

As seen in Figure 3, the frontend handles the user uploading files, selecting mapping type, and filter selection. The frontend will also be used to display the outputs to the user and allow them to download if needed. If the file is invalid the user will be prompted by the frontend to reupload, file validation takes place in the backend. The backend is also responsible for applying the brick mapper, caching embeddings, and populating the results for the frontend display.

With the consideration that the backend largely extends the work done by Dr. Fierro, our design of the backend architecture can be seen below in Figure 4 and incorporates the Brick Schema. The architecture highlights the different inputs that go into the computations of the embeddings for the mapping artifact. The first input is the user provided schema. This is the file that the user submits on the web app to be translated into the Brick Schema. Another input is the Brick Schema which we use to produce the output that the user will see on their end. The final input is the cached embeddings from past computations. As shown in Figure 4 the backend first checks to see if the embedding is cached before making the call to OpenAI to generate the embeddings. If the embedding exists in the cache, then we use that to generate the mappings.

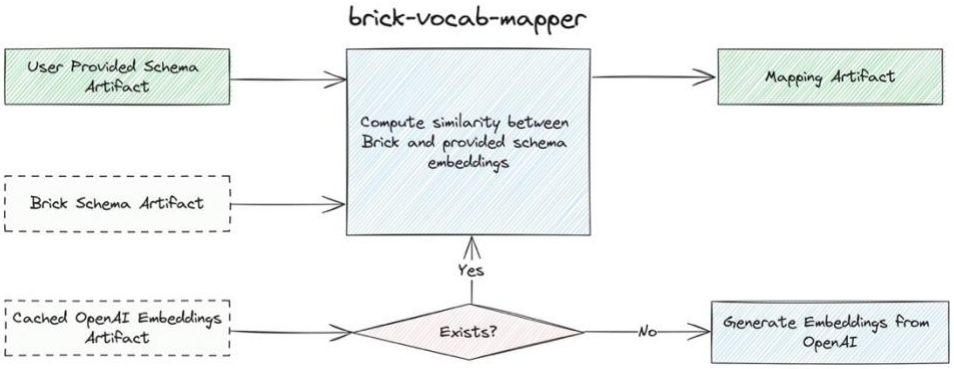

Figure 4: Backend Architecture

Embeddings caching is important to optimizing the efficiency of our system. The cache is generated using a SHA-256 hashing algorithm and then stored in a USearch index for comparison against classes in the Brick index. The cache then is maintained indefinitely using a Python shelf. The cache is also kept indefinitely ensuring maximum price reductions from OpenAI calls. The cache can be cleared if desired by the client. The usefulness of the cache grows as more users upload their vocabularies to match to Brick classes, we would be storing more embedding with every user input. Overall, the caching will minimize the need to call OpenAI to generate the embeddings as well as optimize the run time for computing the mappings.

In consideration of the backend, the Python schema library verifies that the POST body data adheres to the expected format which ensures the proper shape of the JSON object. As for security, we used Trufflehog for monitoring exposed OpenAI keys and Trivy for conducting container Common Vulnerabilities and Exposures (CVE) scanning which overall safeguarded our system against potential vulnerabilities.

In consideration of the frontend, to improve the performance, quality, and accuracy of our web application, we used Lighthouse, an open-source and automated tool. Using Lighthouse emphasizes implementation practices that ensure our application is accessible to all types of users. This was accomplished by the implementation of elements such as image ALTs, input labels, and unique IDs tailored for alternative browsers. In addition, Lighthouse advocates for the establishment of secure connections through HTTPS using Certbot and recommends configuring CORS to regulate API access. As a result, this contributes to a user-friendly web development environment.

Cross-Origin Resource Sharing (CORS) allows servers to specify the sources from which they are able to request and receive information. Both XMLHttpRequest objects and the Fetch API, which are the most common methods in JavaScript for handling HTTP, requests adhere to a single-origin policy by default [2]. This means that, unless otherwise specified in a request, these tools only request resources from their own domain or server. While this security measure provides security and safety against scripting attacks, we considered that there was a possibility it could hamper our application since we decided to push forward with a disparate frontend and backend.

Flask-CORS is an open-source extension for Flask that allows us to make cross-origin requests while using the Flask framework. The extension features a level of granularity that allowed us to limit cross-origin requests to specific resources, or to allow requests from any domain [3]. Using this tool was instrumental in allowing for frontend and backend communication.

VIII. Software Test and Quality

Test	Purpose	Testing Tools	Edge Cases
Many to Many Result Correctness	Ensure backend produces expected results for a known correct many-to-many schema	Python's unittest module	N/A
One to Many Result Correctness	Ensure backend produces expected results for a known correct one-to-many schema	Python's unittest module	N/A
Input Validation	Ensures user-provided input follows necessary format.	Python's unittest module	Users input a file without a definition for a term. This ensures the process can proceed and avoids any errors.
Cache Validation	Ensure cached items are correctly stored and can be accessed successfully afterwards.	Python's unittest module	Must consider when to invalidate caches that are out of date.
Secure API key	Ensure that users cannot access the API key for their own personal use	Trufflehog	N/A
Client Satisfaction	Ensure that the users of the tools can easily navigate the website and are satisfied with the results	Present demo to client. Allow the client to build locally and test.	User is not satisfied with the product or does not fill out survey honestly or meaningfully.

Table 2: Types of Software Tests

It is crucial to make sure the technical design works properly to achieve project objectives. Python's unittest module tested if the mapping tool works to ensure correct information is outputted regardless of the mapping option chosen. Python's unittest module was also used to test correctness on the format of both input and output files to make sure there is consistency. Python's unittest module also tests cached items are stored correctly to maintain the site's

efficiency. Lastly, Trufflehog ensured the API cannot be accessed to help protect the project's security. There are no extreme cases when testing mapping correctness and the security of the API key.

IX. Project Ethical Considerations

Although our project is meant to address a niche need, our team still considered potential ethical implications. Most of this project covers Principle 1, Principle 2, and Principle 3 from the ACM Principles which address the public, the client and employer, and product ethical considerations. As we developed this project, we referred to these ethical principles that allowed us to deliver a high-quality product that maintains trust with the client and public.

Principle 1 ensured that our team addressed general ethics. Our team and client had good intentions with our product solution and avoided harm that would from invalid data or results. Avoiding harm was most at risk of being violated by the team since there was a possibility that we would create a product that does not create proper mappings. If the team were to violate this principle, we would have created the risk of people losing credibility and potentially careers. Our team made sure our frontend design was fair and did not discriminate against users. We did this because there was a risk of creating a product that could discriminate against users. We therefore created a solution that is accessible to everyone regardless of physical disabilities and other identities. As we developed our product, we also made sure we did not discriminate among our own team members.

Another principle we considered was Principle 2, which covers professional responsibilities. For example, we considered that we should perform work only in areas of competence since the team may fail to get expertise. Another professional responsibility we considered was to accept and provide professional review since the team is responsible for presenting to the client and being open to review. The standards of the product were of high quality because the team wanted to avoid the risk of failing to test everything which would lead to the risk of not delivering the client's needs to its best performance and could lead to potential failures.

Finally, we kept Principle 3, professional leadership, in mind throughout the development process. The main objective for professional leadership is to create growth opportunities. This was in danger of being violated if a member of the team takes charge of a certain task and does not allow the rest of the team to have growth opportunities. Preventing growth opportunities for the team would also imply that the product solution was created through the lens of one teammate which results in poor perspective of the problem statement.

Two Michael Davis tests that we kept in mind as we moved forward with the project and faced choices were the harm test and the reversibility test. The harm test ensured that the results we produced were correct and did not result in any harm for the users. At this moment, if we apply the harm test, our project does not fail the test because we did not cause any harm during the development process. In addition, the risk of harm was lower than the benefits which are the results from the use of the product. The reversibility test allowed us to see the product the way the user would. As we made development and design choices, we considered how the users would be impacted. As of now, the project also passes the reversibility test.

X. Project Completion Status

By the end of the project, we completed most of the functional and non-functional requirements. For the front end, we have accomplished creating a user-friendly web app. The web app can take in user-submitted files and allow the user to customize their mapping parameters. As for the backend development, we have achieved the task of extending the existing script from Dr. Fierro to establish a backend and created a new endpoint that facilitates the process of users supplying an existing building schema. We were able to achieve this through using Flask to extend Dr. Fierro's mapper library. The backend can process and validate the submitted schema using Dr. Fierro's Brick mapper library to return the mapped Brick vocabulary. According to feedback, our project has met our client's standards. Unfortunately, due to time constraints, we were unable to dedicate time to improving the existing vector database solution and allowing for multiple schema input types.

XI. Test Results

The web app has been tested and is stable for its current development stage. For the backend, we've tested the results from a simple sample input, and it successfully generated reasonable mapping values. We have also timed the backend performance on a large sample input, it averaged out to be 4200 ms to compute a one-to-one mapping. We conducted a Trivy scan on the backend Docker image and all addressable CVEs were fixed. The next step would call for more unit testing as we would like to create some ground truths to programmatically ensure results are correct for different options. This is particularly difficult as mappings are partially subjective, and embedding results from OpenAI are likened to change over time. Input validation has also been tested and the backend will only accept JSON objects in the proper shape with a CSV file that has a name row. Additionally, we used Trufflehog to check if we've erroneously exposed any credentials in the repository. This was done to ensure the OpenAI key and SSL certificates for the production domain are not leaked. Additional tests that remain involve Selenium testing for the web application to see if it can handle multiple users at once and potential cache validation. The Selenium testing should cover end-to-end submission testing and rudimentary high-traffic simulations.

XII. Future Work

The Brick Schema currently defines around 800 classes, and it is being updated constantly. Due to this large number of possible options, users may want to further limit the number of results they get and consider only a subset of the Brick schema. Future endeavors with this web app should work to allow users to restrict mapping results to subsets more specific than equipment, point, or location classes.

To aid users in better understanding their results, some sort of output visualization would be a great area to explore. The Brick team has already created a tool that projects Brick's RDF code into a visual tree, so it may be possible to simply create a mapping using our tool, send it to this visualization tool, save the result, and display it back on the frontend. Of course, an in-house solution may be pursued as well. Whatever is chosen, the visualization should be interactable: perhaps more information about a class is shown when a user hovers over a node, or users may be able to edit the tree to their liking.

If this project is to be actively maintained into the future, it would be useful to include some kind of feedback system for the mappings that allows users to send written feedback that future teams can use to finetune the mapping process. This could be as simple as a textbox that allows users to write their thoughts and send them over, or a more complex system where users can flag particular mappings and suggest their own mappings.

XIII. Lessons Learned

Throughout this project, our team learned many lessons. Overall, this project gave us all the opportunity to learn full stack development. Coming into this project, the majority of us have not had the opportunity to develop a web application before. With this project, we learned how to develop and deploy a website. By working on the front end and back end separately, we learned how to connect and tie everything together.

When it comes to the frontend technical aspects, we learned how to code in JavaScript which was a language most were not familiar with. We chose to use React, a frontend JavaScript library that is used to develop applications. Learning React was completely foreign to most of the team. Through reading a lot of documentation and looking at examples, our team was able to figure out basic frontend development techniques along with how to debug in React. Using a React template really helped our team get started. We also found out that using Chakra UI would improve the appearance of our application. One of the challenges we faced was formatting and displaying the results table correctly. We learned that the Tabulator library was very useful in displaying data on a webpage and makes implementing

customized functionality straightforward. By using Tabulator on the front end, we learned a lot of new techniques in terms of customizing the table's design, functionality, and how to integrate third-party React libraries. Another challenging problem we had to solve was figuring out how to have a working dropdown menu within the table.

As for the backend, we overcame many challenges throughout its development. Again, we learned many new debugging techniques and overall, how to select technology and tools that meet the needs of our project since so many solutions exist. One of the most valuable takeaways from this project was learning the importance of containerization and how it is a very useful and effective industry method. Containerization was an idea suggested to us by our advisor which is why we decided to use Docker, a platform used for developing and running applications by abstracting the underlying operating system and filesystem of the project. The benefit of containerizing with Docker is that software can be delivered more quickly and allows for scalability. Exploring Docker resulted in us learning the fundamentals of containerization and Docker networking concepts. Subsequently, we learned the importance of container security, which is why we chose to use Trivy, a wide-spectrum security scanner with a plethora of scanners for different languages and platforms. In particular, Trivy offers CVE scanning for Docker images. This tool introduced our team to the nature of CVEs, container vulnerabilities, and static security scanning. Another tool we used was Flask, a web framework for Python. We used Flask to wrap Dr. Fierro's Brick mapper library and learned how to create an API. Nginx was used as a reverse-proxy server. Our Nginx instance receives requests and either serves our static frontend or forwards the request to the backend. Using Nginx was one of the most challenging parts of the project because there were issues getting routing information between the frontend and backend in conjunction with Docker networking. When using Nginx, we had to learn new debugging techniques to remedy these issues. Finally, we used Certbot, a tool used for automating the creation of SSL/TLS certificates for use with HTTPS. We chose to use Certbot due to its popularity and integration with Nginx. From using Certbot, we learned how to secure a website with HTTPS.

As for nontechnical lessons, by using Agile software development, we learned many valuable lessons. We learned how to communicate with our client properly and regularly. Having this regular communication not only motivated us but assured us that our team and the client were on the same page throughout the whole development process. As a result of a lesson learned from a miscommunication mistake we made early on with the client, our communication became clearer moving forward. As we progressed to each new sprint, we were able to reflect on previous sprints and reassess our capabilities so that our deliverables were reasonable and attainable for the new sprint. Therefore, we were able to ensure that we met the MVP. However, we did temporarily fall out of the habit of regularly updating our backlog therefore resulting in organizational issues. We learned that proper upkeep of the backlog contributes to the success of each sprint, and we were able to resolve this issue near the final stages of the project. Overall, working on a team taught us how to communicate with each other as well. We learned the importance of asking questions and admitting when we needed help.

XIV. Acknowledgments

On behalf of the entire team, we would like to thank our client, Dr. Gabe Fierro, for providing us with this opportunity. We appreciate the time he took to meet with us, answer our questions, and provide us with feedback. We learned a lot through his guidance and instruction. In addition, we would like to thank our advisor, Tree Lindemann-Michael, for giving us advice and feedback throughout the semester to help us be successful in our technical and professional work.

XV. Team Profile

Alejandro Estrada Silva

Senior
Computer Science + Robotics and Intelligence Systems
Hometown: Aurora, Colorado
Interests: Art, Video Games, Music

Angel Lechuga Gonzalez

Senior
Computer Science + Data Science
Hometown: Greeley, Colorado
Interests: Racquetball, Reading, Video Games, Salsa Music

Tiana Nguyen

Senior
Computer Science + Computer Engineering, Cyber Security
Hometown: Highlands Ranch, Colorado
Interests: Hiking, Concerts, Cello

Michelle Torres

Senior
Computer Science + Data Science
Hometown: Aurora, Colorado
Interests: Basketball, Music, Hiking

Donovan Keohane

Senior
Computer Science
Hometown: Branford, FL
Interests: Reading, Music, Linux, Cooking

References

[1] B. Balaji et al., “Brick: Towards a Unified Metadata Schema For Buildings,” Nov. 2016, doi: 10.1145/2993422.2993577.
[2], “Cross-Origin Resource Sharing (CORS),” Mozilla.org. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS. [Accessed: 16-Sep-2023].
[3], “Flask-CORS — flask-Cors 3.0.10 documentation,” Readthedocs.io. [Online]. Available: https://flask-cors.readthedocs.io/en/latest/index.html. [Accessed: 16-Sep-2023].

Appendix A – Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
Large Language Model (LLM)	*A deep learning algorithm that intakes a high volume of data to understand or accomplish a specific task dealing with natural language*
Schema	*The organization of a database.*
Ontology	*An abstraction of data models that is used to model knowledge regarding individuals and their attributes. Often ontologies are used to model concepts and relationships.*
Mappings	*A match from a set of items to another set of items.*

One-to-one mapping	*A single set of elements is matched with another single set of items. In this case, every singular vocabulary input has one matching output.*
One-to-many mapping	*A single set of elements is matched with multiple sets of items. In this case, every singular vocabulary input has one or many matching outputs to choose from.*
Many-to-one mapping	*Multiple vocabulary inputs is matched with one set of items. In this case, every various vocabulary input has one matching output.*