HEBSE
Holistic Exploration of Binary Stellar Evolutions
May 2025

SDMAY25-20
Client and Advisor
Dr. Goce Trajcevski

Byrd, James
Collins, Eamon
Norris, Alek
Polston, Alexander
Snyder, Andrew
Varnitskyy, Svyatoslav
sdmay25-20@iastate.edu
https://sdmay25-20.sd.ece.iastate.edu

Executive Summary

Astrophysicists using the POSYDON synthesis tool needed to be able to easily access simulated results to find meaningful connections for binary star systems in order to facilitate research and further their own understanding. Currently there exists no system to allow Astrophysicists to effectively query POSYDON's generated data. Our created application hosts and allows accurate querying of the simulated data regardless of prior database querying experience. The design created includes an intuitive user interface (UI), as well as the implementation of natural language processing using AI, allowing a user with no prior querying experience to fully utilize the application. Our created application consists of four main components: frontend, data parser, backend, and database. The frontend was designed using React, Typescript, and Material UI. These were chosen because they are the industry standard. The data parser and backend were written with Python because it is easily developed and widely supported. The backend specifically utilizes the FastAPI web framework for its modern and efficient performance. Over the past year we have completed development of our application including a completed user-friendly interface, a fully functional backend including a script to parse various versions of the data into the database, and the full implementation of the latest publicly available versions of OpenAI GPT. We are confident our design meets all of the intended users' needs because of our strict adherence to our detailed design plan and consistent communication with our client to ensure requirements are met.

Learning Summary

Development Standards & Practices Used:

- ISO/IEC/IEEE 12207:2017 Systems and software engineering - Software life cycle management
- ISO/IEC/IEEE 24748-3:2020 Systems and software engineering - Life cycle management
- ISO/IEC/IEEE 29148:2018 Systems and software engineering - Life cycle processes
- ISO/IEC 9075-1:2023 Information technology - Database languages SQL
- IEEE Code of Ethics

Summary of Requirements:

- Develop a system that would take the original data and import it into a database to allow easy, accurate querying of the simulated data
- Intuitive UI that enables the users to either write their own SQL queries or construct SQL queries with the help from AI
- Implement several ("sample") SQL queries as a proof-of-concept

Applicable Courses from Iowa State University Curriculum:

- COMS 3090 (Software Development Practices)
- ENGL 3140 (Technical Communication)
- COMS 3190 (Construction of User Interfaces)
- COMS 3630 (Database Management Systems)
- CYBE 2340 and CPRE 3940 (Ethics)
- COMS 228 (Introduction to Data Structures)

New Skills and Knowledge acquired that was not taught in Iowa State courses:

- Programming with Typescript
- Programming with REACT
- OpenAI API Usage
- Prompt engineering

Table of Contents

Figures and Tables

Figures:

Tables:

1. Introduction

We now provide an overview of a problem domain and a discussion of the intended users.

1.1 Problem Statement

Astrophysicists studying binary stars use an application called POSYDON (POpulation SYnthesis with Detailed binary-evolution simulatiONs) to simulate binary star interactions. A binary star system consists of a pair of stars that orbit each other due to gravitational pull. Studying stars directly can be time-consuming and costly, so astrophysicists often conduct their research via simulations before verifying their results with real astronomical observations. POSYDON studies these binary star systems.

Given the number of simulations it performs and the criteria it collects, POSYDON generates a large amount of data and stores this across numerous spreadsheet files in Comma Separated Values (CSV) format. Due to the dispersed nature of the data, there is currently no efficient way to search and analyze the information. To address this, our project provides a software tool to import and query this data in a common location. Following data processing, users will be able to search collected binary star data by writing custom queries using Structured Query Language (SQL), or by standard English language requests that will then be translated through a large language model (LLM).

Without a tool such as this, users would have difficulty formulating their requests for obtaining particular subsets of the simulated data that satisfies their desired properties. Specifically, for every desired query, a complex program would need to be constructed - as opposed to having a compact, high-level language providing a declarative ("what") without the need for procedural ("how") details. By simplifying data access and supporting custom data, this application aims to make binary star data querying more accessible to students, educators, and researchers, allowing for greater collaboration. In addition, similar queries can be reused almost verbatim by changing the values in their parameters.

1.2 Intended Users

Our intended users are divided into three primary groups: astrophysicists, educators, and students. Each of these groups has different needs when utilizing our tool.

Astrophysicists primarily utilize the data produced by POSYDON to compare data points between the stars. However, they are often very busy, and the current offerings do not support efficient access to this dataset. Many of these researchers may not be familiar with querying languages such as SQL. As a result, they need a tool that can efficiently retrieve the data using natural language queries instead of requiring complex SQL commands. In turn, our product will provide opportunities for increased productivity for researchers by reducing the need to look through multiple files. Instead, they will be able to write a short request and receive the data they need in a timely manner.

Educators would primarily use the tool to help facilitate an engaging and interactive classroom learning environment. Since they may not be experts in binary star systems, educators would need this tool to be intuitive and easy to use so that creating exercises and classroom activities is simple. This would assist educators in showing students the complex relations between these astrophysical events and how they connect to larger cosmic anomalies.

Students will primarily use this tool for their own study of binary star systems to assist with coursework and personal research. Since students may not be familiar with SQL, students need a tool that provides an intuitive, user-friendly interface that simplifies the querying process. This simplified tool will help the student focus on understanding and analyzing the data rather than struggling with the technical challenges.

2. Requirements, Constraints, and Standards

As part of a completed project that meets client needs, it was crucial that their requirements were met while sticking to our constraints and meeting engineering standards.

2.1 Requirements, Constraints and Standards

We met various requirements for our project to be successful. These requirements follow the categories of functional, resource, physical, aesthetic, user experiential, and economic

2.1.1 Functional Requirements

- The tool correctly converts CSV files into a PostgreSQL database
- The application correctly convers natural language into SQL queries through OpenAI GPT integration
- Previous Requests are easily retrievable during that user session

2.1.2 Resource Requirements

- TypeScript
- Node.js
- React
- PostgreSQL
- Python
- Sufficiently powerful computer to run the tool; specifics depend on the desired response time

2.1.3 Physical Requirements

- Users should have a powerful computer capable of hosting a large database as well as the frontend application or a computer to host the frontend application and an outside machine capable of hosting a large database

2.1.4 Aesthetic Requirements

- Easily navigable User Interface

2.1.5 User Experiential Requirements

- Ability to query a database without technical knowledge of querying language

2.1.6 Economic Requirements

- Open AI Input Tokens ~ *Optional requirement if NLP is desired*

2.1.7 UI Requirements

- Visual panel for viewing query resolutions
- Button to download query resolutions as CSV files
- Query input options for both NLP and SQL queries
- Pages for setup and general settings
- Query history

2.1.8 Privacy and Security Requirements

- These are not the main functions of the program. However, the program provides local encryption of credentials, and tunnels information through SSH for encryption, requests to OpenAI are also handled through HTTPS and encrypted by OpenAI.

2.2 Engineering Standards

Our project aimed to adhere to several engineering standards to ensure a quality product.

2.2.1 ISO/IEC/IEEE 12207:2017 Systems and software engineering - Software life cycle management

- This standard specifically outlines the definitions and requirements dictated by the software lifecycle. The definition of the individual aspects of our software and how they are handled throughout the development and maintenance cycle.
- Our project adhered to this standard as we developed our software. A taut structure during our development helped ensure that the final deliverable was a product that is fully functional and exceeds the expectations of its user base.

2.2.2 ISO/IEC/IEEE 24748-3:2020 Systems and software engineering - Life cycle management

- This standard specifically outlines a framework and establishes a guideline for implementing standard 12207:2017 for the software life cycle process. This standard provides methods to ensure consistency throughout the development process and acts as a technical guidance tool for implementing standard 12207:2017
- Our project implemented this standard because it went hand in hand with and provided guidance and assistance in implementing 12207:2017.

2.2.3 ISO/IEC/IEEE 29148:2018 Systems and software engineering - Life cycle processes - Requirements engineering

- This standard outlines the required engineering provisions for the process and products throughout the software development life cycle, defining how to create good requirements and pinpointing the tools required for proper iterative and recursive development. This standard is, in part, a guideline for 12207:2017.
- Our project implemented this standard because it goes hand in hand with and provided guidance and assistance in implementing 12207:2017.

2.2.4 ISO/IEC 9075-1:2023 Information technology - Database languages SQL

- This standard outlines the minimum requirements for what a database engine should fulfill to ensure proper SQL syntax for interpreting SQL queries.
- Our project implemented this standard, as the primary function of our project was to create an application that queries an SQL database.

2.2.5 IEEE Code of Ethics (2020)

- This code outlines the professional responsibilities of engineers and technology professionals. It defines the ethical obligations of individuals in ensuring safety, integrity, and fairness in their work, as well as fostering trust and transparency within the broader community.
- Our project and the team's activities adhered to the IEEE Code of Ethics ensuring that we operated with the highest ethical standards, safeguarding public welfare while maintaining honesty in all claims and technical practices. This adherence guided our team in managing conflicts of interest, making responsible decisions, and ensuring that the technology we develop prioritizes the health, safety, and welfare of the public.

3. Project Plan

In this section, we discuss various aspects of our project plan and timeline.

3.1 Project Management & Tracking Procedures

To maintain our project, our team utilized an agile style. This methodology ensures that we easily adapted to the challenges we faced at each step of the project process. By splitting our work into shorter sprints, we set short-term deadlines that allow us to better seek feedback and potentially pivot without losing substantial progress.

To decompose our tasks and assign them to group members, our team kept track of progress by utilizing the Gitlab issue board. To facilitate communication and ensure all team members are making progress and not running into roadblocks, we utilized Discord for regular communication. Through this medium, we were able to discuss the division of topics, schedule meetings, track important information, and keep well-documented meeting minutes.

3.2 Task Decomposition

To delineate and break down the tasks that needed to be accomplished to complete the final software deliverable, we broke the project into discrete conceptual tasks. These tasks cover the main sets of functionalities that we need to implement. The tasks are broken down into four groups represented by different colors. Yellow represents database and data parsing tasks, red represents documentation, blue represents UI development, and orange represents our SQL querying system. All of this can be seen below in the diagram.

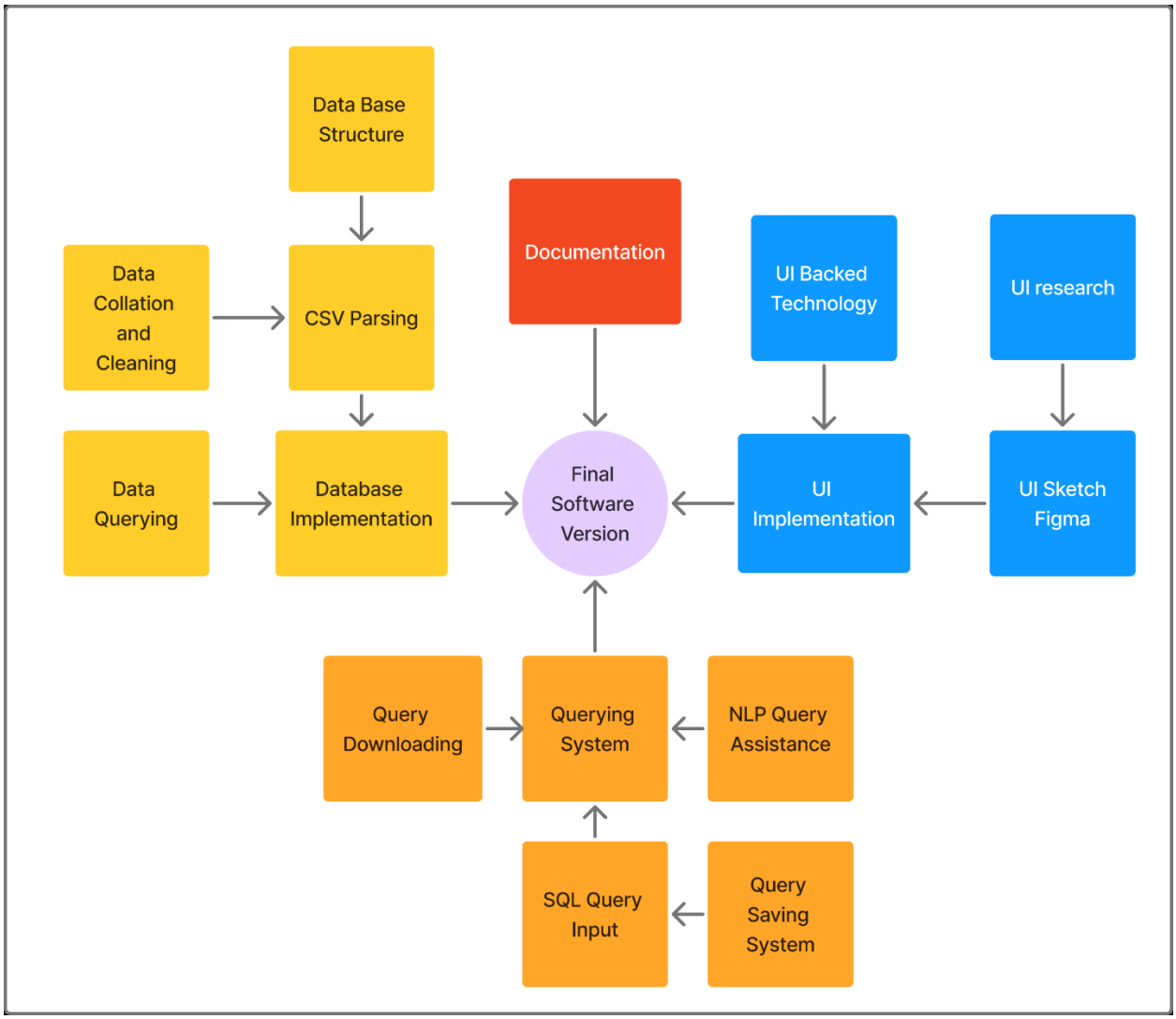

Figure 3.1 - Task Decomposition

The tasks from the task decomposition are the following:

3.2.1 Database and Data Parsing (Yellow)

- Formulate Database Structure
  - Create a schema that easily matches the raw data into the tabular format.
- Create and test Data Collation and Cleaning system
  - Ensure that all processes relating to data loading work as expected.
- Develop CSV Parsing and Data normalization System
  - Create subject scripts to parse data from CSV files.
- Integrate Database at Rest saving system
  - Implement mechanisms to ensure the data integrity when not being accessed.
- Fully Implement and Integrate Database
  - Integrate the database into the larger system, ensuring that data is properly handled.

3.2.2 Documentation (Red)

- Create Project Documentation
- Create User Documentation
    - Ensure the user knows the features of and how to use the program.

3.2.3 User Interface (Blue)

- UI Research
    - By researching potential libraries/frameworks we wanted to utilize for developing our project. We ensured consistent design and easier development by pre-establishing our desired tech stack.
- Develop UI Sketch in Figma for all UI pages
    - By creating Figma mockups of what we want our screens to look like, we were able to work towards a goal and establish how our design should appear prior to implementation, resulting in less wasted time.
- Develop UI without full logic
    - By creating a basic UI without all of the logic, we could determine any visual changes that need to be made. We then had a better understanding of how the Figma mockups translated to reality.
- Develop and Finalize Full UI Implementation
    - By implementing the natural language processing API, all logic, and database connection we were able to ensure all basic functionality.

3.2.4 SQL Querying System (Orange)

- Develop AI-Powered Natural Language Query Formatting System for our Database and Schema
    - We built a system that takes user-generated natural language queries and provides them to OpenAI for transformation into structured queries for the implemented database.
- Develop and Integrate Query Saving System
    - We implemented a system to store, save, and retrieve custom queries for easy access and reuse.
- Develop and Integrate the Structure-based Query Formatting System
    - We ensured that queries are checked for correctness syntactically.
- Finalize and Integrate the Structure-based Query Formatting System
    - Combine the natural language, saved query, and structure-based query system into one system.

3.3 Project Milestones, Metrics, and Evaluation Criteria

Milestones:

1) Set up the development environment for all members
2) Create Figma mockups for front-end and consult with the client
3) Create UI components in React
    a) Query Page
    b) Query History Page
    c) Data Upload Page
    d) Settings Page
4) Create Database schema
5) Set up CI/CD for the project
6) Implement data parsing solution
7) Implement the backend server for data hosting
8) Implement OpenAI API for Natural Language Processing
9) Attempt to query data
10) Polish and finalize backend and data parsing solutions after testing
11) Polish and finalize Frontend design
12) Create final user manual

3.4 Project Timeline

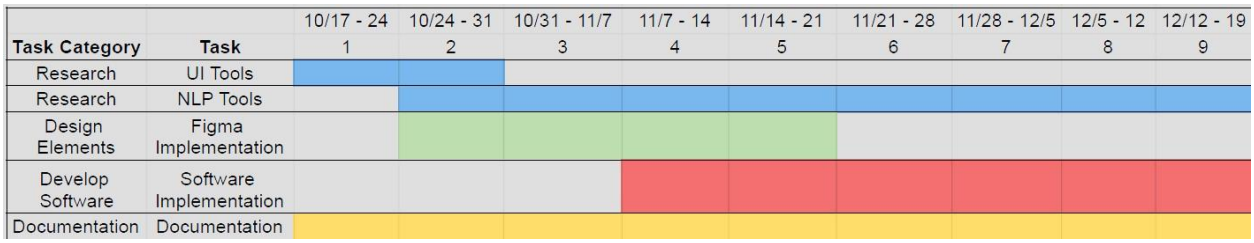

Figure 3.2 – Fall

Figure 3.3 – Spring

Our development cycle was broken down into multiple sprints with events in chronological order from the top down.

- **Sprint 1: October through December**
  - Project Planning
  - Product Research
  - Documentation
  - UI Creation in Figma and with React
  - Begin creation of NLP implementation methods
  - Test UI's ability to handle NLP
- **Sprint 2: January**
  - Finalize design of database structure
  - Finalize design and list of required database tools
  - Begin creation of database structure
  - Begin creation of database tools
  - Test back-end communication with the database
- **Spring 3: February**
  - Continue creation of database structure
  - Continue creation of database tools
  - Continue creation of NLP implementation methods
  - Begin mapping front-end buttons to back-end tools
  - Test and implement currently created tools using front-end UI
- **Spring 4: March**
  - Finalize database structure
  - Finalize database tools
  - Finalize NLP implementation
  - Finalize mapping front-end buttons to back-end tools
  - Begin optimization of database tools
  - Begin optimization of NLP responses
- **Spring 5: April**
  - Finalize optimization of database tools
  - Finalize optimization of NLP responses
  - Finalize documentation of the final deliverable
  - Encrypt local credential storage
  - Make UI more welcoming
  - Finalize user manual

3.5 Risks, Risk Management, and Mitigation

Table 3.1 – Project Risks and Mitigation

Risks	Risk Probability	Mitigation
Missing the desired deadline	0.2	Our team has established a solid communication channel and regularly meets to determine the progress of our project. If a team member is struggling, our group will work together to ensure the deadline is met.
Software issues	0.3	We are using frameworks and libraries that are already on the market. If we encounter a limitation, we need to prepare to switch technologies.
Database taking too long to query	0.4	When dealing with large datasets, there is some uncertainty regarding queries as it could take longer than expected to process. One mitigation strategy would be to provide an index of certain columns. Another potential mitigation strategy would be to partition the database (theoretically). When it comes to hardware specific limitations, mitigations go beyond the scope of this project.
Natural Language Processing is not working correctly	0.4	OpenAI GPT works well for SQL queries based on a simple schema. However, with multiple terabytes of data and parameters that are not as easy to understand, we might encounter scenarios where generated queries contain invalid syntax. We must thoroughly test queries generated by the AI model to ensure our expected results make sense and attempt to interact with and guide GPT toward improving the syntax.

3.5.1 Encountered Risks, and subsequent actions

During development we encountered two large risks and one small, the storing and potential leakage of sensitive information such as credentials, and the inability to obtain the latest dataset from POSYDON. In addition to the above two large risks, one smaller risk we encountered was the existence of much lower powered personal computers.

To address this first risk, we implemented frontend encryption for all the stored data; the API key, and the SSH information. This information is then decrypted when it is needed and never stored in plaintext.

To address the second risk, while designing the parsing script we ensured that it strictly followed all found conventions of the h5 files contained in the partial datasets we could obtain.

To address the third smaller risk that a user might have a weak personal computer and the inability to host the backend on their local machine, we implemented the ability to choose between hosting the backend on the user's computer as well as on a remote server along with the database.

3.6 Personal Effort Requirements

Table 3.2 – Personal Effort Requirements

Task	Projected Effort
Develop a user interface	60 Hours
Determine the required tables for the database	5 Hours
Develop a tool to transfer data to the database	20 Hours
Handle natural language processing via OpenAI	30 Hours
Handle communication between the front-end application, the back-end application, and the database	20 Hours
Develop software to analyze queries and determine compliance with the schema and accuracy of the original request	15 Hours

3.6.1 Actual Effort Requirements

Table 3.3 – Actual Effort Requirements

Task	Time spent
Develop user interface	136 Hours
Develop tool to transfer data to the database and fill required tables	18 Hours
Developing, testing, and integrating natural language processing via OpenAI.	15 Hours
Handle communication between the front-end application, the back-end application, and the database	32 Hours
Develop Query Saving and History features	12 Hours
Develop User Manual	10 Hours

3.7 Other Resource Requirements

We required a local server with large amounts of storage to host our testing database and run our back-end application. In addition, this requires a continuous connection between itself and the device running the front-end application. Beyond this, we required OpenAI API input tokens to ensure we correctly retrieve queries given natural language. As a largely software-based project, most of our resources depend on developers owning personal computers. In sum, the development efforts throughout the project did not cause any unforeseen financial strain.

4. Project Design

To complete this project, our group needed to deeply explore potential designs and come up with a solution.

4.1 Design Context

We now discuss the context behind design decisions made by our team.

4.1.1 Broader Context

Our project was primarily designed to be of use to astrophysicists who are studying binary star systems. These astrophysicists needed a tool that allows them to easily host and query the data generated by the POSYDON project. The two secondary communities this program is designed for are professors and graduate students who are either interested in binary star systems or are looking to use this tool to prepare datasets for teaching.

To address any environmental and economic concerns we decided to make the AI-powered language processing optional so that the program can be used totally free of charge to the end-user if desired, as well as drastically decreasing the environmental impact from power generation for AI-processing.

4.1.2 Similar Projects and Related Works

Currently there are no applications that serve the same purpose for the same clientele, as an application to create and query a binary star database utilizing the data generated by the POSYDON project. This, however, does not mean that there are no unrelated applications and organizations. Beginning with the POSYDON project, they are a group whose mission is to create evolution models and generate full stellar structures of binary star systems. It should be noted that while our application was created to generate a database and querying system for this information, we have no relation to the POSYDON project themselves. Utilizing an application, such as DBeaver, it is possible to access the same data we are with our project, but our project offers a more friendly interface and setup process for less technically inclined users.

From the institute of Astronomy, in Moscow, Russia there is the Binary Star Database – BDB project. This project compiles and synthesizes data from published catalogs and databases on both binary star systems and multiple star systems, multiple star systems are systems of a higher order. This system has a web-interface with a strict input method of dropdown menus allowing a user to select a variable such as mass, and filter to systems within a specified range. While this is simpler than our SQL querying system it does not allow for the same amount of freedom in querying.

4.1.3 Technical Complexity

In the original proposal for our project there were no AI-powered features described, and the deliverable was to make a tool that allowed the simulation data to be accessible. We as a team felt that this was not complex enough so we, alongside Dr. Trajcevski decided to add AI-powered natural language processing so that users unfamiliar with SQL can use the program.

4.2 Design Breakdown

We now discuss our methodology for designing our project.

4.2.1 Design Decisions

Due to the nature of our project, our design decisions consisted primarily of software and technology selections. These include database management systems, user interface (UI) design, and generative AI model options.

One of the most important decisions we made was which database management system we wanted to utilize. We decided on PostgreSQL for several reasons. The most important reason we chose this is due to it being a relational database. Given the large dataset, a more structured database would better support retrieving relevant information. However, as opposed to other relational databases, PostgreSQL allows us to store data in custom data types. This allowed us to group aspects of data together for easier navigation and querying.

While most of our choices are purely for development and performance purposes, the UI design directly impacts the user experience. This made it important for our design to be visually appealing and easy to use. As a result, we decided to implement our UI through the React library for TypeScript. While there are other similar libraries available, we chose to use React largely for its support of Virtual DOM (Domain Object Model). This allows us to quickly modify visual elements of the application as they need to be changed. Furthermore, React has a large user base and as a result, has many resources for speeding up development and enhancing applications.

We also considered which AI language model would handle natural language processing. This leads us to choose OpenAI. We chose OpenAI due to the large number of tools that support our application and its sophisticated implementation.

4.2.1.1 Changed Design Decisions

During development we pivoted from multiple planned design decisions, in particular we decided to forgo the usage of the LangChain Python package for handling AI integration. As this is an externally created package with the intent of simplifying communication with the Open AI API at the cost of reduced customization, we decided to communicate with the OpenAI API directly to afford users the ability to tweak their experience to fit their needs.

Initially we had also planned to have the returned translated query automatically checked by the backend for validity, and if deemed invalid it would be returned to GPT for reformatting. We ultimately decided against doing this as databases may not be consistent in structure for different versions of the dataset. Without being able to obtain the latest dataset we would be incapable of ensuring a 100% accurate return rate of translated queries. Instead, we decided to leave validation of these generated queries up to the user as the most common issues fall to the AI misunderstanding the names of attributes utilized in POSYDON's datasets.

Initially during our planning phase, we had only planned to include the latest available version of the POSYDON dataset, midway through our development we decided instead to include the ability to load multiple versions of the available datasets inside the application at will. This decision was made to increase user freedom and allow our users to choose which version of the dataset they want to query at any given time.

4.2.2 Ideation

Figure 4.1 – Ideation

4.2.3 Decision-Making and Trade-Off

Our decision-making process for design choices consisted of discussions amongst ourselves, personal knowledge, research, and input from our faculty advisor to compare the pros and cons of our options. When selecting a database system, we considered SQL and NoSQL database technologies and decided to use an SQL-based database. This was because our data is heavily relational, consisting of many (~100) attributes linked to each star and stored in CSV format. This made the querying functionality of an SQL database optimal for ease of use, speed, and storage of queries. Among the many options available for SQL database systems, we chose PostgreSQL because of its capability for flexible, complex querying, as well as the ability to create custom data types and not strictly limited to tables. As each star has many attributes that can be stored, PostgreSQL enabled us to manage our data much more efficiently through custom data types, such as information on star density.

4.3 Final Design

To implement our project, we developed a detailed design consisting of deeply integrated subsystems.

4.3.1 Overview

Our project consists of four main subsystems. The user interface (UI), utilizing the React library for TypeScript, visualizes the application for the user. It allows the user to make natural language or SQL queries, displays queried data in an easy-to-understand format, and enables users to save their queries for later use.

The UI is connected to the back-end application. This system is responsible for communication between the UI and database as well as the mode of communication with OpenAI for handling natural language processing. When a user sends natural language to the backend, it sends this text alongside the database's blueprint to OpenAI. In response, GPT returns a properly structured SQL query.

When GPT translates a query, it is first returned to the user for a validity check, if the user deems the query not valid, such as in the case where the element "Helium" is labeled "Hel" instead of the correct "He" the user is provided the opportunity to modify the query or try again with GPT. If the query is deemed valid, the user may copy the query into the query box to retrieve data from the database.

The database is a relational database managed by PostgreSQL. It manages the data and returns the desired data upon retrieving a query from the backend. At this point, the backend sends this retrieved data to the user interface to be displayed. The final subsystem is the data parser. The data parser is a script that retrieves the CSV files generated by POSYDON and parses them. This parsed data is then stored in the database. To simplify the set-up for the user, a function is included in the "utilities" page that allows a user to choose a particular version of the dataset and automatically runs the data parsing script.

4.3.2 Chosen Technologies

Each technology within the system has been selected and integrated based on its unique capabilities to address specific functional and non-functional requirements. These choices support efficient data processing, user accessibility, and seamless interaction between the system's components. The following descriptions outline the purpose, role, and operations of each technology and component in achieving an optimized user experience, robust data management, and high query performance across extensive datasets. For ease of understanding, please refer to Table 4.1.

Table 4.1 – Final Detailed Design

System Component	Chosen Technology	Role	Operations	Considered Technologies & Alternatives	Pros	Cons
User Interface (UI)	React & TypeScript	Provides user interaction point; captures inputs and displays results	Captures user input for queries, provides dynamic data display, user settings	Angular, Vue.js	Flexible, widely adopted for UI; strong community support	Complex setup compared to simpler UI libraries
Backend	Python	Central processing layer for data routing, validation, and AI integration	Manages commands, connects components, validates, and processes queries	Node.js, Django	Efficient for scripting and data handling; extensive libraries	May need additional libraries for certain backend tasks
Database	PostgreSQL	Stores parsed data and allows SQL querying	Supports relational queries and data retrieval from parsed CSVs	MySQL, SQLite, DynamoDB, MongoDB, Neo4j	Robust for handling large datasets; advanced SQL support	Requires setup and maintenance; not as lightweight as SQLite
Data Parser	Python (Custom-built)	Parses and normalizes raw CSV data and prepares it for database entry	Reads, cleans, normalizes CSV files, estimates missing values, ensures consistency	Pandas, SQLAlchemy	Customizable; optimized for POSYDONS complex data requirements	Requires custom coding for specific data parsing needs

4.3.3 Detailed Design

User Interface (*UI*): Built using React and TypeScript, this layer provides users with interactive elements and visuals. It captures the user's input (e.g., SQL or NLP queries) and displays the query results, assisting in facilitating an intuitive end-user experience.

Backend: Manages requests from the UI, relays them to OpenAI GPT for NLP processing, validates SQL requests and responses, and communicates with the PostgreSQL Database while also supporting the parsing and initialization of the database. The backend acts as the bridge between the UI, database, and the AI model.

Database (*PostgreSQL*): The database system stores the data from the parsed and normalized CSV files and enables the SQL querying of the data.

Data Parser: Built using Python, the Data Parser Interprets the raw CSV files from POSYDON and normalizes them to make meaningful SQL queries possible. Reads the data from the CSV files and applies normalization methods to allow comparisons more readily between different simulations and enters the data into the Database.

4.3.4 Component Breakdowns

User Interface (*UI*) with React:

Role: Provides the primary user interaction point, enabling users to input SQL queries, view results, save, import database, and configure personal settings such as the ability to use OpenAI GPT for NLP.

Operations: Allows us to create and utilize interactive and dynamic functions that can be reused across the code for simplicity and continuity

Backend (*Python*):

Role: Serves as the central data processing layer, routing commands, validating queries, formatting responses, and the primary means of interconnecting components.

Operations: Connects the UI, database, and OpenAI handling all data routing. Depending on whether the user is using natural language or raw SQL, the backend either directly processes the SQL query or uses GPT to generate a structured query. It then validates all queries to ensure security and accuracy. Validated queries are executed, and the results are temporarily saved for the session before being sent back to the UI for display.

Database (*PostgreSQL*):

Role: Central data repository, where parsed CSV data from POSYDON is organized

Operations: Structured to handle relational queries, the database supports complex data types, allowing efficient data retrieval. The data relationships are designed to reflect the structure within the original CSVs that were used to build the database.

Data Parser (*Python*):

Role: Parses Raw POSYDON data, normalizes it, and enters it into the database

Operations: The parser, integrated with the backend, reads and interprets CSV files while standardizing data types. During this process, it performs initial data cleaning, including normalization, to ensure a uniform structure across entries. If gaps in the data are found, the parser estimates missing values by averaging similar local data points. This step is especially crucial given the massive datasets generated by POSYDON, which uses machine learning to simulate binary star systems. The parser ensures data integrity and accuracy, preparing all datasets for efficient storage and querying within PostgreSQL tables.

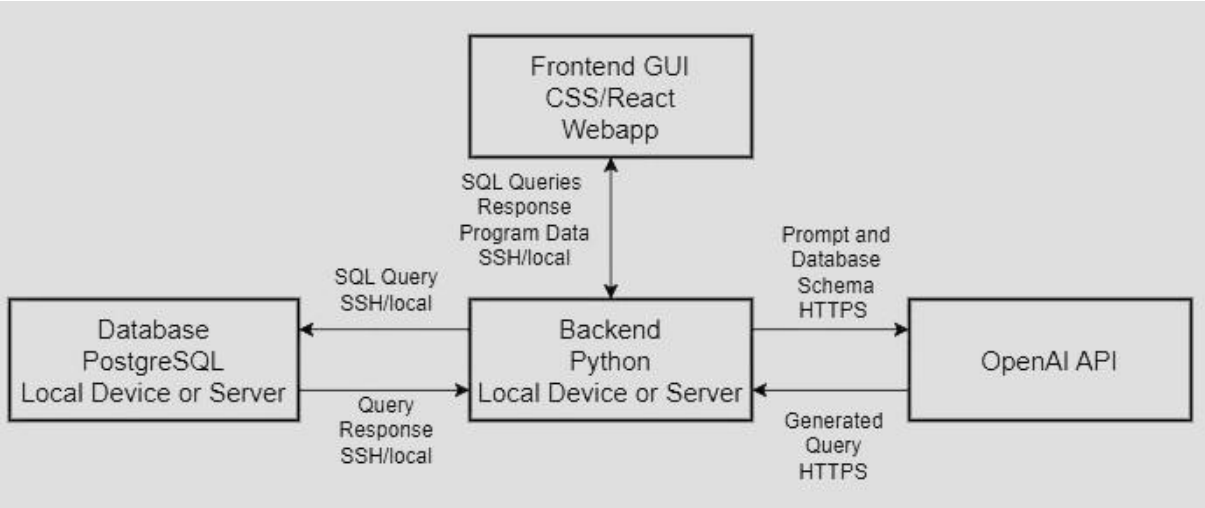

Figure 4.2 – Final Visual Representation

4.3.5 Functionality

In a typical use case of our application, an end user would use a local server or a high-spec personal computer to run a set-up script to host the database. The end-user would run the desktop application used to parse the database, and running this, the application would also build the database and connect it to the application. After a successful connection, the end user would be able to perform two types of queries: a query using their own constructed SQL statement or an SQL statement generated through a machine learning-powered natural language processor, which would translate their request to a properly formatted SQL query. This query would then be performed, and a small subset of the data would be returned to the home screen of the application for preview. The whole result would then be available for download to a file on the end user's machine. The user could then perform another query or close the application. Including the above-described local application and remote database configuration, after set-up is completed, a user can decide whether to run the application and database: all locally, all remotely, with just a remote database, and with a remote database and backend.

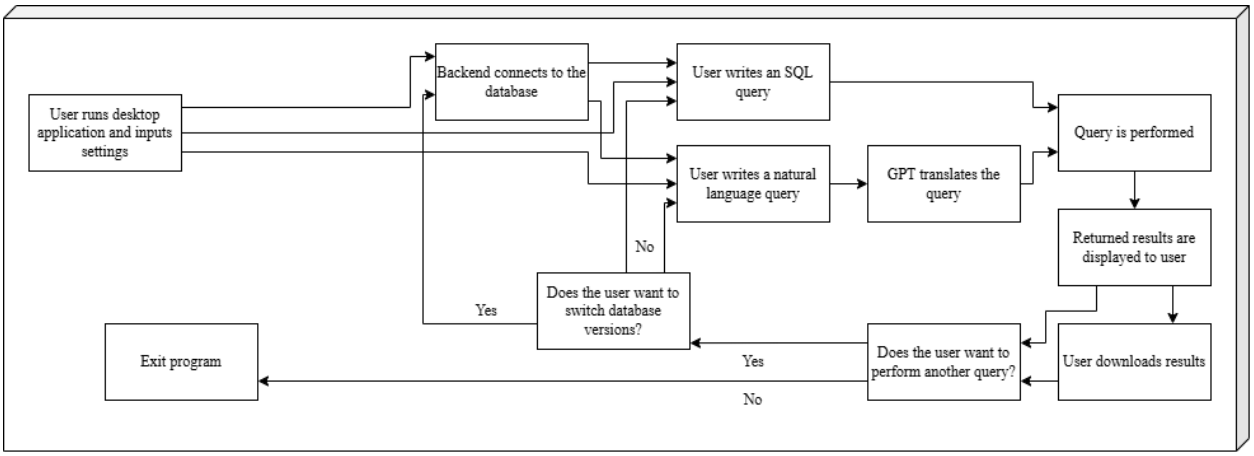

Figure 4.3 – Final Task Flow

4.3.6 Areas of Concern and Development

Going into this project, we determined our requirements and user needs. At its core, our requirements are to build a database that stores the data generated by POSYDON simulations and to allow users to easily retrieve data from this database. Beyond this, we noted that non-technical users would make up a large portion of our user base. As a result, we created several additional requirements to support these users. These users need to be able to set up the database without necessarily knowing how to maintain a database, and they need to have the option of querying the database without necessarily having a detailed knowledge of SQL and its syntax.

Our project fulfills these requirements through each of its major components. To solve our initial requirements, we chose to store the data in a PostgreSQL database. As a result of its relational nature, SQL is easy to develop queries for, ensuring the database consistently retrieves all desired data when queried. This is handled even further through our front-end user interface. Through this, users are able to enter queries without interfacing with the database directly. Similarly, this user interface supports our requirement of allowing non-technical users to query the database. By integrating our user interface with OpenAI GPT via a textbox for natural language, users can find relevant information from the database without writing SQL queries. Furthermore, these users can initially set up the database through our database generation script. This ensures users have no need to understand relational databases to set this up.

Given this design, our primary area of concern is our reliance on an LLM. Though OpenAI's GPT is improving over time, it does not always follow prompts well, to counteract this we constructed a very specific and detailed prompt that outlines exactly what the AI is allowed to do and the bounds that it is to be constrained by. Even with this tightly bound prompt, the returned translated query is not guaranteed to be 100% accurate, because of this the query is first returned to the user for verification of items such as the specific variables that the user is performing the query for such as mass, density, or elemental compound.

4.4 Technology Considerations

This section describes the distinct technologies used in our design, including their strengths, weaknesses, trade-offs, and any alternative solutions considered.

4.4.1 Windows

Windows is a widely used, industry-standard operating system requiring significant computational power to run effectively on modern hardware. Given its performance capabilities, a personal computer with an up-to-date version of Windows provides adequate processing power to run our program, initialize the database, and execute queries efficiently. Windows also offers compatibility with all the frameworks in our project and simplifies GUI design. However, a major drawback of using Windows is its lack of cross-compatibility with other operating systems, such as Linux and macOS, which some users may prefer.

4.4.2 Python

Python is a high-level programming and scripting language known for its flexibility and ease of use. In this project, Python serves as the main back-end language, providing frameworks that facilitate connections between the user interface, backend, database, and OpenAI API. Python's cross-platform compatibility allows it to be installed on virtually any modern operating system. However, Python's high-level nature results in increased overhead, particularly when compared to lower-level languages like C, C++, or Java, potentially impacting performance in high-demand use cases.

4.4.3 PostgreSQL

PostgreSQL is a powerful, open-source relational database management system (RDBMS) chosen for its secure and reliable data storage, robust support for complex queries, and ability to handle large datasets—ideal for research environments. Its ACID compliance ensures data integrity and flexibility. However, PostgreSQL's complexity requires careful tuning for optimal performance, particularly with intricate datasets like those from the POSYDON Project. This can increase the need for regular maintenance and updates to ensure proper storage and organization. These trade-offs between advanced functionality and the need for diligent optimization are key considerations in our design.

4.4.4 OpenAI (GPT)

OpenAI's GPT provides extensive support for natural language processing (NLP), allowing our system to assist users with minimal SQL knowledge to interact seamlessly with the database. This integration enhances the user experience by providing an accessible, AI-driven interface for those who need it or those who want ease of use. OpenAI's API enables streamlined responses by associating queries with a unique API key. This key can be shared among people, or individuals may purchase one for their own personal needs. By using this unique API key, individuals are able to offload computational load to OpenAI's servers, reducing local system overhead. Compared to other large language models, GPT stands out for its ease of setup, low computational overhead, and outstanding customer support.

4.4.5 React

React is a JavaScript (JSX) library developed by Meta (Facebook) to facilitate the development of single-page applications (SPAs) with streamlined user interfaces. React's component-based architecture enables modularity and code reusability, reducing the need for redundant code and allowing dynamic page rendering with minimal updates. Additionally, React's extensive library ecosystem offers pre-built components for customized user interface design. However, React's reliance on JSX presents a learning curve for our team, as it is a new technology for most members. Despite this challenge, React's modular approach and rich feature set made it a promising choice for developing a responsive, maintainable interface.

4.5 Design Analysis

Our final design has resulted in a mostly unchanged application from our initial plans. During our development, by utilizing the technologies above we constructed a tightly woven application that supports the large datasets generated by POSYDON. Our frontend UI design changed dramatically from our initial plans. Originally, we had predicted that we would encounter unforeseen issues from working with these large datasets and while we did encounter some issues, we altered our design in such ways that unexpected dataset changes would have minimal impact on our controlled functions such as our data parser. In conclusion, our design consists of four tightly interconnected components that come together to create a fully featured binary star querying application for users of all skill levels.

5. Testing

Given the high dependencies between each component in this project, it was crucial that we performed continuous testing to ensure changes work on their own and together. As a result, our team followed the philosophy that, before any code was accepted, the team member responsible would write automated test cases, as appropriate, that reach complete code coverage. Additionally, the team members performed manual testing and shared the results for other team members to examine. The format of these results depend on the added feature but generally include photos or videos as well as a written memo with the described results of each test. Each addition featured test cases of various types, including unit tests, interface tests, integration tests, system tests, regression tests, and security tests.

5.1 Unit Testing

We perform unit tests on several aspects of our project. As a software-based project, this meant creating test cases for functions used in both our back-end and front-end applications.

Backend:

We created an automated testing pipeline for Python code utilizing a virtual environment generated through Tox. This was run automatically during each commit and was required to pass in order to merge a branch into our main production environment.

Frontend:

Like our backend, we created an automated testing pipeline, but we instead utilized Vitest for TypeScript testing. We developed unit test cases for each potential route to ensure expected results. This utilized Vitest's mocking capability to ensure we only test a single function at a time.

5.2 Interface Testing

Our project had three primary interfaces: the database, the backend, and the frontend.

Database:

The database interface is accessed via SQL queries. To test this, we created a list of potential queries that access various fields in the database. When a team member worked on something that may impact the database, they performed each of these queries and reported any discrepancies.

Backend:

The back-end interface acts as a REST API. As a result, each change was tested by multiple group members performing a set interaction to ensure that any result was the same regardless of minute differences in hardware and software versions. I.e. if a button was added to download a query, multiple group members would ensure that it behaved the same regardless of differences in environment.

Frontend:

The front-end interface is a graphical user interface that users can navigate through. To test this interface, we required team members to write reports about navigating the database. They performed both common actions and edge cases to ensure they worked as expected.

5.3 System Testing

The system tests we performed were done to showcase that all components work together through complete paths. We performed these tests from the positions users interact with, including the front-end application and the back-end application. The primary exception to this rule of expected outcomes was natural language processing. Given the nature of GPT, we could not expect the response to be written the same each time, so we performed these tests manually and verified the correctness of responses ourselves. The paths we tested included, but were not limited to, returning SQL to the frontend from the backend, querying the database, and saving queries.

5.4 Regression Testing

When we made changes, we ensured our changes did not break what we have already created. During each iteration we only made minor changes and automatically ran each of our previous test cases before any production change took place. In the unlikely event that a change made it through without us noticing it breaking something, the team member who found the mistake looked at the commit history on our GitLab repository and notified the responsible party and the rest of the group before reverting the commit.

Though all features of our application are important, we wanted to be particularly careful about several functions. The path for sending a query from the frontend to the database via the backend and retrieving the data in the frontend is particularly important. Without this feature, there was very little that could be done with this project, as the database would still need to be used manually. Additionally, we wanted to make sure our script for populating the database worked as expected. If this ever broke, new users would not be able to set up the database, and, likewise, the application would provide no benefit to the end user.

5.5 Acceptance Testing

To ensure our project met requirements, we continuously tested each functional component in various categories. These categories included, first of all, that each feature works as expected. This meant verifying that the result retrieved and displayed is what one would expect to get at each point in the process. We also tested categories such as performance and visuals to comply with nonfunctional requirements. We then confirmed with our client at our weekly meeting that these elements follow the vision of the project.

5.6 Security Testing

We ensured our project remained secure by utilizing encrypted tunneling methods for our connecting parts such as SSH, HTTPS for OpenAI requests, and encryption of local variables. While we could not directly test functions such as SSH and HTTPS, we did run Wireshark to packet sniff both the HTTPS and SSH traffic to ensure it was encrypted. The local variables were tested by opening the stored file on each new merge and ensuring that they were unreadable.

5.7 Results

We believe that our constructed testing methodology outlined above helped contribute to an effective development environment and ensured that as we continuously developed the application, we did not have any major problems. While the AI responses cannot be guaranteed to be perfect, that is somewhat out of our control as we are limited by the capabilities of OpenAI, but we constrained these issues to the best of our ability.

6. Implementation

This section outlines our implementation of components as well as providing images of our frontend components.

6.1 Frontend

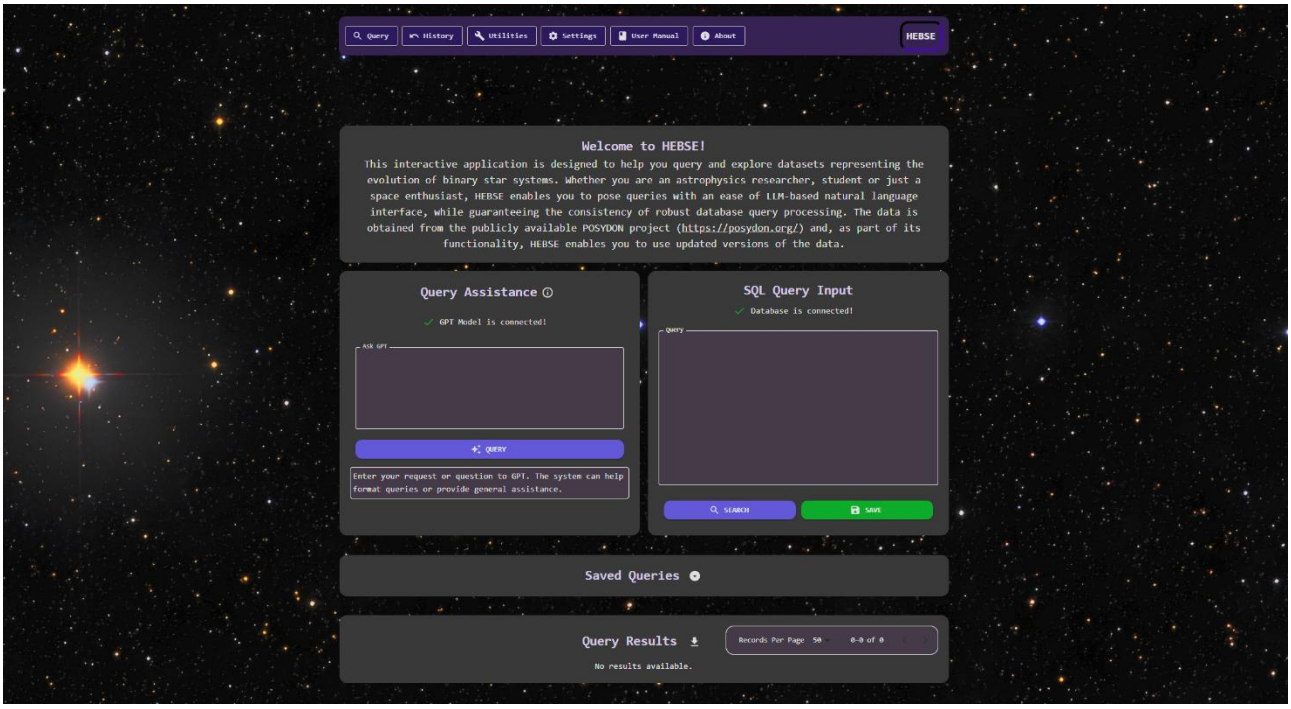

Figure 6.1 – Query Page

- **Query Page**
  - Ability to query database with SQL queries
  - Query viewing, saving and management
  - Query generation using GPT
  - Custom SQL queries
  - Download current query

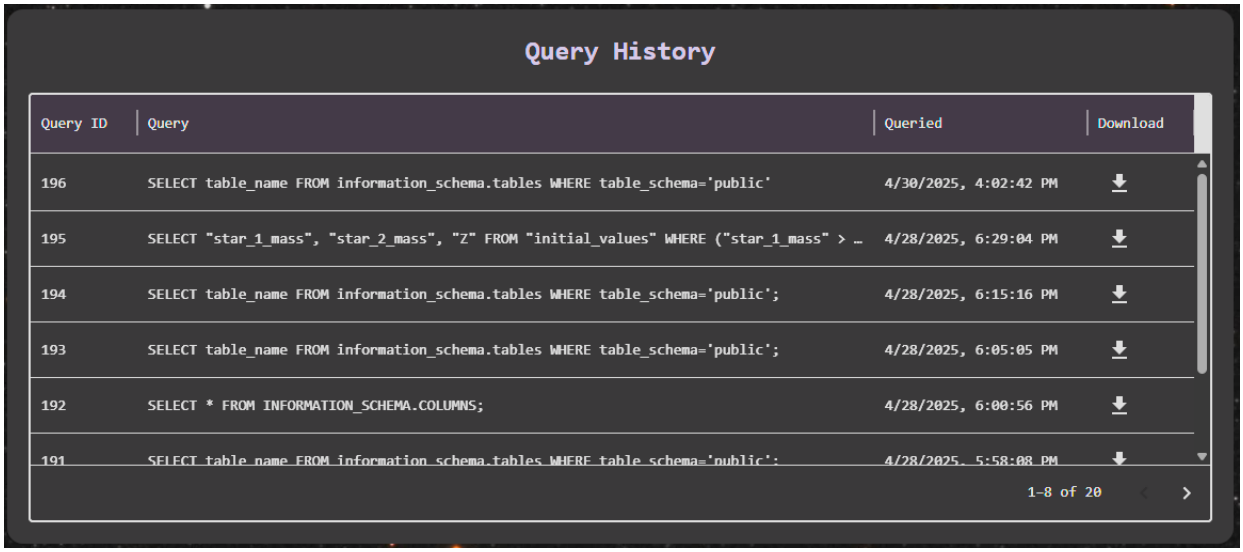

Figure 6.2 - History Page

- **History Page**
  - View and download past query results

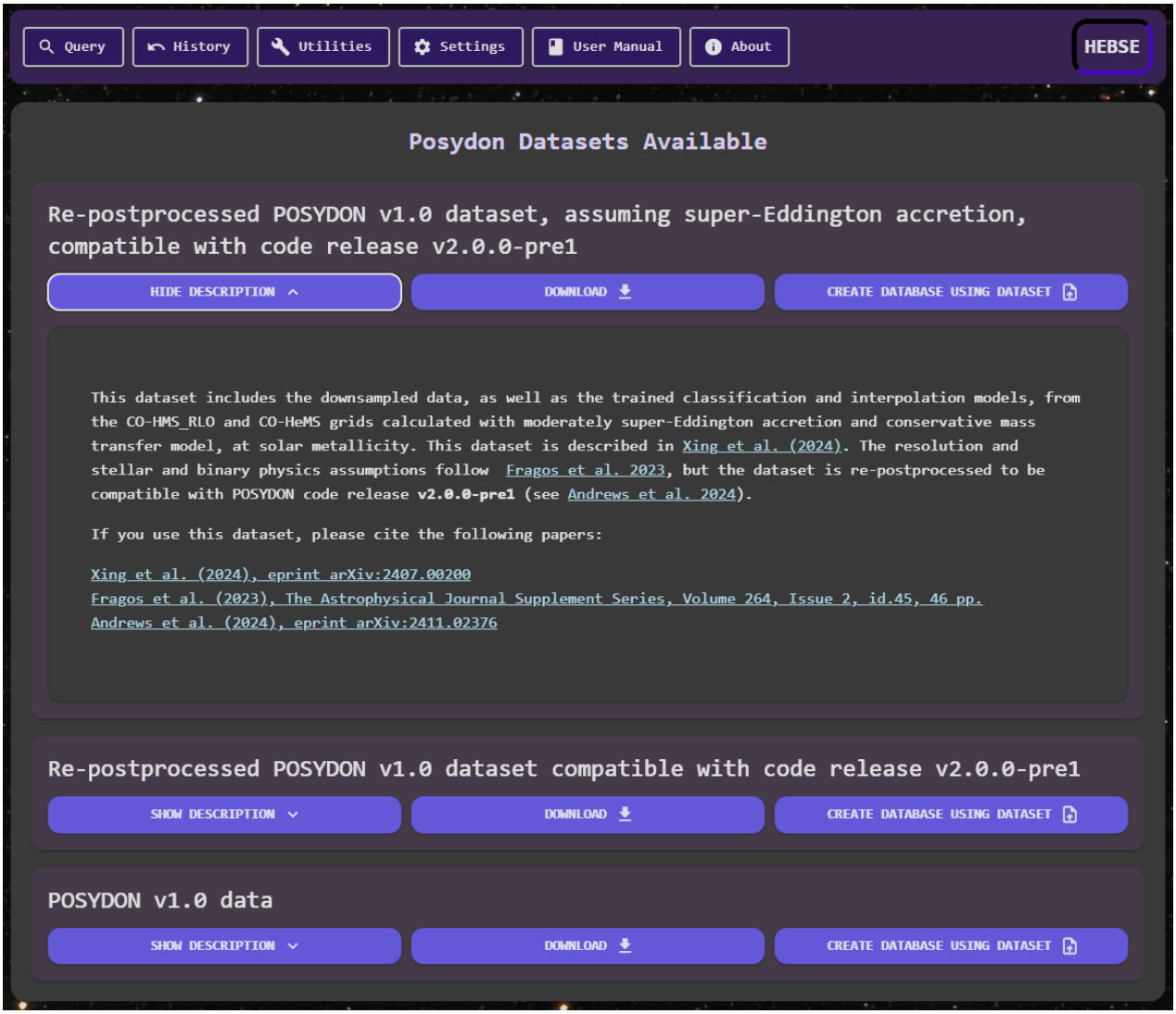

Figure 6.3 - Utilities Page

- **Utilities Page**
  - View and download available dataset versions

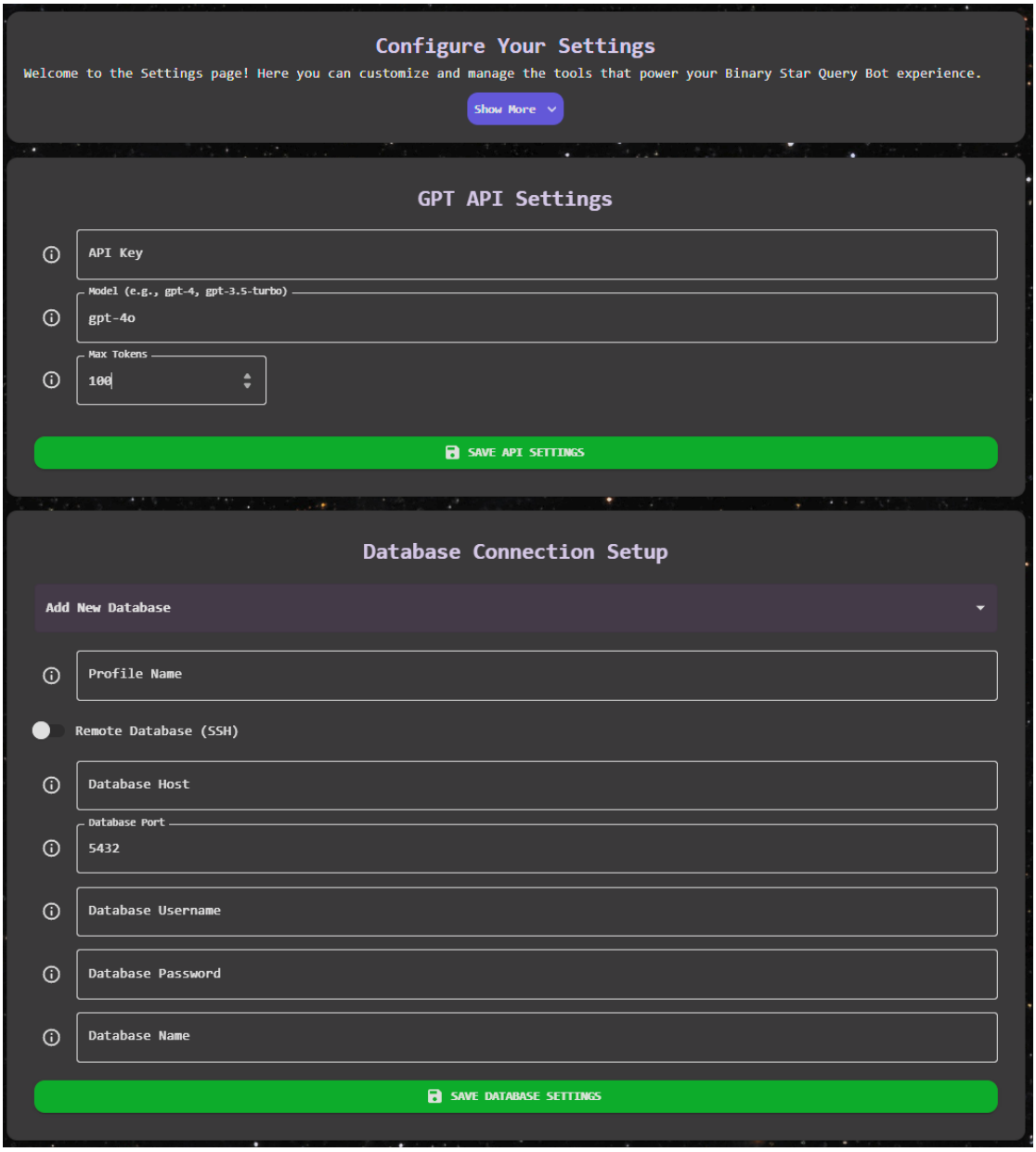

Figure 6.4 - Settings Page

- **Settings Page**
  - Connect to OpenAI
  - Connect to and setup database
    - Remote database with local backend
    - Remote database with remote backend

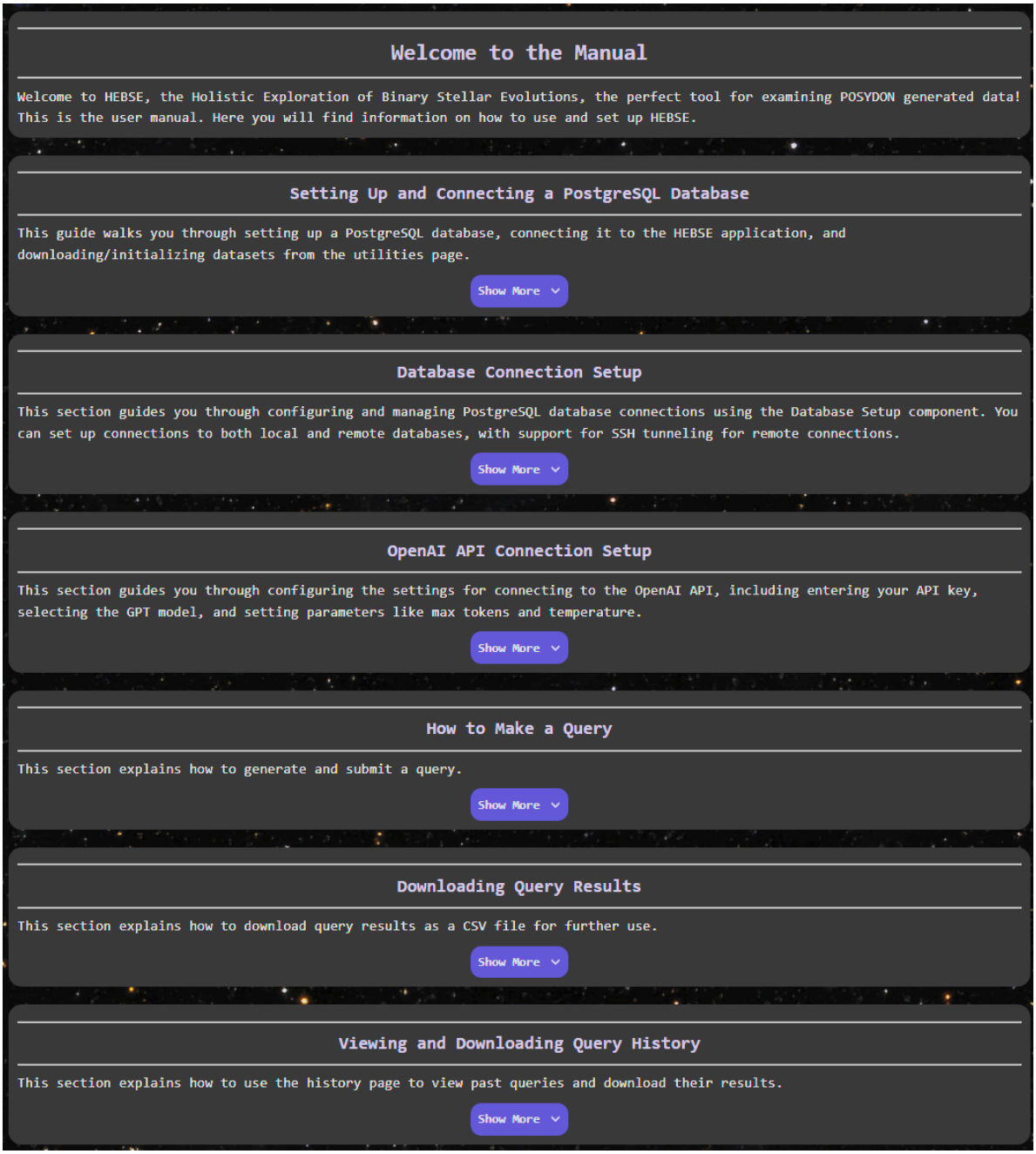

Figure 6.5 - User Manual

- **User Manual**
  - Contains information on how to set-up and use the application
  - Contains several sample queries

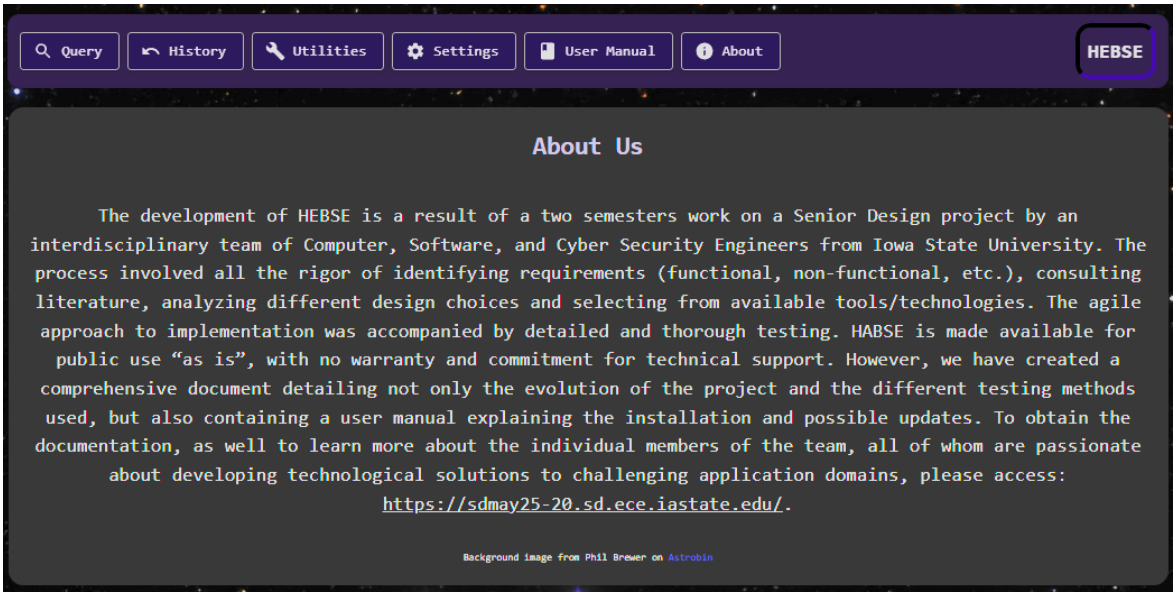

Figure 6.6 - About

- **About**
  - UI with information about team and application

6.2 Backend

- **Frontend-Backend Communications**
  - Implemented back-end APIs in Python utilizing FastAPI to handle requests and responses to the database.
  - Connection via SSH to a remote server or entirely locally
- **Query Execution**
  - Utilizes raw strings to query database
  - User can both query and manage the database
- **NLP Query Execution**
  - Full integration of a GPT model prompted to convert natural language queries into SQL.
- **Environment Setup**
  - Configured back-end development environments and ensured compatibility with the frontend.
- **Error Handling**
  - Errors connecting to GPT model are relayed to the user through the frontend
  - Query errors are displayed inside the query results field for the user
- **Security**
  - Python creates the database through SSH, ensuring encrypted communication
  - All communications between user and OpenAI utilize https protocol
- **Backend and NLP Communication**
  - Developed a script enabling seamless communication between the backend and the OpenAI API.
- **Scalability Improvements**
  - Designing back-end architecture to handle larger datasets as needed.
- **Testing and Validation**
  - Implemented an automated testing pipeline using Tox for continuous integration and complete code coverage. Unit tests simulate external dependencies using unittest.mock to ensure robust testing.
- **Backend Documentation**
  - Documents for the APIs, back-end architecture, and interaction workflows to increase maintainability as features are added.

6.3 Database

- **Database Handling**
  - PostgreSQL handles the database for storing and managing large datasets.
- **Query Support**
  - Supports relational queries and data retrieval from parsed datasets.
- **Initial Database Schema Design**
  - Developed schema to efficiently store and query these large datasets
- **Database Parity**
  - Added functionality to save multiple databases and quickly switch between them
- **Advanced Database Schema Design**
  - Developed a schema that is more adaptable to new and changing datasets.

6.4 Data Parser

- **Initial Data Parsing and Loading**
  - Using Python scripts to preprocess, parse, and clean datasets (CSV/H5 files) before uploading into PostgreSQL.
- **Data Validation**
  - Ensures the data is of the correct format and is readable before attempting to parse and upload it into PostgreSQL.
- **Data Cleaning and Transformation**
  - Converts raw and hexadecimal data to readable formats and fills missing values through interpolation for consistent and complete datasets
- **Additional Parsing Features**
  - Created new features to assist in uploading datasets that may deviate from the normal seen.
- **Monitoring and Reporting**
  - Added features to track parsing progress, flag errors, and generate logs.
- **Parallel Processing**
  - Optimized the parsing process to handle large datasets using parallel processing techniques.
- **Documentation for Data Parser**
  - Created clear guidelines and examples for using data parsing scripts effectively.
- **GUI** Implementation
  - Integrated data parsing scripts into a user-friendly GUI component within the main front-end UI to streamline processes and enhance usability for end-users.

Our implementation plan followed the major sprints outlined in the previous Section 3.3. For visual reference, you may also refer to Figure 3.1 and Figure 3.2.

7. Ethics

This section outlines our approach to engineering ethics and professional responsibility as integral components of our project. To ensure ethical and responsible conduct throughout the project, we implemented specific measures and practices that align with these principles, which are detailed in the following discussion.

7.1 Areas of Professional Responsibility/Code of Ethics

Table 7.1 – Relevant Areas of Professional Responsibility

Area of Responsibility	Definition	Relevant IEEE Code of Ethics	Team Application
Work Competence	Perform high-quality, ethical work within your expertise, avoiding deception.	I.6	Application development was divided among team members based on experience and expertise.
Financial responsibility	Deliver valuable services at fair costs while acting with loyalty and trustworthiness.	I.3, I.6	Application is provided at no cost to the user, and AI assistance is optional.
Communication honesty	Share clear, truthful, and transparent information with all stakeholders.	I.1, I.2 I.3, I.5, I.6	Full disclosure of database versions used and AI-generated query contents.
Health, safety, and well-being	Prioritize and protect the safety, health, and welfare of stakeholders.	I.1, II.9,	N/A
Property ownership	Respect and safeguard the property, ideas, and sensitive information of others.	I.1, I.4	Client data is entirely localized, no information is uploaded to the internet.
Sustainability	Commit to protecting the environment and promoting sustainable practices.	I.1	Demonstrations are done using the lowest feasible power consumption GPT model.
Social responsibility	Develop solutions that benefit society and uphold the profession's integrity.	I.2, III.10	Application assists the astrophysics community with research and can be accessed by anyone for academic/personal use.

One specific area of responsibility in which our team excelled was Work Competence, indicated by our consistently high quality of work over the course of the project. Due to the complexity of our application, involving both front-end and back-end development, and incorporation of emerging NLP technology, we divided development responsibilities among team members based on personal experience and knowledge. Alongside peer and advisor feedback, this boosted our group's efficiency and production of maximum quality work.

An area of responsibility we had planned to improve on was Sustainability. Despite our application being entirely digital with no direct impact on the environment, training LLMs such as GPT 3.0/4.0 takes place at data centers with vast data processing and power consumption requirements. Depending on energy sources, producing the power to train and sustain these AI models can boost carbon emissions and negatively impact the environment. Ultimately, we had to use the highest performance model available for development, because through our testing we discovered that older models did not work well

7.2 Four Principles

Table 7.2 - Four Principles

	Beneficence	Nonmaleficence	Respect for Autonomy	Justice
Public health, safety, and welfare	Design allows all users to perform queries	N/A	Design allows all users to perform queries	Design allows all users to perform queries
Global, cultural, and social	Design allows all users to perform queries	N/A	N/A	N/A
Environmental	N/A	We will demo with low power AI	N/A	N/A
Economic	Software is free and AI is optional	N/A	N/A	N/A

Of particular importance to our project, the context-principle pair “Global, Cultural and Social – Beneficence” is exemplified through our design accounting for differences in user capabilities to generate database queries. Our application employs NLP technology to assist users who are unfamiliar with database querying, while still allowing custom query entry for experienced users who want maximum customization and control. This addresses the needs of multiple user groups and allows for learning opportunities when inexperienced users see their desired outcome translated into proper SQL query format.

An area in which we could stand to improve is "Environmental – Nonmaleficence". As previously mentioned, LLMs require large amounts of power that could negatively impact the environment in the long-term. Therefore, our team worked to minimize the amount NLP required to reduce power consumption of higher-end models.

7.3 Virtues

Demonstrating ethical and virtuous work is crucial for engineers to maintain credibility and ensure solutions meet the needs of consumers. Some virtues our team strived to uphold include:

- **Honesty and Transparency:** Being honest and transparent with consumers means being open and clear in communication and stating facts without embellishment or deceit. Our team demonstrated this in our use of NLP-generated queries from user input. Disclosure of the query that was generated and used to produce results that are presented to the user ensures transparency and prevents miscommunication.
- **Commitment to Quality:** The dedication to consistently meeting high standards in products and services. Through division of work amongst team members based on proficiency and close cooperation and feedback from our advisor, our team strived for excellence and continuous improvement.
- **Inclusivity:** A commitment to fairness and equal opportunity by accommodating all users, regardless of ability. Our application is designed for users of all experience levels and expertise, from professionals in the astrophysics field to educators and students curious about binary star research but unfamiliar with database querying.

Additionally, we believe it is important to demonstrate virtuous merits on an individual level. Our team has selected some qualities that each of us is proud to uphold, and some areas where we could improve:

"One virtue that I strive for in my work is diligence and commitment to quality. When doing my work, I always strive to produce code and work that is both correct and thorough. I strive to create code and work that I can be proud of. I have demonstrated this through my existing work and my efforts to produce a quality product with no defects. One virtue that I have not demonstrated so far in Senior Design is through Documentation. Documentation is something that most engineers struggle with. So far in senior design there has not yet been an effort to produce substantive documentation. It is important to me that we create documentation for this project so that it can be used by real people and continue to be supported."

– Eamon Collins

"One virtue I continue to pursue in my work on this project is that of humility. In a team environment, it's important to be cooperative and have a mindset that is willing to hear and learn from others. I'm always open to constructive feedback and looking for ways I can improve. A quality I could focus on in my future work is altruism. This is a large-scale project with lots of jobs that need to be completed. I should be more proactive in volunteering for tasks, even if it's something I'm not very enthusiastic about, because the success of the project should be my main motivation and not my personal interests."

– James Byrd

"A virtue I strive to uphold is responsibility. In a team-based project where tasks are delegated to individuals and are expected to be completed, upholding work assignments is crucial for success. Even with busy weeks I take the time to ensure my portion of work is completed and my team members are aware of my progress. One virtue I want to improve is orderliness. Orderliness is important to me because it helps ensure that my tasks are completed in a systematic manner. While I often finish all my assignments before the deadline, I would like to complete them even earlier to receive feedback, which can sometimes be challenging with my other classes. In the future, I plan to use a task management app to better track my deadlines."

– Svyatoslav Varnitskyy

"A virtue important to me is adaptability. Being adaptable means adjusting to new and unforeseen situations. During Senior Design, I demonstrated this by learning SQL, React, and how to parse H5 files from scratch. Instead of shying away from these challenges, I took the time to understand and apply these skills, contributing to the team's efforts in development, from frontend design to data retrieval and storage. On the opposite end of the spectrum, one virtue I have struggled to demonstrate is patience. While I understand its importance in the field, I sometimes find it difficult to wait after completing my tasks early. This has led me to start other tasks prematurely, causing unscheduled workflow gaps, some frustration, and reducing overall team cohesion."

–Alek Norris

"I always try to uphold the virtue of cooperation. Through my senior design work, I always make an effort to share my ideas with my team members in a way that all team members can understand. If we have a disagreement in our approaches, I explain my approach while also listening to the ideas of my teammates. In turn, we can come to a decision that all team members are happy with. This is important as coming to a collective decision ensures the best outcome for our end project. At the same time, one important virtue that I have not yet demonstrated in my senior design work is patience. This is important as I believe rushing work leads to an ultimately negative outcome, so giving work the time it needs saves time in the long run. I'll work to demonstrate this in the future by breaking up work into smaller parts and ensuring each part works as expected before moving on to the next."

– Andrew Snyder

"A virtue I strive to uphold is cooperation. Throughout my senior design work much of my focus is on ensuring that the team runs smoothly and that everyone is in an environment that is both fun to be in and conducive to the creation of good work. Cooperation is important to me because I want everyone to have a good time and produce a great product. I've demonstrated this by keeping the environment light while helping facilitate everyone's voices and opinions are heard. A virtue I need to improve upon is my civic-mindedness. This virtue goes hand in hand with my application of cooperation stated earlier. When trying to keep the environment fun and lighthearted, I, on occasion, get carried away and become overbearing. It is important to me to work on this so that I can keep that fun environment while maintaining an excellent level of professionalism. To work on this in the future, I will continue to slow down and take each moment in smaller bites, with more thought before my actions and words."

- Alex Polston

7.4 Changes

During the development of our project almost all of the above outlined ethical decisions did not change, this is because the nature of our project is purely a computer-based SQL querying system and most decisions towards power and security are made by the user. There was a slight alteration to our projected decision towards power consumption of the AI model we used. During our development, through testing, we discovered that the much lower powered versions of GPT would almost always return malformed or incorrect queries. This caused us to shift to utilizing the lowest powered version that would consistently return a proper query, but users may choose to use their preferred model.

8. Closing Material

We will now discuss our closing material.

8.1 Conclusion

Over the semester, we finalized our design for our application. Our development time has culminated in a finished product that fulfills our set-out requirements and exceeds others. Our application has a user-friendly interface and scheme and performs query translation from natural language successfully when needed. Our application fulfills and exceeds our set goals from our Fall Conclusion in that, it hosts a binary star database on a network machine connected through SSH, with support for either a local or remote backend. It features a robust utilities page for easy implementation of a POSYDON dataset powered database and the ability to recall previous queries, and to download them for offline use and sharing.

It fulfills and surpasses the goals outlined in our Fall Conclusion by hosting a binary star database on a networked machine connected via SSH, with full support for either a local or remote backend. The system offers researchers the flexibility to operate fully locally, in a hybrid mode (local backend with remote database), or fully remote, depending on their preferences and resource needs. All user local storage is secured using AES-GCM 256-bit encryption to ensure data confidentiality. Furthermore, researchers are empowered to change and update their GPT models as needed, allowing them to customize their workflow without being tied to a single static configuration.

8.2 Value Provided

Our application serves a very unique purpose, and it is designed to host the POSYDON data into a queryable format, but it could realistically be used for any dataset. Our application addresses the needs of the users of the generated data by providing an easy way to access it.

8.3 Next Steps

For a future group who might work on a project similar to this, or as a continuation to this one the additions of adding a page to select and view whole unfiltered tables, and options to formulate queries utilizing templates, buttons, and dropdown menus would add much improved functionality, especially for those less familiar with creating SQL queries.

9. References

[1] “IEEE SA - IEEE/ISO/IEC 12207-2017,” *SA Main Site*. https://standards.ieee.org/ieee/12207/5672/

[2] “IEEE Standards Association,” *IEEE Standards Association*, 2020. https://standards.ieee.org/ieee/24748-3/7102/ (accessed Dec. 06, 2024).

[3] “ISO/IEC 9075-1:2023,” *ISO*. https://www.iso.org/standard/76583.html

[4] IEEE, “IEEE Code of Ethics,” *ieee.org*, Jun. 2020. https://www.ieee.org/about/corporate/governance/p7-8.html

[5] Sophie Mclean, “The Environmental Impact of ChatGPT: A Call for Sustainable Practices In AI Development,” *Earth.org*, Apr. 28, 2023. https://earth.org/environmental-impact-chatgpt/

[6] Posydon.org, 2024. https://posydon.org

[7] “Albireo - AstroBin,” *AstroBin*. https://app.astrobin.com/i/jfo1on

[8] “Binary Star Database.” http://bdb-old.inasan.ru/

10. Appendices

10.1 Definitions

HEBSE – Holistic Exploration of Binary Stellar Evolutions
POSYDON – POpulation SYnthesis with Detailed binary-evolution simulatiONs
NLP – Natural Language Processing
RDBMS - Relational DataBase Management System
CSV - Comma Separated Value
SQL - Structured Query Language
UI – User Interface
GUI – Graphical User Interface
LLM – Large Language Model

10.2 User Manual

As a note, the User Manual is also included inside the application with dropdown menus and a more user-friendly reading experience

Welcome to HEBSE, the Holistic Exploration of Binary Stellar Evolutions, the perfect tool for examining POSYDON generated data! This is the user manual. Here you will find information on how to use and set up HEBSE.

Setting Up and Connecting a PostgreSQL Database:

This guide walks you through setting up a PostgreSQL database, connecting it to the HEBSE application, and downloading/initializing datasets from the utilities page.

Setting Up a PostgreSQL Database

To set up a PostgreSQL database on your server or local device:
- Visit the official PostgreSQL website to download the installer: https://www.postgresql.org/download/
- Select your operating system and follow the provided installation instructions.
- Once installed, open the PostgreSQL (PSQL) command line or a GUI tool (like pgAdmin) to create a new database.
- Take note of and write down the connection details: host (e.g., 'localhost' for local setups), port (default is 5432), username, password, and the database name you created.

Connecting HEBSE to the PostgreSQL Instance

To connect HEBSE to your PostgreSQL instance:
- Open the HEBSE application and go to the 'Settings' section using the navigation bar.
- Locate the 'Database Connection Setup' area.

- Recall the connection details from the database setup.
- Enter the following details:
  - Host: The server address (e.g., 'localhost').
  - Port: Typically 5432 unless changed during setup.
  - Username: Your PostgreSQL username.
  - Password: Your PostgreSQL password.
  - Database: The name of the database you created.
- Click 'Save' to establish the connection.

Downloading and Initializing the Database

To download and initialize datasets using the utilities page:
- Please ensure that you have already created and connected to the desired database before proceeding with this step.
- Navigate to the 'Utilities' page in HEBSE.
- You'll see a list of publicly available POSYDON datasets. For each dataset, you can:
  - Click 'SHOW DESCRIPTION' to reveal the dataset's details (click 'HIDE DESCRIPTION' to collapse it).
  - Click 'DOWNLOAD' to save the dataset file to your device (a tooltip shows the file name and size in GB).
  - Click 'CREATE DATABASE USING DATASET' to automatically download the dataset and populate your connected PostgreSQL database.
- The 'CREATE DATABASE USING DATASET' option triggers a process that uses the dataset file to initialize the database, making it ready for use.

Important Notes

- Ensure PostgreSQL is running before attempting to connect HEBSE.
- Double-check your connection details to avoid errors.
- Dataset files can be large (sizes are shown in the tooltip); ensure you have enough storage and a reliable internet connection.
- The 'CREATE DATABASE USING DATASET' feature requires a valid database connection and may take time depending on the dataset size.

Database Connection Setup

This section guides you through configuring and managing PostgreSQL database connections using the Database Setup component. You can set up connections to both local and remote databases, with support for SSH tunneling for remote connections.

Selecting a Database

- Dropdown Menu: Use the dropdown to select an existing database configuration or choose

"Add New Database" to create a new one.
- Existing Database: Selecting an existing database loads its saved configuration into the form fields.
- New Database: Choosing "Add New Database" clears the form fields for a fresh configuration.

Configuring Database Connection

Fill in these fields to set up your database connection:
- Host: Enter the hostname or IP address of the PostgreSQL server (e.g., localhost or 192.168.1.1).
- Port: Enter the port number for the PostgreSQL server (default is 5432).
- Username: Enter the database username.
- Password: Enter the database password. Use the eye icon to toggle visibility.
- Database Name: Enter a unique name for the database you want to connect to.

Remote Database Connection

For databases hosted remotely with SSH tunneling:
- Remote Switch: Toggle the "Remote Database" switch to enable remote settings.
- SSH Fields: Complete these additional fields:
- SSH Host: Enter the SSH host for tunneling (e.g., example.com).
- SSH Port: Enter the SSH port (default is 22).
- SSH Username: Enter your SSH username on the remote host.
- SSH Password/Private Key: Paste your SSH private key or password (ensure it’s correctly formatted).

Saving and Removing Configurations

- Save Settings: Click "Save Database Settings" to store the configuration. It updates an existing database or adds a new one to the list.
- Remove Database:
If an existing database is selected, a "Remove Database" button appears. Click it to delete the selected configuration.

Important Notes

- Unique Database Name: Ensure the "Database Name" is unique to avoid conflicts.
- SSH Key Format: For remote connections, verify that your SSH private key is correctly formatted and matches the SSH user and host.
- Local Storage: Configurations are saved in your browser’s local storage.

OpenAI API Connection Setup

This section guides you through configuring the settings for connecting to the OpenAI API, including entering your API key, selecting the GPT model, and setting parameters like max tokens and temperature.

Entering the API Key

Use the 'API Key' field to enter your OpenAI API key, which is necessary for authenticating requests to the OpenAI API. The eye icon allows you to toggle the visibility of the key for added security.

Selecting the GPT Model

In the 'Model' field, enter the name of the GPT model you want to use (e.g., 'gpt-4' or 'gpt-3.5-turbo'). For a complete list of available models, refer to the OpenAI API documentation at https://platform.openai.com/docs/models.

Setting Max Tokens

Adjust the following parameters to fine-tune the GPT model's behavior:

Saving the Settings

Once you have entered your desired settings, click the 'Save API Settings' button to store them. The settings are saved in your browser's local storage, ensuring they persist across sessions.

Important Notes

- Keep your API key secure and do not share it publicly.
- Ensure the model name is correctly entered as specified in the OpenAI documentation.
- Experiment with different max tokens and temperature values to find the optimal settings for your use case.
- All configurations are stored locally in your browser and are not transmitted to any server.

How to Make a Query

This section explains how to generate and submit a query.

Using AI Query Assistance

On the query page locate the "Query Assistance" box. Type your prompt into the "Ask GPT" field. Press the "Query" button. The AI assisted SQL query will show up below. For best results use prompts that are formatted as a question.

Submitting SQL Query

To submit an SQL query, either copy an AI assisted SQL query from the "Query Assistance" box or write your own SQL query and enter the SQL query into the "SQL Query Input" box. Press "Save" to save the query for future use. Press "Search" to submit the Query. The result of the Query will appear below in the "Query Results" box.

Viewing Query Results

To view query results after submitting a query, look at the "Query Results" box on the "Query" page.

Downloading Query Results

This section explains how to download query results as a CSV file for further use.

Locating the Download Button

In the 'Query Results' section, find the download button with a download icon (↓) next to the 'Query Results' title.

Downloading the Results

Click the download button to fetch the complete query results from the server. The results are saved as 'query_results.csv' and automatically downloaded to your device.

File Format

The file is in CSV format, compatible with spreadsheet software like Microsoft Excel or Google Sheets.

Viewing and Downloading Query History

This section explains how to use the history page to view past queries and download their results.

Accessing the History Page

Go to the 'History' section, denoted by a backward arrow, to see a table listing your past queries, including their ID, SQL query, and timestamp.

Viewing Past Queries

The table includes columns for 'Query ID', 'Query' (the SQL statement), 'Queried' (timestamp), and 'Download'. Use the pagination controls at the bottom to browse through multiple pages of queries if available.

Downloading Query Results

Each row features a download button with a downward arrow icon (↓). Click this button to retrieve the query results from the server and save them as a CSV file named 'query_[ID]_results.csv', where [ID] is the Query ID.

Troubleshooting

If you encounter issues:

- Verify the server is active at 'http://localhost:8000'.
- Look in the browser console for error messages if the download fails.
- Ensure a stable internet connection.

10.3 Initial version of Design

The following sections outline our original design and thoughts during the beginning of the development of the project. Some of the below outlined features and plans have been changed during development and the finalized versions are included in prior sections.

10.3.1 Overview

Our project consisted of four main subsystems. The user interface (UI), utilizing the React library for TypeScript, visualizes the application for the user. It allows the user to make natural language or SQL queries, displays queried data in an easy-to-understand format, and enables users to save their queries for later use.

The UI is connected to the back-end application. This system is responsible for communication between the UI and database as well as the mode of communication with OpenAI for handling natural language processing. When a user sends natural language to the backend, it sends this text alongside the database's blueprint to OpenAI. In response, OpenAI returns a properly structured SQL query.

Before returning the query to the user, the backend determines if the query follows proper SQL syntax and retrieves data from the database. If the backend finds the query not valid, it makes a second attempt with OpenAI. If this still fails, it will return a message to the UI requesting the user to rewrite their message for clarity. If the query is deemed valid, however, it will be returned to the UI. At this point, the user can choose to accept or modify the query before returning it to the backend. The backend will then query the database.

The database is a relational database managed by PostgreSQL. The database manages the data and returns the desired data upon retrieving a query from the backend. At this point, the backend sends this retrieved data to the user interface to be displayed. The final subsystem is the data parser. The data parser is a script that retrieves the CSV files generated by POSYDON and parses them. This parsed data is then stored in the database.

10.3.2 Detailed Design

User Interface (*UI*): Built using React and TypeScript, this layer provides users with interactive elements and visuals. It captures the user's input (e.g., SQL or NLP queries) and displays the query results, assisting in facilitating an intuitive end-user experience.

Backend: Manages requests from the UI, relays them to OpenAI for NLP processing, validates SQL requests and responses, and communicates with the PostgreSQL Database while also supporting the parsing and initialization of the database. The backend acts as the bridge between the UI, database, and the AI model.

Database (*PostgreSQL*): The database system stores the data from the parsed and normalized CSV files and enables the SQL querying of the data.

Data Parser: Built using Python, the Data Parser Interprets the raw CSV files from POSYDON and normalizes them to make meaningful SQL queries possible. Reads the data from the CSV files and applies normalization methods to allow comparisons more readily between different simulations. Enter the data into the Database.

10.3.3 Component Breakdowns

User Interface (*UI*) with React:

Role: Provides the primary user interaction point, enabling users to input SQL queries, view results, save, import database, and configure personal settings such as the ability to use OPENAI for NLP.

Operations: How React enables this, how it allows/displays the data dynamically, and works to provide this role and functionality.

Backend (*Python*):

Role: Serves as the central data processing layer, routing commands, validating queries, formatting responses, and the primary means of interconnecting components.

Operations: Connects the UI, database, and OpenAI, handling all data routing. Depending on whether the user is using natural language or raw SQL, the backend either directly processes the SQL query or uses OpenAI to generate a structured query. It validates all queries to ensure security and accuracy. Validated queries are executed, and the results are temporarily saved for the session before being sent back to the UI for display.

Database (*PostgreSQL*):

Role: Central data repository, where parsed CSV data from POSYDON is organized

Operations: Structured to handle relational queries, the database supports complex data types, allowing efficient data retrieval. The data relationships are designed to reflect the structure within the original CVS that was used to build the database.

Data Parser (*Python*):

Role: Parses Raw POSYDON data, normalizes it, and enters it into the database

Operations: The parser, integrated with the backend, reads and interprets CSV files while standardizing data types. During this process, it performs initial data cleaning, including normalization, to ensure a uniform structure across entries. If gaps in the data are found, the parser estimates missing values by averaging similar local data points. This step is especially crucial given the massive datasets generated by POSYDON, which uses machine learning to simulate binary star systems. The parser ensures data integrity and accuracy, preparing all datasets for efficient storage and querying within PostgreSQL tables.

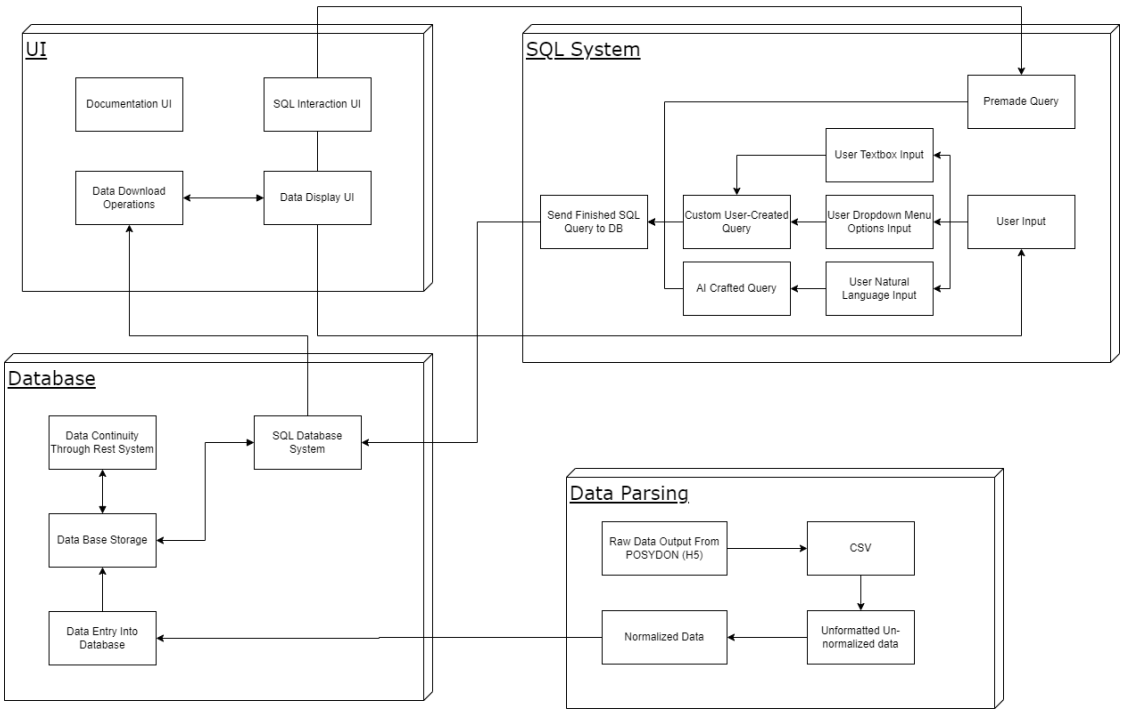

Figure 10.1 – Initial Visual Representation

10.3.4 Functionality

In a typical use case of our application, an end user would use a local server or a high-spec personal computer to run a set-up script to host the database. The end-user would then run the desktop application used to parse the database, and running this, the application would also build the database and connect it to the application. After a successful connection, the end user would be able to perform two types of queries: a query using their own constructed SQL statement or an SQL statement generated through a machine learning-powered natural language processor, which would translate their request to a properly formatted SQL query. This query would then be performed, and a small subset of the data would be returned to the home screen of the application for preview. The whole result would then be available for download to a file on the end user's machine. The user could then perform another query or close the application.

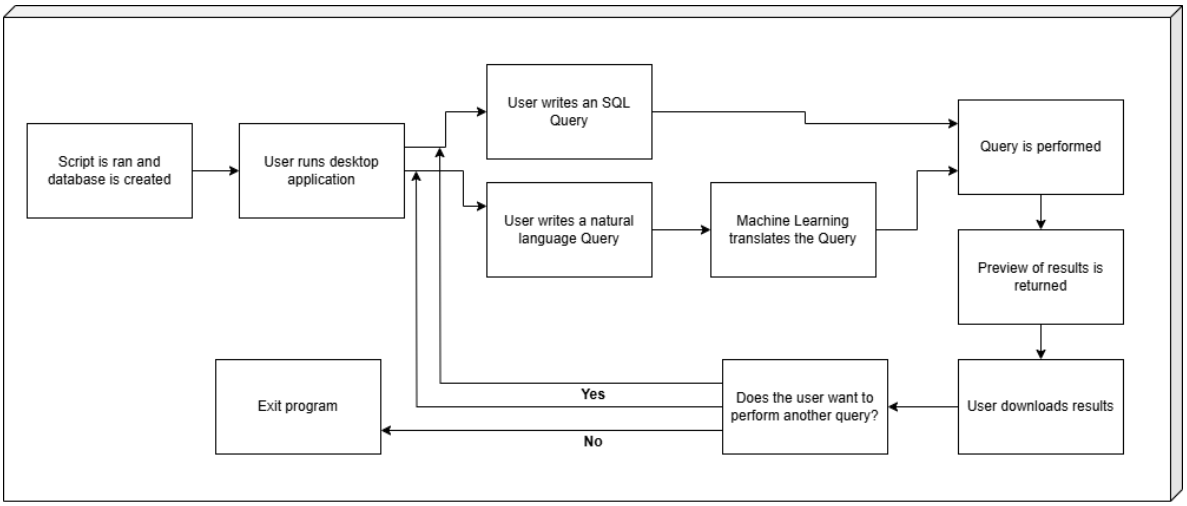

Figure 10.2 – Initial Task Flow

10.3.5 Areas of Concern and Development

Going into this project, we determined our requirements and user needs. At its core, our requirements are to build a database that stores the data generated by POSYDON simulations and to allow users to easily retrieve data from this database. Beyond this, we noted that non-technical users would make up a large portion of our user base. As a result, we created several additional requirements to support these users. These users need to be able to set up the database without necessarily knowing how to maintain a database, and they need to have the option of querying the database without necessarily having a detailed knowledge of SQL and its syntax.

Our project fulfills these requirements through each of its major components. To solve our initial requirements, we chose to store the data in a PostgreSQL database. As a result of its relational nature, SQL is easy to develop queries for, ensuring the database consistently retrieves all desired data when queried. This is handled even further through our front-end user interface. Through this, users are able to enter queries without interfacing with the database directly. Similarly, this user interface supports our requirement of allowing non-technical users to query the database. By integrating our user interface with OpenAI via a textbox for natural language, users can find relevant information from the database without writing SQL queries. Furthermore, these users can initially set up the database through our database generation script. This ensures users have no need to understand relational databases to set this up.

Given this design, our primary area of concern is our reliance on an LLM. Though GPT is improving over time, it does not always follow prompts, and it is possible for users to trick it. As a result, we opted to develop software for our back-end application that verifies that queries performed are properly structured so that they adhere to our database schema. This will be done through a two-stage process in which our back-end application performs initial checks to confirm the validity of the query. If the application finds no issues with the query, the backend will utilize the PostgreSQL database management system to confirm the query is valid. We believe that this will substantially mitigate the risk posed to the project using an LLM.

10.3.6 Initial Implementation

Figure 10.3 - Wireframe

This was the initial proof-of concept implementation of our Query Page. Below are listed all of the requirements, work done during our first semester and features we planned upon.

10.3.6.1 Frontend

- **Query Page**
  - UI
  - Query viewing, saving and management
  - Ability to query database with SQL query
  - Natural language processing integration
  - Query editing after natural language processing
  - Custom SQL queries

- **History Page**
  - View and download past query results
- **Utilities Page**
  - Database connection tool
  - Data ingestion tool
- **Settings Page**
- **About Us**
  - UI with information about team

10.3.6.2 Backend

- **Frontend-Backend Communications**
  - Implementing back-end APIs in Python utilizing FastAPI to handle requests and responses to the database.
- **Query Execution**
  - Initial implementation of SQL query execution on the backend.
- **Environment Setup**
  - Configuring back-end development environments and ensuring compatibility with the frontend.
- **Initial Error Handling**
  - Basic error handling for malformed SQL queries or connectivity issues.
- **Security**
  - Preliminary setup for secure communication between the frontend and backend.
- **Backend and NLP Communication**
  - Developed a script enabling seamless communication between the backend and the GPT API.
- **NLP Query Execution**
  - Full integration of a GPT model trained to convert natural language queries into SQL.
- **Advanced Error Handling**
  - Enhancing error handling with detailed error messages for users.
- **Scalability Improvements**
  - Designing back-end architecture to handle larger datasets as needed.
- **Testing and Validation**
  - Implementing an automated testing pipeline using Tox for continuous integration and complete code coverage. Unit tests will simulate external dependencies using unittest.mock to ensure robust testing.
- **Backend Documentation**
  - Documents for the APIs, back-end architecture, and interaction workflows to increase maintainability as features are added.

- **Basic Security Measures**
    - Explore implementing basic encryption for data in transit and minimal authentication mechanisms (such as token-based or API key-based authentication) to future-proof the system if accounts or access controls are added later.

10.3.6.3 Database

- **Database Handling**
    - Using PostgreSQL as the primary database for storing and managing large datasets.
- **Query Support**
    - Supporting relational queries and data retrieval from parsed datasets.
- **Database Schema Design**
    - Developing schema to efficiently store and query these large datasets
- **Database Optimization**
    - Fine-tunning PostgreSQL performance for handling terabytes of data efficiently.
- **Database Backup and Recovery**
    - Provide robust backup and recovery options with support for multiple databases and tables, ensuring data integrity and resilience against failures.

10.3.7.4 Data Parser

- **Initial Data Parsing and Loading**
    - Using Python scripts to preprocess, parse, and clean datasets (CSV/H5 files) before uploading into PostgreSQL.
- **Data Validation**
    - Ensures the data is of the correct format and is readable before attempting to parse and upload it into PostgreSQL.
- **Data Cleaning and Transformation**
    - Converts raw and hexadecimal data to readable formats and fills missing values through interpolation for consistent and complete datasets
- **Additional Parsing Features**
    - Creating new features to assist in uploading datasets that may deviate from the normal seen.
- **Monitoring and Reporting**
    - Adding features to track parsing progress, flag errors, and generate logs.
- **Parallel Processing**
    - Optimizing the parsing process to handle large datasets using parallel processing techniques.

- **Documentation for Data Parser**
    - Create clear guidelines and examples for using data parsing scripts effectively.
- **GUI Implementation**
    - Integrating data parsing scripts into a user-friendly GUI component within the main front-end UI to streamline processes and enhance usability for end-users.

10.4 Other Considerations

We were unable to obtain the latest dataset from POSYDON and because this program was designed and built in under a year there may be unforeseen bugs and issues encountered by an end user. If another team of developers were to add functionality or obtain the latest dataset additional testing would be required to ensure usability. At its core this project is a GUI environment that provides simplicity to build and use a dataset for less technically inclined users

10.5 Fall Conclusion

Over the semester, we created a design for our database hosting and querying application. As well as preliminary designs and outlined considerations for the use of natural language processing through machine learning. This has culminated in a prototype utilizing a smaller dataset and pre-constructed queries. We have demonstrated the basic functionalities of our design and proved that many of the technological integrations needed for the project have been successful. Moving forward with the learning and skills developed from our prototype, we will be able to produce our designed application.

Our overarching goals for this project are to deliver a functional application, meeting the enumerated requirements, that allows the hosting of a local binary star SQL database alongside an application that allows a user to query their database, with optional support for AI-powered natural language processing to convert sentences into SQL queries.

To achieve our overarching goals for this project, we will continue to develop our prototype, and we will design a PostgreSQL database that will intake queries written by the user or translated through NLP and subsequently parsed with a Python backend. Alongside this, we will design a React-and-Typescript-based front-end application to allow user interfacing with the database.

10.6 Code

To access the project and its related code, please visit our Gitlab or Github:
https://git.ece.iastate.edu/sd/sdmay25-20
https://github.com/hebse-project-org/hebse-application

10.7 Team Contract

Team Members:

1) Alexander Polston 2) Eamon Collins

3) Alek Norris 4) James Byrd

5) Svyatoslav Varnitskyy 6) Andrew Snyder

Team Procedures:

1. Day, time, and location (face-to-face or virtual) for regular team meetings:

   We will meet face-to-face Wednesday 1:15-2:15 and Friday at 2:00.

2. Preferred method of communication updates, reminders, issues, and scheduling:

   Discord and Text messages are used as needed

3. Decision-making policy:

   Consensus in most cases, with a majority vote on split options where there are few options.

4. Procedures for record keeping:

   Everyone takes notes, and they are compiled together after each meeting.

Participation Expectations:

1. Expected individual attendance, punctuality, and participation at all team meetings:

   Individual attendance – Group members are expected to attend all meetings when available. Attendance is excused with reasonable prior notification. Group meeting times can be flexible week to week with group agreement

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

   Members are expected to complete work that is assigned to them by the deadlines laid out, if problems arise it is expected that the individual will reach out and ask for help as well as notifying all members of the group.

3. Expected level of communication with other team members:

   During weekdays responses are expected within a reasonable amount of time between the hours 10am – 10pm.

4. Expected level of commitment to team decisions and tasks:

Members are expected to do their work to the best of their ability and to respect decisions made by other group members. Members should contribute equally to decisions and tasks, while maintaining a respectful environment.

Leadership:

1. Leadership roles for each team member:

   Client interaction - Eamon

   Team morale & Coordination – Alex

   Front-End Lead – Svyatoslav

   Back-End Lead – Andrew

   Development and design are managed by the individual responsible for that specific component. To ensure compatibility between components individuals are expected to leave useful comments and communicate with members that are designing components that work in continuity with their own design.

2. Strategies for supporting and guiding the work of all team members:

   Communicate during and outside of weekly meetings.

3. Strategies for recognizing the contributions of all team members:

   The Team will discuss members' contributions and provide feedback on completed work. Compliments and recognition are key.

Collaboration and Inclusion:

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

   Andrew Snyder – Access control, SQL Queries

   Eamon Collins – Back-end database development

   Alek Norris – Front-end development, Network Packet analysis

   James Byrd – Back-end database management

   Svyatoslav Varnitskyy – Front-end development, SQL Queries

   Alex Polston – Identity management, Front-end Development

2. Strategies for encouraging and supporting contributions and ideas from all team members:

   Practicing active listening when members are presenting ideas and contributions.

3. Procedures for identifying and resolving collaboration or inclusion issues

Our group will hold “open room” style of meetings where everyone is free to speak their mind, where no-one is above another and if there are inter-member issues they will be addressed as a group.

Goal-Setting, Planning, and Execution:

1. Team goals for this semester:

   Work actively as a team and successfully work on developing a program to query dynamic datasets stored in CSV files.

2. Strategies for planning and assigning individual and team work:

   Work will be on a “claim” basis, where we will identify all tasks that need to be performed, and they will be “claimed” based on interest and availability.

3. Strategies for keeping on task:

   Weekly meeting with progress checks and clearly outlined deadlines for deliverables.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

   First serious infraction – Team discussion about performance and expectations

2. What will your team do if the infractions continue?

   Repeated serious infractions – discussions with faculty in addition to team.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Alex Polston September 17, 2024

2) Andrew Snyder September 17, 2024

3) Eamon Collins September 17, 2024

4) James Byrd September 17, 2024

5) Alek Norris September 17, 2024

6) Svyatoslav Varnitskyy September 17, 2024