Module Guide for Software Engineering

Team #18, Gouda Engineers
Aidan Goodyer
Jeremy Orr
Leo Vugert
Nathan Perry
Tim Pokanai

November 15, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Reference Material

This section records information for easy reference.

2.1 Abbreviations and Acronyms

Symbol / Abbreviation	Description
AC	Anticipated Change
API	Application Programming Interface
APP	Appearance Requirement
CAP	Capacity Requirement
DPP	Data Processing Pipeline (Module)
DSS	Data Schema and Storage (Module)
DV	Data Visualization (Module)
EOU	Ease of Use Requirement
EPE	Expected Physical Environment Requirement
FEM	Fault and Error Management (Module)
FR	Functional Requirement
FRDR	Federated Research Data Repository
FUNC	Functional Requirement (Prefix)
HH	Hardware-Hiding (Module)
HTTPS	Hypertext Transfer Protocol Secure
IMM	Immunity Requirement
INT	Integrity Requirement
IWAS	Interfacing with Adjacent Systems Requirement
JSON	JavaScript Object Notation
LEA	Learning Requirement
LEG	Legal Requirement
LON	Longevity Requirement
LLM	Large Language Model
M	Module
MAI	Maintenance Requirement
MG	Module Guide
NLP	Natural Language Processing
NLPQP	NLP Query Processor (Module)

Symbol / Abbreviation	Description
NFR	Non-Functional Requirement
OCD	Obsessive-Compulsive Disorder
OI	Open Issue
OS	Operating System
POA	Precision or Accuracy Requirement
PROD	Productization Requirement
R	Requirement
REL	Release Requirement
REST	Representational State Transfer
ROFT	Robustness or Fault Tolerance Requirement
SAL	Speed and Latency Requirement
SOE	Scalability or Extensibility Requirement
SRS	Software Requirements Specification
SUP	Supportability Requirement
UAP	Understandability and Politeness Requirement
UC	Unlikely Change
UD	User Documentation Requirement
UI	Front-End Interface (Module)
WE	Wider Environment Requirement
WR	Waiting Room (Deferred Requirement)
AAC	Authentication and Access Control (Module)
ADA	Adaptability Requirement
ACC	Accessibility Requirement
JSON API	Structured data exchange format over HTTP(S)

Contents

List of Tables

List of Figures

3 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (Parnas et al., 1984). We advocate a decomposition based on the principle of information hiding (Parnas, 1972). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by Parnas et al. (1984), as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (Parnas et al., 1984). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility, and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section 4 lists the anticipated and unlikely changes of the software requirements. Section 5 summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section 7 gives a detailed description of the modules. Section 8 includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section 9 describes the use relation between modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The layout, accessibility, and responsiveness of the front-end interface.

AC2: Fine-tuning or using a different query NLP Model.

AC3: The constraints of filter and query metadata field inputs.

AC4: The framework used for creating visualizations from data and presenting them.

AC5: The algorithms implemented for behavioural analysis or classification.

AC6: The virtual hosting location of Dr. Szechtman's dataset.

AC7: The implementation of the user authentication and access control mechanism for scalability.

AC8: The implementation of error detection and fault recovery mechanisms.

4.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: The read-only access to Dr. Szechtman's dataset.

UC2: The dataset hosted on FRDR being our primary and individual data source.

UC3: Scope of data processing analysis (Does not generalize rat behaviour trial data to other animal models or domains).

UC4: The software application's MVC design pattern.

UC5: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC6: User's computing device (The assumption that the user's device will support modern browsers with Javascript enabled, will not change)

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Hardware-Hiding Module

M2: NLP Query Processor Module

M3: Front-End Interface Module

M4: API Layer Module

M5: Data Schema and Storage Module

M6: Data Visualization Module

M7: Data Processing Pipeline Module

M8: Authentication and Access Control Module

M9: Fault and Error Management Module

6 Connection Between Requirements and Design

This section summarizes how the system design satisfies the requirements defined in the SRS. The traceability matrices in section 8 provide the detailed mappings. We highlight the main design rationale.

The functional requirements are met through the coordinated behaviour of the M3, M4, M2, M5, and M7 modules, which together support querying, dataset retrieval, processing, and visualization. Non-functional requirements related to usability, appearance, and accessibility are addressed primarily by the M3 and M2 modules, which enforce a simple interface and natural-language interaction.

Performance, reliability, and scalability needs are handled by backend modules such as M4, M5, M7, and M9, while security and maintainability are supported by M8, and M9. These design choices isolate complexity and align with information-hiding principles.

Anticipated change requirements map to the modules identified in Table 4, demonstrating that the system decomposition localizes change and supports future extensibility.

Level 1	Level 2
Hardware-Hiding Module	
Behaviour-Hiding Module	Front-End Interface Module API Layer Module Data Schema and Storage Module Data Visualization Module
Software Decision Module	NLP Query Processor Module Data Processing Pipeline Module Authentication and Access Control Module Fault and Error Management Module

Table 2: Module Hierarchy

7 Module Decomposition

The system is decomposed according to principles of information hiding (Parnas et al., 1984). Each module is defined by its *Secrets* (hidden design decisions), *Services* (exposed functionality), and the software or technology used for implementation. Only leaf modules in the hierarchy (section 5) are directly implemented.

7.1 Hardware-Hiding Module (M1)

Secrets: Configuration of computing infrastructure including backend servers, databases, and frontend deployment environments.

Services: Provides physical and virtual infrastructure for computation, storage, and web services.

Implemented By: OS / Cloud Infrastructure

7.2 Behaviour-Hiding Modules

7.2.1 Front-End Interface Module (M3)

Secrets: Frontend state management, routing, and layout design.

Services: User interface for dataset search, visualization, filtering, and download.

Implemented By: React

7.2.2 API Layer Module (M4)

Secrets: API routing, endpoint definition, query optimization.

Services: Provides REST API for structured and natural language queries.

Implemented By: FastAPI, Python

7.2.3 NLP Query Processor Module (M2)

Secrets: Query interpretation logic and integration with language model APIs.

Services: Converts natural language queries into structured database commands.

Implemented By: Custom Python code

7.2.4 Authentication and Access Control Module (M8)

Secrets: User authentication and role-based access control.

Services: Manages secure access to data and APIs.

Implemented By: FastAPI Security Layer

7.3 Software Decision Modules

7.3.1 Data Processing Pipeline Module (M7)

Secrets: Data processing algorithms and pipeline configuration for efficiency and scalability.

Services: Processes behavioral datasets to compute metrics and prepare data for visualization.

Implemented By: Python (NumPy, Pandas)

7.3.2 Data Schema and Storage Module (M5)

Secrets: Database schema, indexing strategies, and query optimization.

Services: Efficient storage and retrieval of behavioral datasets and metadata.

Implemented By: PostgreSQL

7.3.3 Data Visualization Module (M6)

Secrets: Visualization design and interactive plotting logic.

Services: Generates charts, trajectories, and heatmaps for research analysis.

Implemented By: React + Charting Libraries

7.3.4 Fault and Error Management Module (M9)

Secrets: Error detection, logging, and recovery mechanisms.

Services: Ensures robustness and reliability during runtime exceptions or unexpected input.

Implemented By: Python logging / monitoring frameworks

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Table 3: Trace Between Requirements and Modules

Requirements	Modules
FUNC.R.1	M3, M5, M7
FUNC.R.2	M2, M4, M5
FUNC.R.3	M3, M5, M7
FUNC.R.4	M5, M6, M7
FUNC.R.5	M3, M4, M6
FUNC.R.6	M3, M4, M5
FUNC.R.7	M1, M3, M4
APP.R.1	M3, M7
APP.R.2	M3
APP.R.3	M3, M7
EOU.R.1	M3, M2
EOU.R.2	M2, M3, M6
LEA.R.1	M2, M3
LEA.R.2	M3, M6
UAP.R.1	M2, M3, M4
UAP.R.2	M3
ACC.R.1	M3
SAL.R.1	M7
SAL.R.2	M4
POA.R.1	M5, M7
POA.R.2	M4, M5, M7, M9

Continued on next page

Table 3 (continued)

Requirements	Modules
ROFT.R.1	M3, M9
ROFT.R.2	M9
CAP.R.1	M4, M5
CAP.R.2	M1, M5
SOE.R.1	M4
SOE.R.2	M4, M7
LON.R.1	M9
LON.R.2	M4
EPE.R.1	M1, M4
EPE.R.2	M1, M4
EPE.R.3	M1
WE.R.1	M1, M3, M4
IWAS.R.1	M4, M5
PROD.R.1	No Module
PROD.R.2	No Module
REL.R.1	No Module
REL.R.2	No Module
MAI.R.1	M9
MAI.R.2	M4, M5
SUP.R.1	M3
SUP.R.2	M3
ADA.R.1	M1, M4
ADA.R.2	M1, M3, M4
ADA.R.2	M4
INT.R.1	M8, M9
IMM.R.1	M8
IMM.R.2	M4, M8
LEG.R.1	No Module
STA.R.1	No Module
STA.R.2	No Module
WR.1	M6, M7
WR.2	M5, M7

Continued on next page

Table 3 (continued)

Requirements	Modules
WR.3	M7

AC	Modules
AC1	M3
AC2	M2
AC3	M4, M5
AC4	M6
AC5	M7
AC6	M1, M5
AC7	M8
AC8	M9

Table 4: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. Parnas (1978) said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

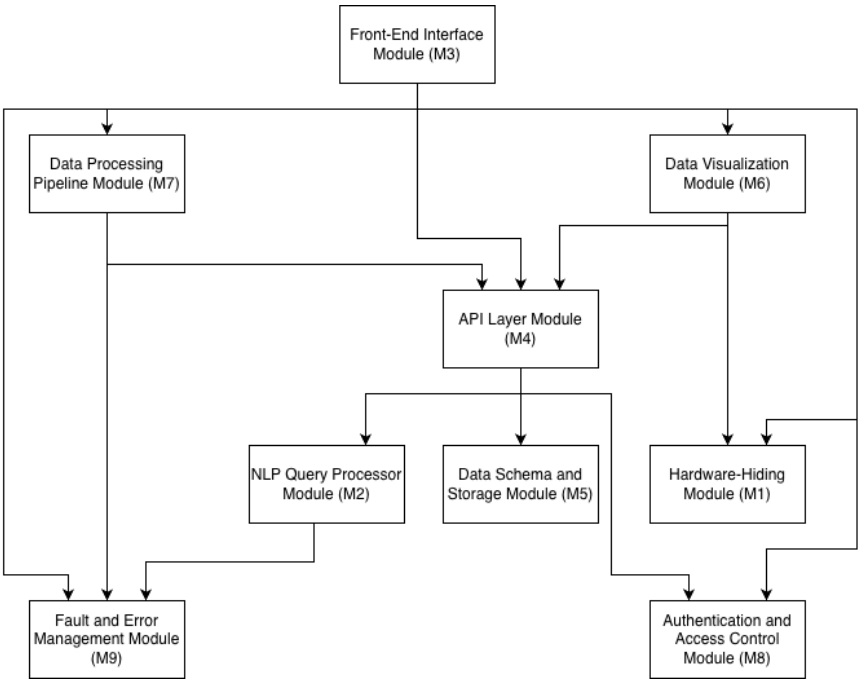

Figure 1: Use hierarchy among modules

10 User Interfaces

This section presents the main user interface designs for **RatBat2**, it shows the structure and layout of the system's main pages. The following figures show a revised user interface design of the application, developed to guide implementation and maintain a consistent user experience across the program.

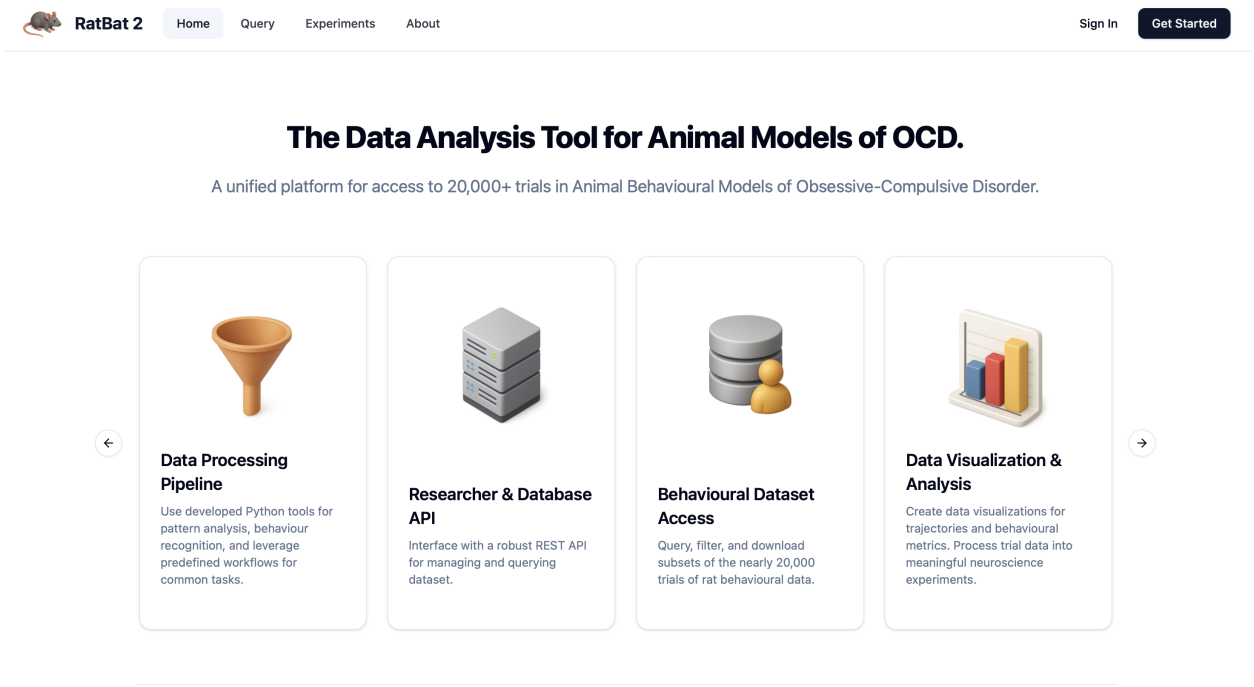

Figure 2: RatBat2 Homepage – Initial Layout Frame 1

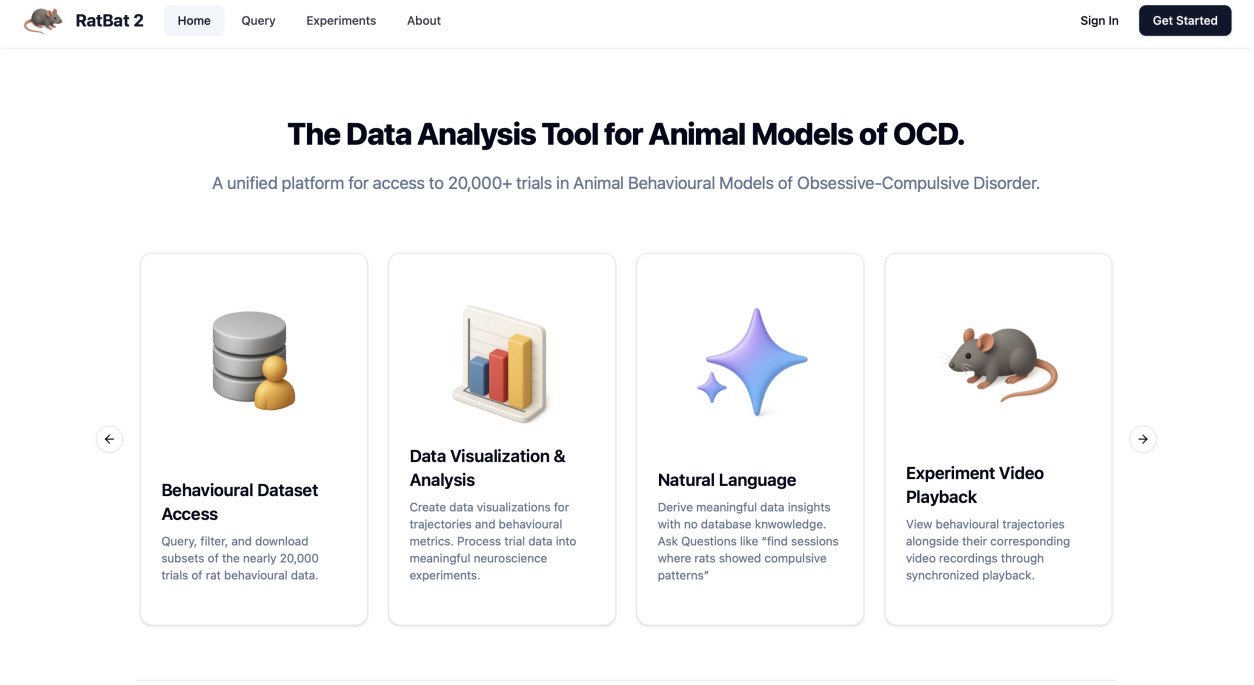

Figure 3: RatBat 2 Homepage – Initial Layout Frame 2

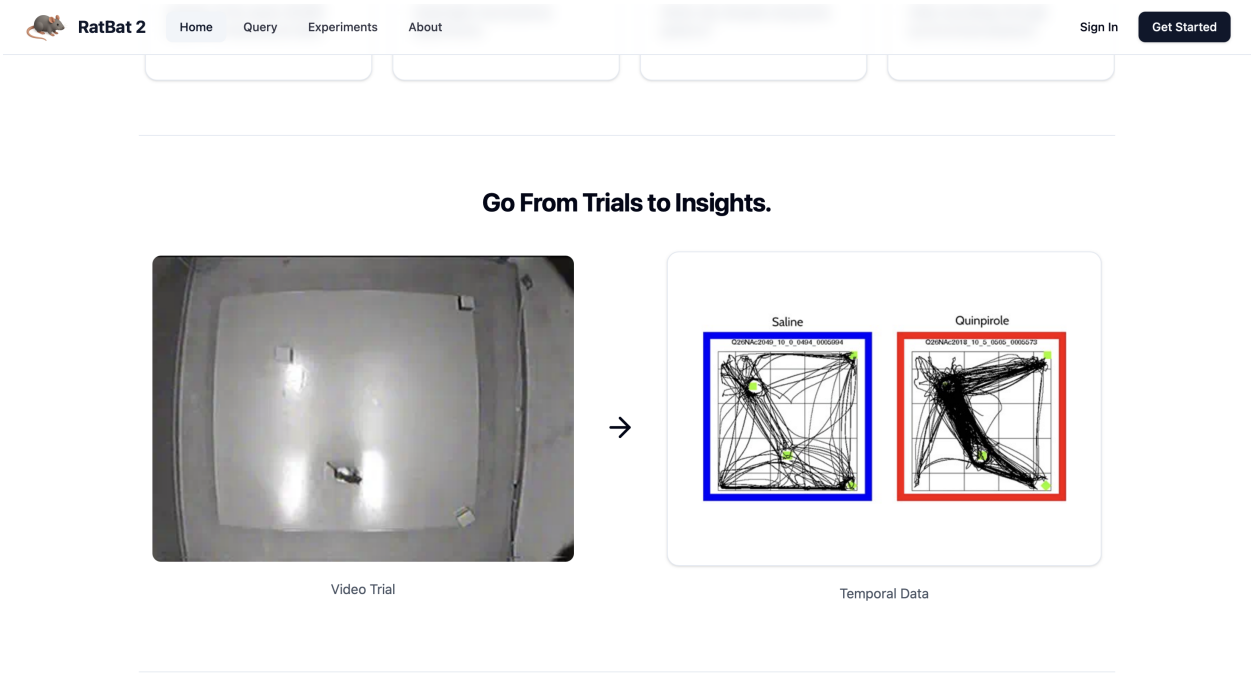

Figure 4: RatBat2 Homepage – Trial Data Display

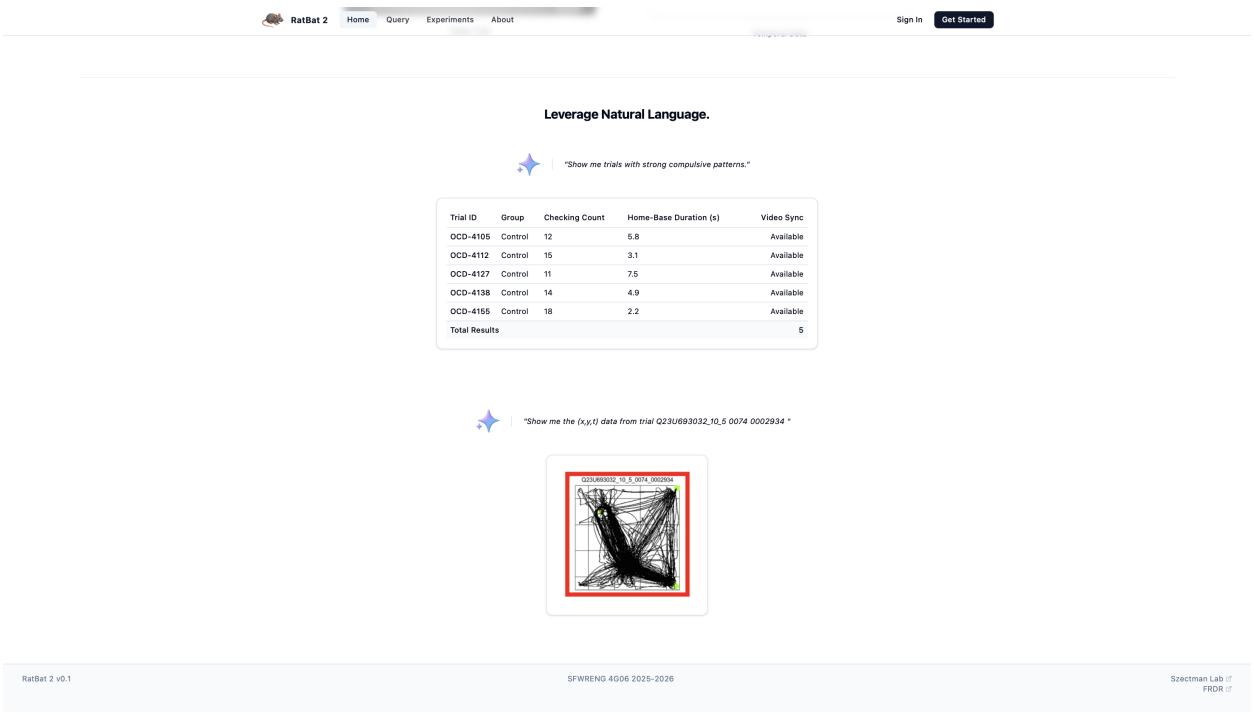

Figure 5: RatBat2 Homepage – NLP Information

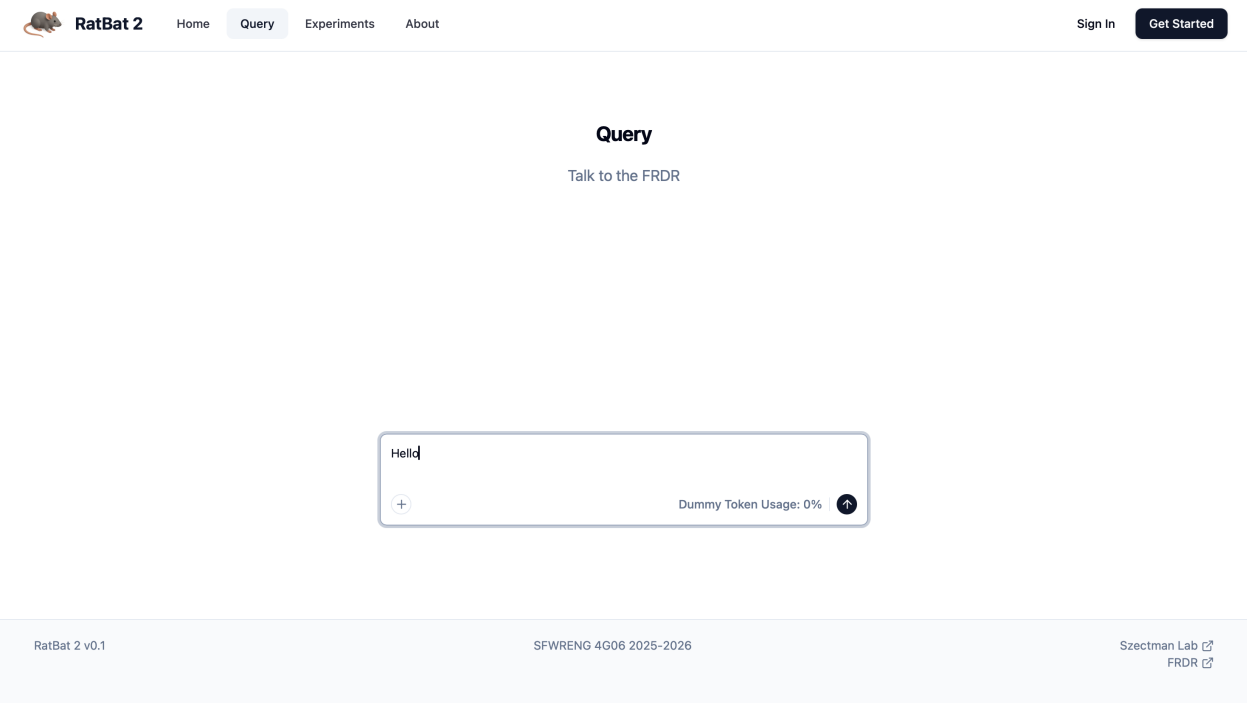

Figure 6: Query Page – Search and Filter Interface

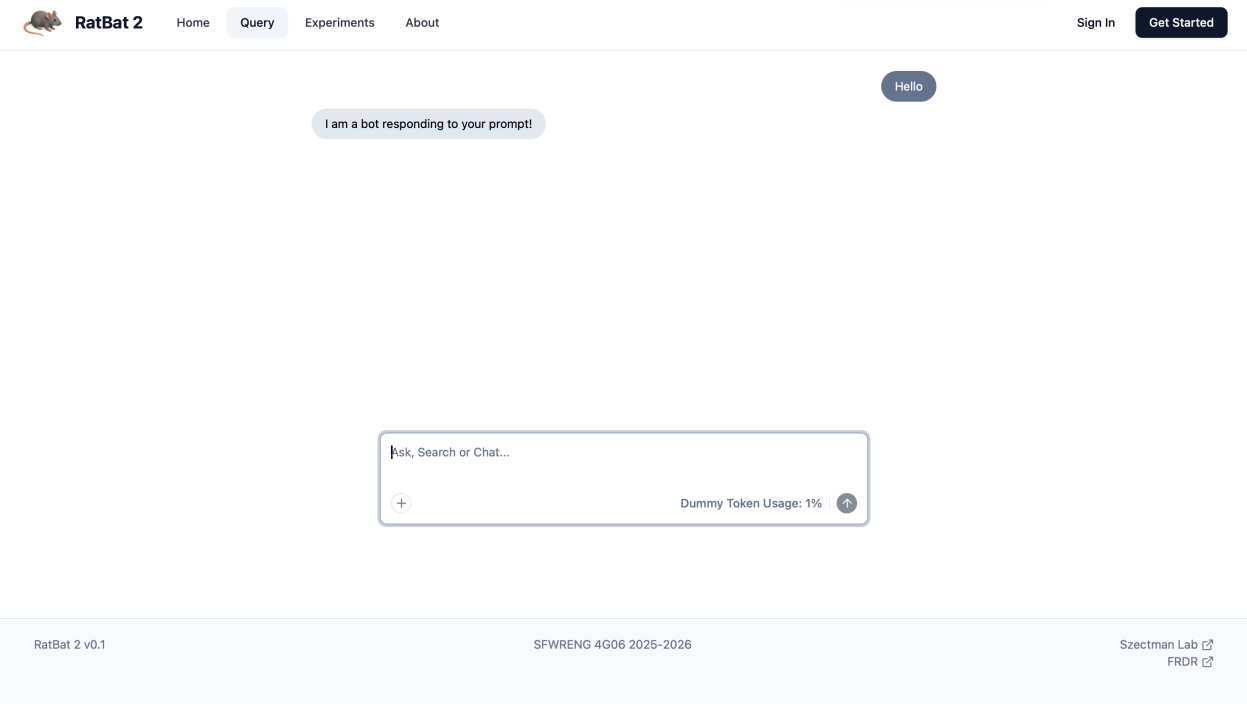

Figure 7: Query Page – Test of Sample Prompt

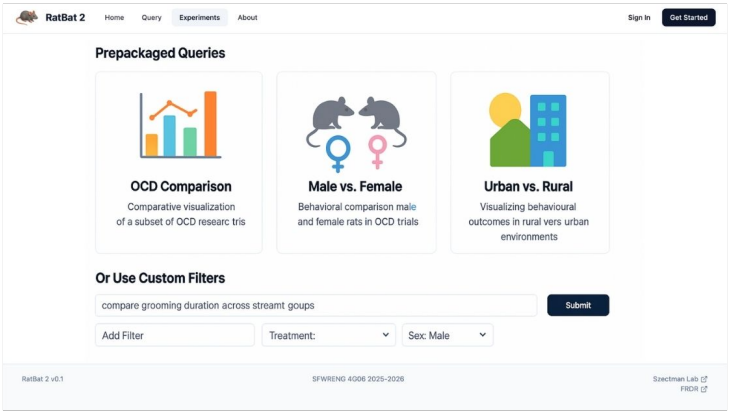

Figure 8: Experiment Page – Experiment Management and Overview

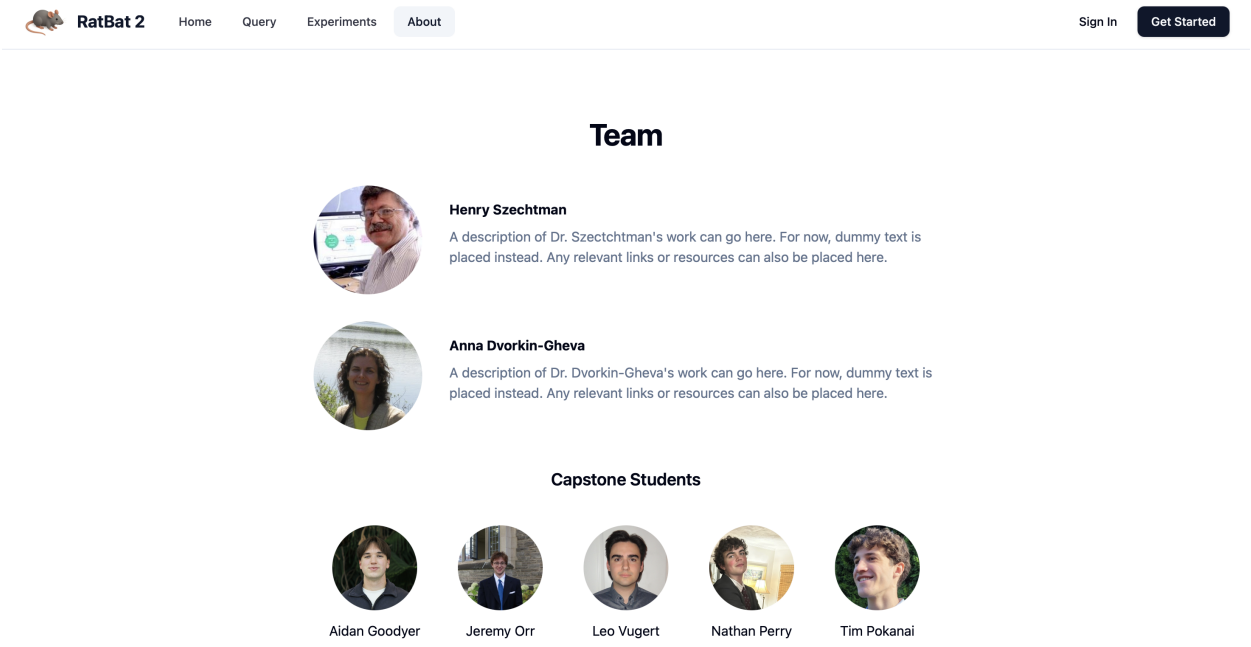

Figure 9: About Page – Team Section

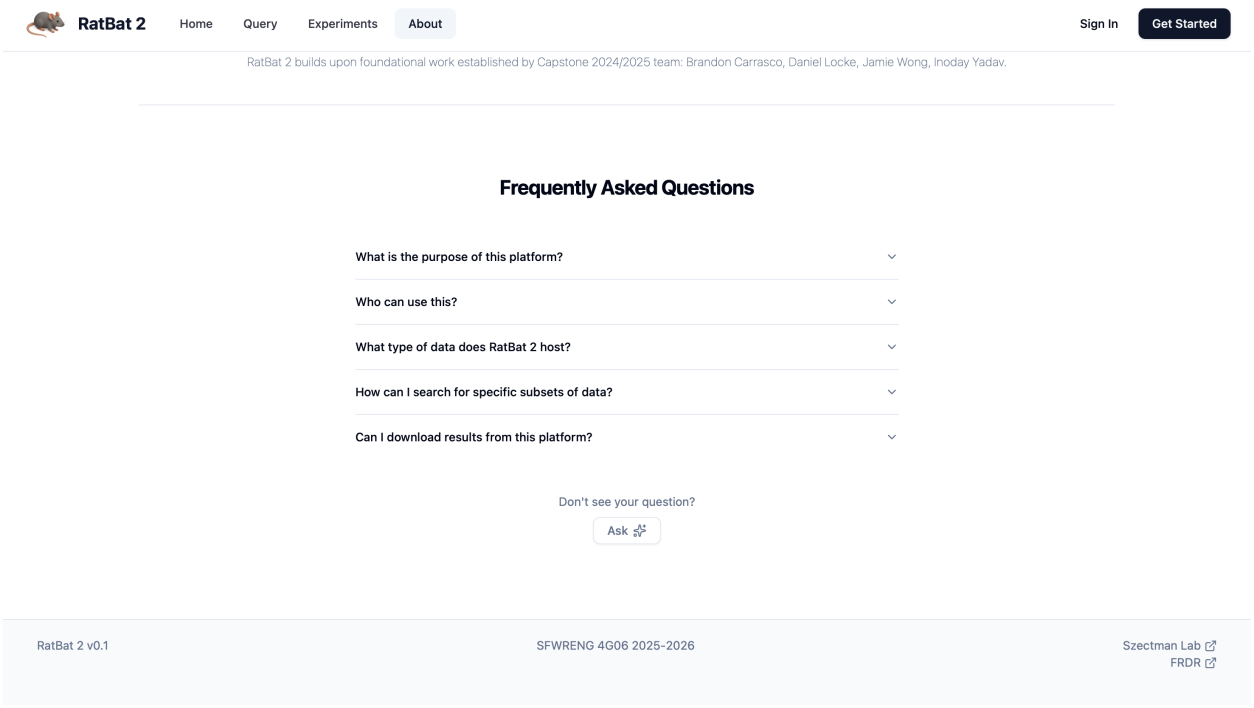

Figure 10: About Page – Frequently Asked Questions (FAQ) and Background Information

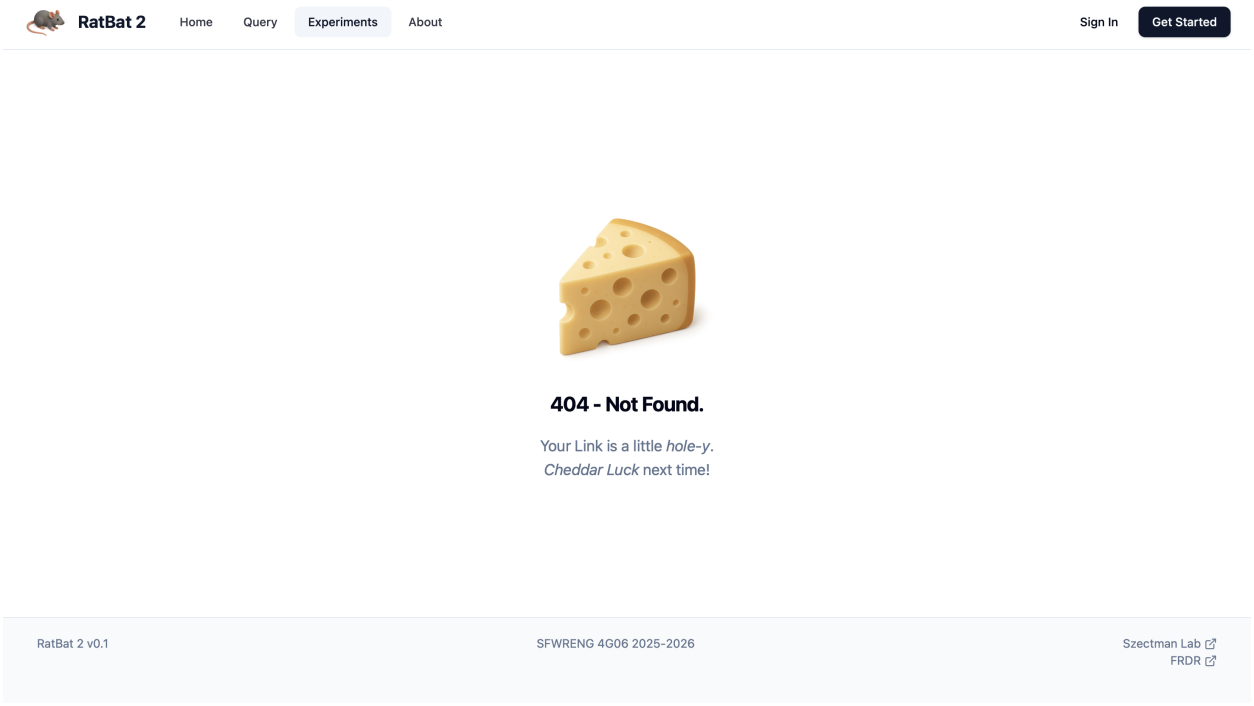

Figure 11: Error Page – 404 Not Found Screen

11 Design of Communication Protocols

The communication protocols define how system modules exchange information and coordinate data flow between the user interface, processing components, and data sources. The goal is to maintain consistent, secure, and modular interactions across all subsystems.

11.1 Overall Communication Architecture

All communication follows a client–server model where the Front-End Interface Module (M3) acts as the client and the API Layer Module (M4) serves as the central hub between client requests and backend modules. Data is exchanged in JSON format over HTTPS to ensure interoperability and security.

11.2 External Communication

The front-end sends RESTful HTTP(S) requests to the API layer for querying behavioural datasets, submitting NLP-based searches, and requesting visualizations. Responses are returned as JSON objects containing results, metadata, or error information.

11.3 Internal Communication

The API Layer (M4) intermediates between modules:

- Sends parsed queries to the NLP Query Processor (M2).
- Retrieves structured data from the Data Schema and Storage Module (M5).
- Forwards subsets to the Data Processing Pipeline (M7) for analysis.
- Provides outputs to the Data Visualization Module (M6).
- Coordinates with Authentication and Access Control (M8) for validation.
- Works with Fault and Error Management (M9) for monitoring and recovery.

11.4 Communication Reliability and Standards

All communication uses HTTPS/TLS encryption. Backend interactions are stateless, enabling independent module deployment. Standard HTTP response codes are used for consistency (e.g., 200, 400, 401, 500). Asynchronous queues handle long-running tasks to prevent blocking.

11.5 Scalability and Fault Tolerance

Modules support horizontal scaling through containerization. The Fault and Error Management Module (M9) provides retry and fallback mechanisms. The Hardware-Hiding Module (M1) abstracts hosting details, allowing migration across environments without affecting communication.

12 Timeline

The project timeline outlines the major milestones, deliverables, and responsibilities for each development phase. Progress, task assignments, and discussions are tracked publicly through the project's GitHub repository at:

https://github.com/OCD-Rats-Capstone/OCD-Rat-Infrastructure/issues

This repository serves as the primary project management platform, where issues correspond to individual tasks or features, each labeled with priority, category, and assignee.

Table 5 summarizes the high-level schedule of the project, organized by development phase and major deliverables.

Ongoing updates, issue tracking, and future work will continue to be maintained through GitHub to ensure transparency and collaboration with supervisors and contributors.

References

David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

David L. Parnas. Designing software for ease of extension and contraction. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 264–277, Piscataway, NJ, USA, 1978. IEEE Press. ISBN none.

D.L. Parnas, P.C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

Table 5: Project Timeline and Responsibilities

Phase	**Major Tasks / Deliverables**	**Responsible Members**	**Duration**
Week 1–2	Project setup, environment configuration, and repository initialization. Define database schema draft.	Backend + Database Teams	2 weeks
Week 3–5	Develop and test PostgreSQL schema. Implement initial API endpoints for data ingestion and retrieval.	Backend Team	3 weeks
Week 6–8	Implement frontend React interface for search and filtering. Integrate with backend API endpoints.	Frontend Team	3 weeks
Week 9–10	Implement data visualization and synchronized video playback components.	Frontend + Visualization Teams	2 weeks
Week 11–12	Integrate NLP query functionality and improve search capabilities.	NLP + Backend Teams	2 weeks
Week 13–14	Conduct system testing, debugging, and performance optimization. Prepare user documentation.	All Members	2 weeks
Week 15	Final deployment, demonstration, and report submission.	Entire Team	1 week