Convoys in Atomic and Molecular Trajectories

Final Report

Team 19

Advisers: Goce Trajcevski

Team Members: Ayden Albertsen, Benjamin Hall, TJ Thielen, Ben McClannahan, Benjamin Riemersma

Team Email: sdmay24-19@iastate.edu

Team Website: https://sdmay24-19.sd.ece.iastate.edu/

Executive Summary

Development Standards & Practices Used

IEEE/ISO/IEC 12207-2017 [1]

- Covers the common framework for the software development life cycle.

IEEE/ISO/IEC 90003-2018 [2]

- supply, development, operation and maintenance of computer software and related support services.

ISO/IEC 27001 [3]

- Standard dealing with information security and practices that should be followed.

V. Phoha, "A standard for software documentation," in Computer, vol. 30, no. 10, pp. 97-98, Oct. 1997, doi: 10.1109/2.625327. [4]

Summary of Requirements

- front-end UI that will enable:
  - (i) users to select dataset;
  - (ii) users to enter parameters;
  - (iii) selection of algorithms;
  - (iv) presentation of the results to the end-user;
- back-end that will store the molecular simulation datasets;
- "middleware" that will connect the front-end and back-end;

Applicable Courses from Iowa State University Curriculum

S E 309 - Software Development Practices
COM S 363 - Introduction to Database Management Systems
COM S 327 - Advanced Programming Techniques
COM S 311 - Introduction to the Design and Analysis of Algorithms
COS S 319 - Construction of User Interfaces

New Skills/Knowledge acquired that was not taught in courses

- Background knowledge of flocks and convoys
- Inner workings of a convoy detection and related algorithms
- 3D Graphing
- Data Visualization
- Chemical Process Simulators

Table of Contents

Table of Figures & Tables

1 Team

1.1 Team Members

- Ayden Albertsen
- Benjamin Hall
- TJ Thielen
- Benjamin Riemersma
- Ben McClannahan

1.2 Required Skill Sets

- Frontend development – Design and implement the UI
- Backend development – Design and implement the API
- Database design and management – Design and implement a database to store user information
- 3D Data Visualization – Create an interactive way to view the data

1.3 Skill Sets covered by the Team

- **Frontend development** – Ayden Albertsen, TJ Thielen, Ben McClannahan
- **Backend development** – Benjamin Hall
- **Database management** – Ayden Albertsen, Benjamin Hall
- **Visualization** – Ayden Albertsen, Benjamin Hall

1.4 Project Management Style

Given the team's experience working on Agile teams, it makes the most sense to implement this management style on our project. Agile also allows for iterative development which is great for developing a unique interface for customers that don't have strict requirements.

1.5 Project Management Roles

Client/Advisor Contact & Communication – Benjamin Riemersma
Git Management Leader – Ayden Albertsen
Frontend Leader – Ayden Albertsen
Backend Leader – Benjamin Hall

2 Introduction

2.1 Problem Statement

To avoid high cost and to provide safety during exploratory stages, most manufacturers of drugs run simulations of molecular interactions and analyze the movements of atoms (in the context of multiple molecules that they belong to). The main purpose is to detect whether certain events of interest occur – which, in turn, would mean that certain properties of the drug under development are satisfied (or not). For this project's purposes, our event of interest is the formation of a Hydrogen Bond (HB) during the evolution of the chemical compound, and its persistence for a set amount of time. Our project aims to develop a system that will allow users to analyze the simulation datasets and: (1) detect the occurrence of such long-lasting HBs; (2) provide a detailed report to the user; (3) provide a visual representation of the persistent HBs, if they occur within the data set.

2.2 Intended Users & Uses

There are several classes of stakeholders that could potentially benefit from the results of this project.

1. **Chemical scientists and engineers:**
    - How will they use our project?
        - Easily filter received data and visualize results in a way that is useful to chemists.
        - Display easy to understand information about properties regarding reactions of chemical compounds to peers and others.
    - What do they gain from our project?
        - Speeds up the process of drug development.
        - Reduces time and effort on evaluating the data our application does for them.
2. **Pharmaceutical Investors and Businesses:**
    - How will they use our project?
        - Easy, high-level tool to understand drug interactions without needing to understand the details.
    - What do they gain from our program?
        - Reduce the risks when investing in development for potential drugs.
3. **Educators:**
    - How will they use our project?

- Safe environment for demonstrating properties of chemical reactions to students.

2.3 Requirements & Constraints

1. **Functional**
    a. Take input from the user regarding data set, algorithm, parameters. Parameters will change depending on the algorithm selected. The system needs to ensure that the correct parameters are provided.
    b. Output the results of the algorithm so that the user can clearly see the clusters that persist over time.
    c. System needs to scale well with multiple users as well as larger datasets.
    d. Computations need to be done on a server rather than the client's machine due to high intensity computations.
    e. Validation needs to be done on user input parameters to ensure that the system does not crash.
    f. Runtime of the algorithms needs to be reasonably efficient (Algorithms provided)
    g. Access to data should be fast and reliable.
2. **Environmental**
    a. System should not make excessive/unnecessary computations and should be power efficient.
3. **Economic**
    a. System should not make excessive/unnecessary computations and should be power efficient and should be extensible so that other datasets can be easily incorporated, and other criteria be added.
    b. Guarantees of quality of output so that drug companies can clearly see the results as real-world testing involves high costs and risks.
4. **User Interface**
    a. UI needs to provide ease of navigation.
    b. It should be intuitive to understand the (purpose and use of) different components.
    c. It should provide input validation.

2.4 Engineering Standards

IEEE/ISO/IEC 12207-2017 [1]

- Covers the common framework for the software development life cycle.

IEEE/ISO/IEC 90003-2018 [2]

- supply, development, operation and maintenance of computer software and related support services.

ISO/IEC 27001 [3]

- Standard dealing with information security and practices that should be followed.

V. Phoha, "A standard for software documentation," in Computer, vol. 30, no. 10, pp. 97-98, Oct. 1997, doi: 10.1109/2.625327. [4]

3 Project Plan

3.1 Project Management

Our group will be using an agile project management style. Our tasks/subtasks will be broken down into 2 week long sprints (Roughly 8 sprints). The goal of our project is to produce a user friendly and easy to use product which requires constant communication, flexibility, and adaptability. By choosing the agile methodology we can iteratively develop each of our subtasks and get meaningful feedback after each sprint which is crucial when developing a customer focused product.

Our progress and tasks will mainly be tracked using GitLab issues and milestones. GitLab allows issues to be assigned to specific group members and have weights assigned to them ensuring an even workload distribution for our group members. Daily standups and communication will mainly be done through Discord.

3.2 Task Decomposition

For our project we derived 5 major tasks that need to be completed in order to fulfill the functional requirements that we defined in the previous section. From the above tasks we have defined more specific subtasks that will need to be completed in order to fully complete the tasks.

Task 1: Database

1. Pick a database to store npz datasets (Could potentially use file system)
2. Pick relational database to store user information and results
3. Determine tables needed for relational database
4. Create ER diagram for relational database
5. Translate ER diagram to SQL statements
6. Set up/deploy database server

Task 2: UI

1. Design Prototype for UI (Figma)
2. Choose UI framework (React, Angular)
3. Determine list of detection algorithms and required parameters
4. Project Setup
5. Select Database/Algorithm/Parameters Page (Functional Requirement 1)
6. Import Dataset Page
7. Login/Create user Functionality
8. Job Status Page
9. View Results/Select Convoy to Visualize Page

10. Connect UI to backend (Axios)

Task 3: Server/API

1. Choose Backend Framework (Flask, Spring)
2. Project Setup
3. Obtain all algorithms being used for the project
4. Endpoint to take in job parameters and start a convoy detection job
5. Import Dataset Endpoint
6. User Login/Creation Endpoints
7. Retrieve Job Status Endpoint
8. Retrieve results of convoy detection algorithm endpoint

Task 4: Convoy Visualization

1. Discuss specific visualization needs with client
2. Choose visualization framework (Plotly, VMD as failsafe)
3. Create endpoint that produces a 3D visualization of a convoy
4. Send convoy visualization to frontend
5. Display an interactive visualization to the user in the UI

Task 5: Testing

1. Ensure that all components of the project can communicate with each other and produce the desired output
2. Ensure that all component unit tests are correct.
3. Ensure that all functional requirements are met.

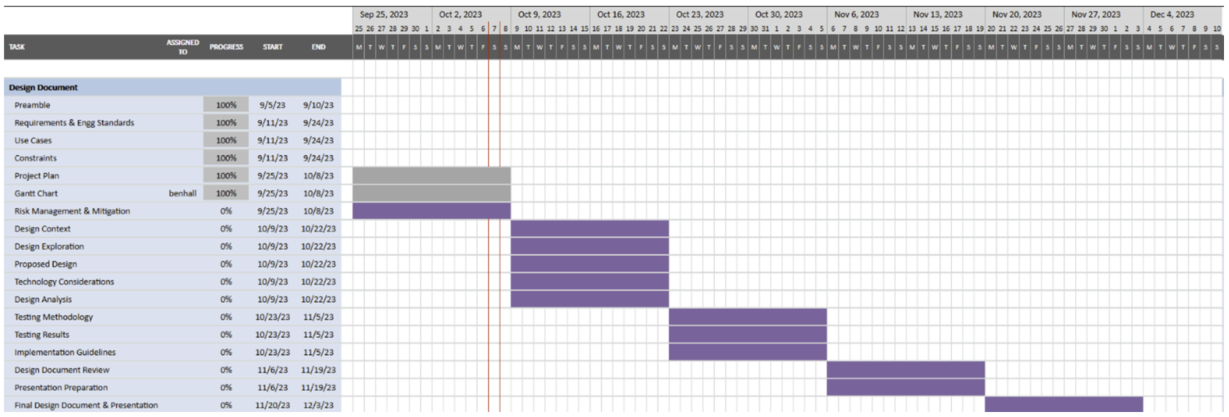
3.3: Project Proposed Milestones, Metrics, and Evaluation Criteria

Milestone	**Metrics**
Relational Database is designed, implemented, and deployed (Task 1)	Database captures/maintains the required data for the project. Database response time: Less than 10 milliseconds for 95% of queries. Transaction throughput: Handle up to 10 concurrent users without performance degradation.
Method to store and retrieve element datasets is implemented (Task 1)	Efficiency: Large datasets are stored efficiently with a compression ratio of at least 30% Data retrieval should be reasonably fast,

	and dataset should take no longer than 1 minute to be loaded into the computation system
Login/Register functions are implemented (Task 2)	Prospective users are able to register new accounts and login with them. Security: all users should have their passwords and data encrypted.
Dataset/Algorithm/Parameter selection is implemented (Task 2)	Ensures that all of functional requirement 1 is met. Users can easily select the algorithm and parameters for convoy detection in an intuitive and easy to use way
Upload Dataset Functionality (Task 2/Task 3)	System should be able to have datasets as large as 5GB. Users should be able to drag and drop datasets to upload them to the server. Upload speed should only be affected by the client's internet speed.
UI component is able to communicate with backend (Task 2)	Only authenticated users should be able to interact with backend Response time should be less than 100ms
Convoy Detection Algorithms are implemented (Task 3)	Convoy algorithm runtimes should be the same as described in journal articles. Users should have a way of seeing the status of the algorithms
All systems are able to work and communicate with each other (Task 5)	Latency between communicating from one system to another should be less than 100ms Proper https encryption standards are used when communicating over the internet

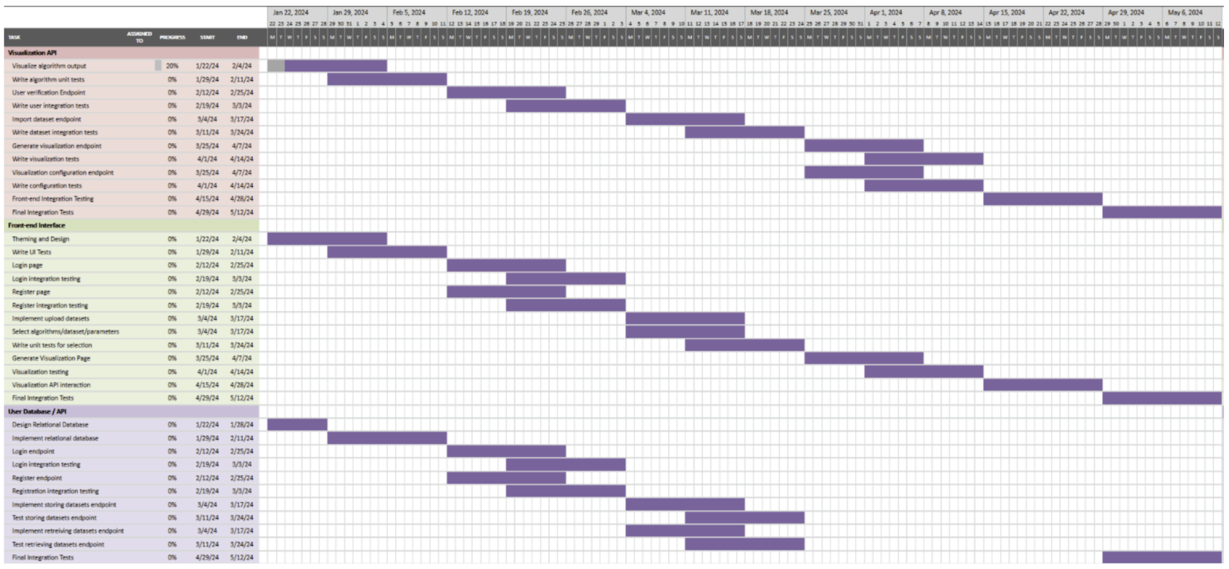
Table 3.1 Project Proposed Milestones, Metrics, and Evaluation Criteria

3.4 Project Timeline/Schedule



TASK	ASSIGNED TO	PROGRESS	START	END
Design Document				
Preamble		100%	9/5/23	9/10/23
Requirements & Engg Standards		100%	9/11/23	9/24/23
Use Cases		100%	9/11/23	9/24/23
Constraints		100%	9/11/23	9/24/23
Project Plan		100%	9/25/23	10/8/23
Gantt Chart	benhall	100%	9/25/23	10/8/23
Risk Management & Mitigation		0%	9/25/23	10/8/23
Design Context		0%	10/9/23	10/22/23
Design Exploration		0%	10/9/23	10/22/23
Proposed Design		0%	10/9/23	10/22/23
Technology Considerations		0%	10/9/23	10/22/23
Design Analysis		0%	10/9/23	10/22/23
Testing Methodology		0%	10/23/23	11/5/23
Testing Results		0%	10/23/23	11/5/23
Implementation Guidelines		0%	10/23/23	11/5/23
Design Document Review		0%	11/6/23	11/19/23
Presentation Preparation		0%	11/6/23	11/19/23
Final Design Document & Presentation		0%	11/20/23	12/3/23

Figure 3.1 Gantt Chart for Fall 2023 Semester



TASK	ASSIGNED TO	PROGRESS	START	END
Visualization API				
Visualize algorithm output		20%	1/22/24	2/4/24
Write algorithm unit tests		0%	1/29/24	2/11/24
User verification Endpoint		0%	2/12/24	2/25/24
Write user integration tests		0%	2/19/24	3/3/24
Import dataset endpoint		0%	3/4/24	3/17/24
Write dataset integration tests		0%	3/11/24	3/24/24
Generate visualization endpoint		0%	3/25/24	4/7/24
Write visualization tests		0%	4/1/24	4/14/24
Visualization configuration endpoint		0%	3/25/24	4/7/24
Write configuration tests		0%	4/1/24	4/14/24
Front-end Integration Testing		0%	4/15/24	4/28/24
Final Integration Tests		0%	4/29/24	5/12/24
Front-end Interface				
Theming and Design		0%	1/22/24	2/4/24
Write UI Tests		0%	1/29/24	2/11/24
Login page		0%	2/12/24	2/25/24
Login integration testing		0%	2/19/24	3/3/24
Register page		0%	2/12/24	2/25/24
Register integration testing		0%	2/19/24	3/3/24
Implement upload datasets		0%	3/4/24	3/17/24
Select algorithms/dataset/parameters		0%	3/4/24	3/17/24
Write unit tests for selection		0%	3/11/24	3/24/24
Generate Visualization Page		0%	3/25/24	4/7/24
Visualization testing		0%	4/1/24	4/14/24
Visualization API interaction		0%	4/15/24	4/28/24
Final Integration Tests		0%	4/29/24	5/12/24
User Database / API				
Design Relational Database		0%	1/22/24	1/28/24
Implement relational database		0%	1/29/24	2/11/24
Login endpoint		0%	2/12/24	2/25/24
Login integration testing		0%	2/19/24	3/3/24
Register endpoint		0%	2/12/24	2/25/24
Registration integration testing		0%	2/19/24	3/3/24
Implement storing datasets endpoint		0%	3/4/24	3/17/24
Test storing datasets endpoint		0%	3/11/24	3/24/24
Implement retreiving datasets endpoint		0%	3/4/24	3/17/24
Test retrieving datasets endpoint		0%	3/11/24	3/24/24
Final Integration Tests		0%	4/29/24	5/12/24

Figure 3.2 Gantt Chart for Spring 2023 Semester

3.5 Risks and Risk Management/Mitigation

Task	**Risk**	**Mitigation**	**Probability**
Task 1: Database	Data becomes compromised.	Follow ISO/IEC 27001 standards	0.1
Task 1: Database	Data is deleted.	Keep a backup of the data so that it can't be deleted or become unavailable	0.1

Task 2: UI	Framework does not meet our requirements	Define a backup frontend framework in case our first choice does not work	0.2
Task 2: UI	Unable to communicate with backend	Fallback to a different http request library	0.1
Task 4: Visualization	Visualization Framework unable to handle the amount of data	Reduce resolution of data being displayed, only show some time slices, render elements as simple dots.	0.8
Task 4: Visualization	Visualization API cannot handle several connections and requests	Implement queue system for visualization requests	0.3

Table 3.2 Risks and Risk Management/Mitigation

3.6 Personnel Effort Requirements

Task	Setup/Research Hours	Implementation Hours	Explanation
1a. Pick a database to store .npz datasets (Could potentially use file system)	3	1	There may only be a few options available
1b. Pick relational database to store user information and results	2	0	Will mostly likely use MySQL but need to ensure that it will meet our requirements.
1c. Determine tables needed for relational database	5	0	Will take some time to figure out what tables are needed

			for the project
1d. Create ER diagram for relational database	2	4	Some setup time to learn Lucid Charts or some other diagram tool
1e. Translate ER diagram to SQL statements	2	4	Just some research and testing for simple SQL statements.
1f. Set up/deploy database server	1	4	Getting access to Iowa State servers and setting up the database on it.
2a. Design Prototype for UI (Figma)	2	10	Learn Figma if the team has not learned it yet and create program flow and design themes
2b. Choose UI framework (React, Javascript)	1	0	Decide on which framework would be best suited for our team.
2c. Determine list of detection algorithms and required parameters	2	0	Research the algorithm to determine all the parameters needed from the user.
2d. Project Setup	0	1	
2e. Select Dataset/Algorithm/ Parameters Page (Functional Requirement 1)	6	12	Might have to pull available datasets and algorithms from the backend for this.
2f. Import Dataset	6	12	Developing an

Page			upload function with security concerns and writing tests in conjunction with the backend to test functionality.
2g. Login/Create user Functionality	6	12	Developing a user account system with security concerns in mind. Writing tests to confirm security and functionality.
2h. Job Status Page	8	10	
2i. View Results/Select Convoy to Visualize Page	12	16	Might take some research with the visualization aspect as the output is an HTML file.
2j. Connect UI to backend (Axios)	12	24	Integration with backend for user and dataset storage. Will need to test login, register, upload, and retrieval functions.
3a. Choose Backend Framework (Flask, Spring)	2	0	Research what framework meets our requirements.
3b. Project Setup	1	0	Setting up the project in GitLab.
3c. Obtain all algorithms being used for the project	1	0	Receive all algorithms from client.

3d. Endpoint to take in job parameters and start a convoy detection job	9	19	Will also have to develop or research queue framework for jobs.
3e. Import Dataset Endpoint	9	18	Requires integration with the user database and API.
3f. User Login/Creation Endpoints	9	19	Integration of User API and database.
3g. Retrieve Job Status Endpoint	6	12	Research the best way to implement and retrieve job status.
3h. Retrieve results of convoy detection algorithm endpoint	4	8	Requires integration testing from visualization API and frontend.
4a. Discuss specific visualization needs with client	4	0	Meeting with client.
4b. Choose visualization framework (Plotly, VMD as failsafe)	4	0	Research and decide which framework fits our client’s needs.
4c. Create endpoint that produces a 3D visualization of a convoy	9	18	Need to optimize and refine usage of visualization library.
4d. Send convoy visualization to Frontend	2	4	Need to research ways to optimally send data securely and efficiently.
4f. Display an	9	18	Researching and

interactive visualization to the user in the UI			designing an elegant way to display the results of the algorithm
5a. Ensure that all components of the project are able to communicate with each other and produce the desired output	9	18	Writing and running integration tests.
5b. Ensure that all functional requirements are met	9	18	Comprehensive testing and client feedback.

Table 3.3 Personnel Effort Requirements

3.7 Other Resource Requirements

Algorithms and datasets will need to be obtained from the clients. We will also need to obtain a server to host our various project systems.

4 Design

4.1 Design Context

4.1.1 Broader Context

Our project is designed to be used as an aid for drug manufacturers to detect how certain substrates and drug molecules may interact over time as well as detect events of interest such as convoys. Because of this, both chemists and non-technical industry professionals need to be able to use and understand our software.

Area	**Description**	**Examples**
Public Health, Safety, and Welfare	How does your project affect the general well-being of various stakeholder groups? These groups may be direct users or	Aid in providing a comprehensive tool to automate convoy detection and allow chemists to develop

	may be indirectly affected (e.g., solution is implemented in their communities)	potentially lifesaving drugs faster.
Global, Cultural, and Social	How well does your project reflect the values, practices, and aims of the cultural groups it affects? Groups may include but are not limited to specific communities, nations, professions, workplaces, and ethnic cultures.	Chemists' workload will be lightened by removing the tedious repetitive task of detecting convoys by hand enabling a more effective exchange of findings and processes.
Environment	What environmental impact might your project have? This can include indirect effects, such as deforestation or unsustainable practices related to materials manufacture or procurement.	The servers that run the convoy detection algorithms will use energy, however, our system will enable more efficient separation of potential reactions among atoms without sacrificing the effectiveness.
Economic	What economic impact might your project have? This can include the financial viability of your product within your team or company, cost to consumers, or broader economic effects on communities, markets, nations, and other groups.	Drug research can be extremely costly and have risks involved. Our system will mitigate some of these costs by using simulated data and detecting convoys with efficient algorithms. This can make for faster drug research as well as quicker time to market.

Table 4.1 Broader Context Considerations

Our design attempts to address the needs of the client by creating two fundamental systems; (a) a frontend that provides the UI/UX to the user as well as display the rendering of the convoy displayed to the user, and (b) a backend that handles data access, computation, and visualization rendering.

4.1.2 Prior Work/Solutions

There are no comprehensive systems that provide convoy detection and visualization using a cohesive user interface. There has been prior work done with studying molecular flocks [9] and convoys [8]. The downside of these two studies is that they neglect to consider tolerance gaps in convoy detection which is still interesting to drug manufacturers. There also exists software that can visualize molecular trajectories but none that work in conjunction with a convoy detection system.

4.1.3 Technical Complexity

The design consists of multiple components/subsystems that each utilize distinct scientific, mathematical, or engineering principles.

1. **Frontend**
    a. Authentication/Authorization
        i. Users need to view only data assigned to them.
        ii. Users need to log in to access their data
    b. UI/UX
        i. Database, algorithm, parameter selection
        ii. View Job Status
        iii. View Results
    c. Visualization of convoys
        i. Interactive
        ii. 3D
2. **Backend**
    a. Data Access and Storage
        i. Upload and retrieve stored datasets in the database
        ii. Load datasets into algorithm parameters
    b. Job Broker
        i. Start job on API request and return Job Identifier.
        ii. Portion out computer resources for algorithms.
        iii. Retrieve and return status of Job from an identifier.
    c. Convoy Computation/Processing
        i. Load large datasets into algorithm
        ii. Run a computationally difficult algorithm while several requests are going on
        iii. Return processed data to visualization
    d. Visualization Rendering
        i. Take in the processed data and render the dataset to an HTML format for easy and dynamic viewing by the user
        ii. Reduce size for output
        iii. Reduce resolution of data

4.2 Design Exploration

4.2.1 Design Decisions

4.2.1.1 Backend

Defined Criteria:

- Needs to be able to work with the data and algorithms provided by the client.
- Needs to be able to scale with multiple users.
- Needs authentication support.
- Needs to be able to communicate with frontend using HTTP
- Support for rendering visualizations

Given the criteria we narrowed down our options to two different frameworks:

- Java Spring
    - Group Familiarity/Experience
    - Fast
    - Authentication Support
    - Testing Support / Dependency Injection
    - Good Documentation
    - Bloated Codebase
    - Not compatible with given datasets
- Python Flask
    - Older framework
    - Little group experience
    - Compatible with given datasets and algorithms
    - Performant
    - Lightweight
    - Can work with many visualization and job queue frameworks.

After analysis of our two options, we have decided to go with Flask to create our backend. Its compatibility with the given datasets being its biggest draw. It will also give the group an opportunity to learn a new framework and for some of us a new language. Though this will come at the cost of added time as members may have to do more initial research before completing their tasks. Python also has a larger number of graphing/visualization framework available which also led us to choose Flask.

4.2.1.2 Frontend

Defined Criteria:

- Needs to be able to give a good user experience.

- Interface with our visualization framework to display information

Given the criteria we narrowed down our options to two different frameworks:

- React
    - Most popular frontend framework
    - Group Familiarity
    - Flexible
    - Larger community
    - Larger file size
- Vue
    - Second most popular frontend framework
    - Smaller file sizes
    - Better for simplicity

After analysis of our two options, we have decided to go with React to create our frontend. Of all our members, we have the most experience with React. Additionally, it has great community support as it is the most popular frontend framework that developers use.

4.2.1.3 Visualization

Defined Criteria:

- Display convoys through various time periods
- Output multiple convoys
- Highlight hydrogen bonds within convoys.
- Output results to our frontend
- Fast and efficient

Given the criteria we narrowed down our options to two different frameworks:

- VMD
    - External application
    - Unknown API
    - Suggested by client
    - Built for modeling molecules
    - Built-in scripting
    - Lacks interactivity
- Plotly
    - Python library
    - Interactive
    - Flexible
    - May lack complexity and tooling for molecule modeling.

After analysis of our two options, we have decided to go with Plotly to create our visualization framework. We concluded that VMD would require too much work to interact with our frontend and it would be better to use a more flexible tool like Plotly over one tool designed for molecule modeling. Plotly being a Python library also allows for simplicity without a backend API being developed in Python as well.

4.3 Final Design

4.3.1 Overview

This project can be broken down into two main parts: the frontend, and the backend. The frontend will encompass parameter selection, the API service, the Results, and the Visualization. The backend will contain the Flask API, the Convoy Job Queue, the Visualization Render Queue, and communicate with the database.

The process will start on the frontend, where a user is able to upload a dataset for the convoy detection algorithms to run on. This will go through the API Service to communicate with the Flask API, which will then upload that data to the database. After this is done, the user can use the Parameter Selection component to determine the specifics of how the convoy detection algorithm will work. It will pass that information to the API service, which will communicate with the Flask API. The Flask API will send this information to the Convoy Job Queue. This will execute the algorithm needed to detect convoys and return the results to the API. Next, it will send these results to the Visualization Render Queue. This will create a visualization using Plotly to display information about the convoys. Finally, the Flask API will send back this visualization to the API Service, which will send this information to the frontend for the user to view.

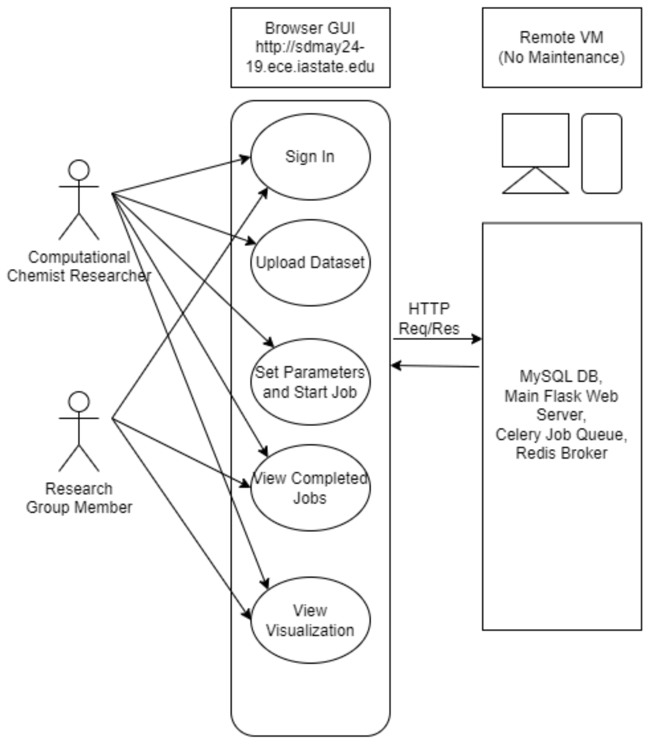

Browser GUI
http://sdmay24-19.ece.iastate.edu
Remote VM
(No Maintenance)
Computational Chemist Researcher
Research Group Member
Sign In
Upload Dataset
Set Parameters and Start Job
View Completed Jobs
View Visualization
HTTP Req/Res
MySQL DB, Main Flask Web Server, Celery Job Queue, Redis Broker

Figure 4.1 Use Case Diagram

4.3.2 Detailed Design and Visuals

Based on our use case model (cf. Figure 4.1), here is our diagram of our system and subsystems:

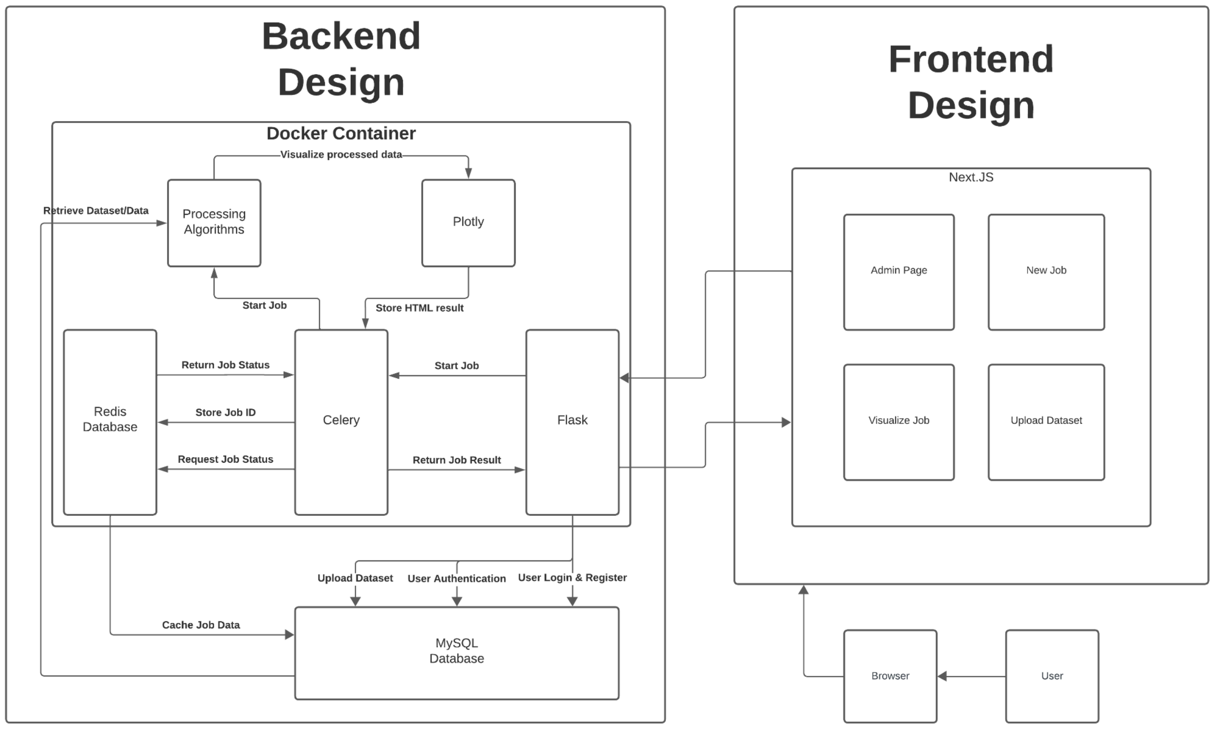


Figure 4.2 System Block Diagram

4.3.2.1 Frontend

- **NextJS** is a web development framework for providing React-based web applications with server-side rendering and static website generation. This provides us with better development, performance, and scalability.
- **Admin Page** provides an interface for the user to interact with datasets, companies, and jobs.
- **New Job** provides a page for the user to specify parameters for a new job, and start them.
- **Visualize Job** provides a page for the user to visualize the processed data from the jobs that they and their company have created.
- **Upload Dataset** provides a page to upload new datasets for use in their own and their company's jobs.

4.3.2.2 Backend

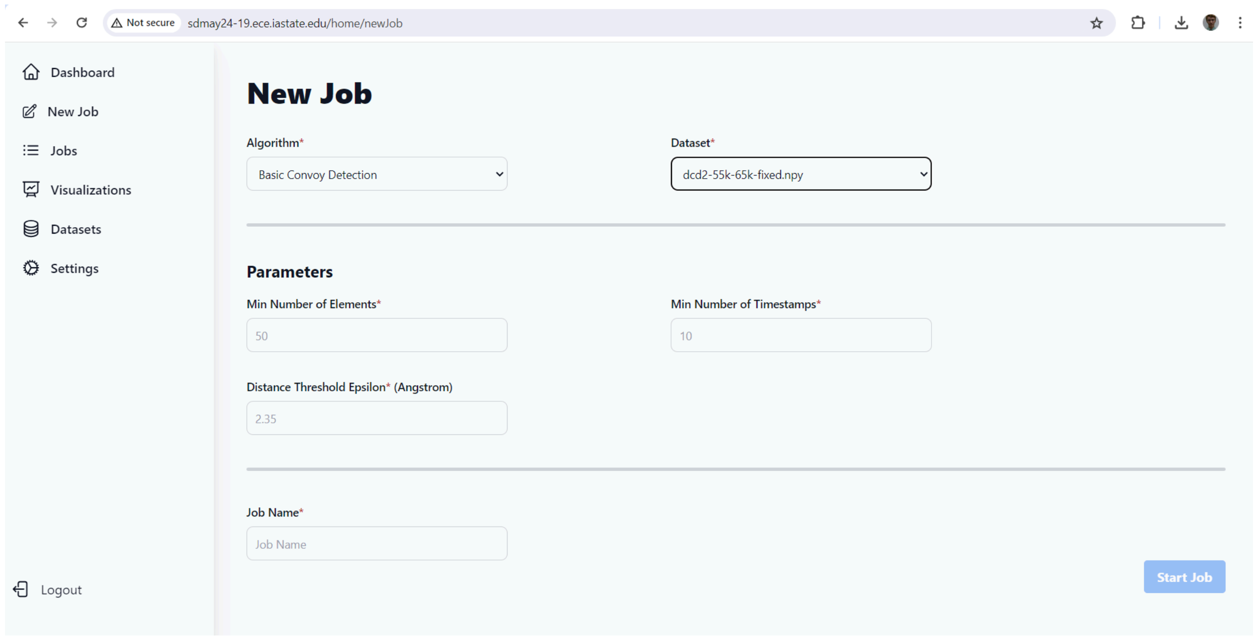
- **Flask** [6] provides a stable but updated library for developing and hosting an API through Python.
- **Celery** [10] is an asynchronous job queue library that will be used for starting and retrieving job status. This tool will provide the most flexible and fast response to

users. It provides interactivity with Flask which makes it perfect for this use case. It will allow for the client to request new jobs, their status, and their results.

- **Redis** [11] is an in-memory database which can be used as a broker for job statuses for Celery.
- **Docker** [12] allows for quick deployment of the multiple backend scripts like Flask and Redis, making it a more deployable package.
- **MySQL** [13] provides a traditional database structure perfect for storing user data, the stored data sets uploaded by the user, and caching results of jobs for the long-term.
- **Processing Algorithms** were provided by the client for use in our backend design.
- **Plotly** [7] provides an expansive visualization library which can output a HTML rendering of the data after processing, which can be returned by the API.

4.3.3 Functionality

This design is intended to operate in the real world by first allowing chemists who need to view the formation of convoys and hydrogen bonds between molecules to upload their data to the database. Next, they can select the necessary parameters and algorithm to execute(figure 4.3). Both this and the visualization of the results will be done remotely and sent back to the frontend for the users to view.



New Job

Algorithm*: Basic Convoy Detection

Dataset*: dcd2-55k-65k-fixed.npy

Parameters

Min Number of Elements*: 50

Min Number of Timestamps*: 10

Distance Threshold Epsilon* (Angstrom): 2.35

Job Name*: Job Name

Figure 4.3 Parameter Selection

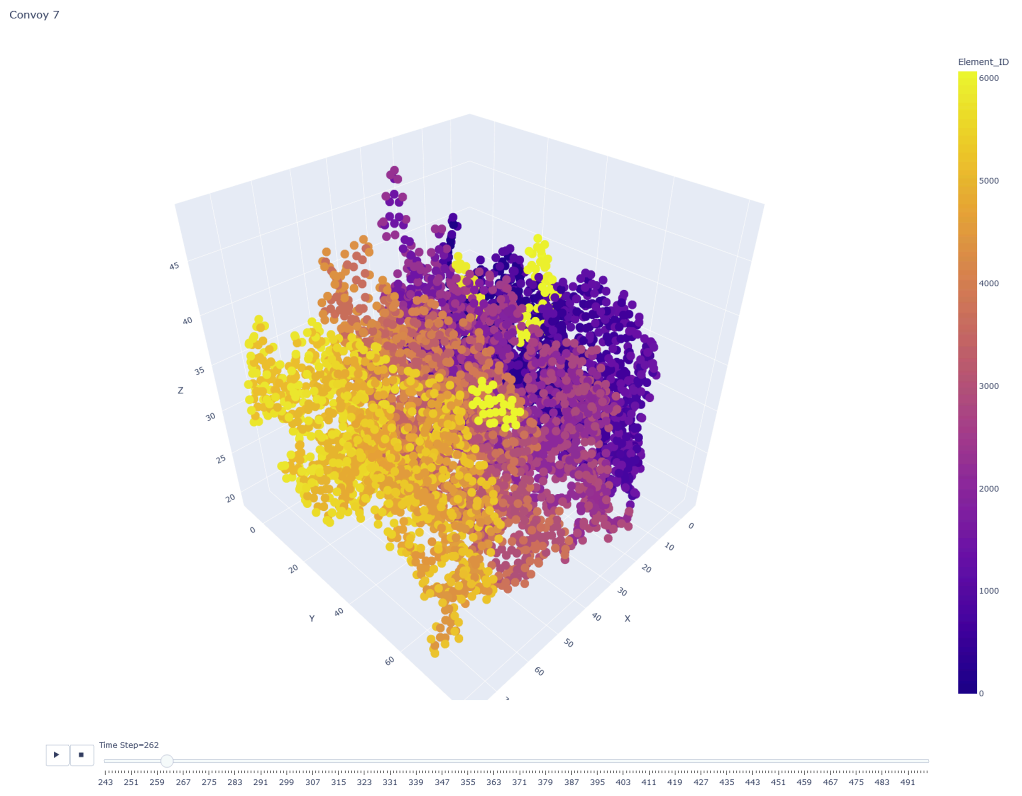


Figure 4.4 Convoy Visualization

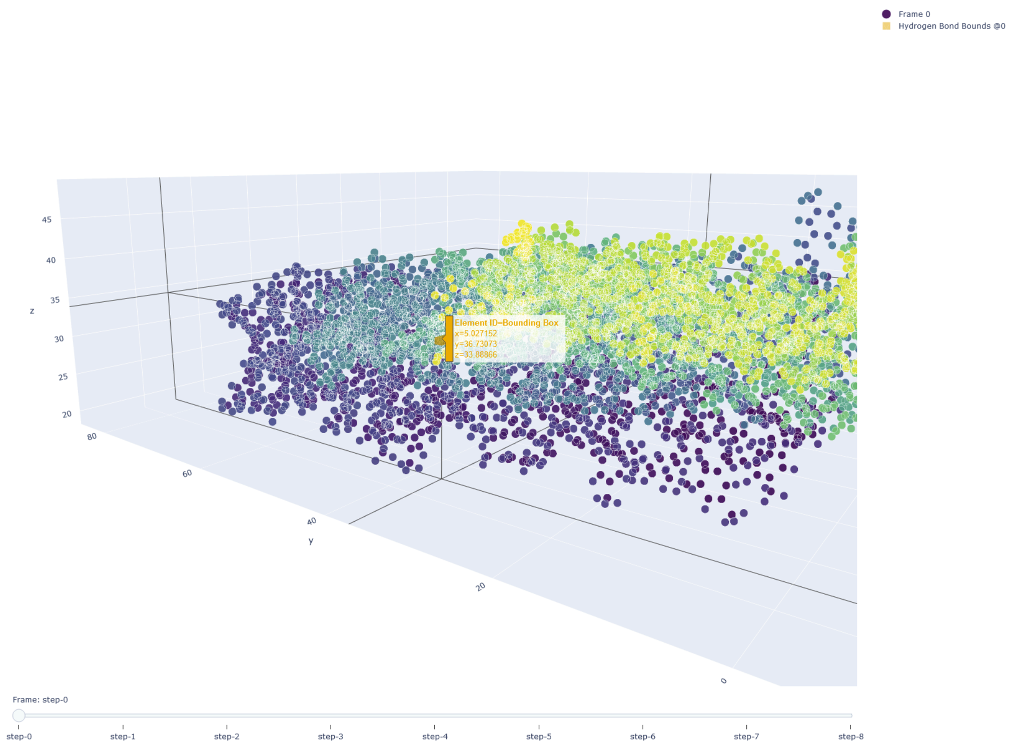


Figure 4.5 Hydrogen Bond Visualization

4.3.4 Areas of Concern and Development

We had bi-weekly discussions with our client to ensure that our project was meeting all the requirements and never strayed from the requirements. For example, one of the biggest needs expressed was the ability to visualize hydrogen bond groups over a period of time. We detailed those decisions in Section 4.2.1 of this document. Just like with that decision, we went through the creation of evaluation criteria for each decision, allowing us to act confidently and consistently in our decisions. We also went over our decisions and why we made them with our client to ensure they felt that the project was meeting their needs.

While we considered different frameworks and technologies to use, we had concerns whether our decided framework would be sufficient for the client's needs. We continued to communicate with our client in order to ensure that our design met all their requirements as development progressed.

4.4 Technology Considerations

When making the design decision on the backend language, we analyzed the strengths and weaknesses of both Java Spring and Python Flask. We ended up landing on Flask due to its good performance, light weight, and ability to work with many visualizations and asynchronous jobs. There were some disadvantages including being an older framework, and a lack of Flask experience within our group. We could have gone for Java Spring, which would have been easier to get started with due to more group experience, however we determined the advantages of Flask outweighed those of Spring.

Another decision that was made was with our visualization technique. It was initially suggested that we try to use VMD, but after further research we found a much better solution – plot.ly. Plot.ly is a python library that allows you to create a visualization similar to what is possible with VMD, however it is lighter and works really well with our current design, so we opted to choose plot.ly.

Finally, we had the opportunity to choose either React or Vue for the frontend framework. They had very similar strengths and weaknesses but given that our group has more experience in React, we decided to use React. Additionally, Next.js was chosen to provide improved page transition functionality on top of the methods provided by React. Next provided a clean way to implement URL path arguments.

4.5 Design Analysis

The tasks on this project were highly effective at directing the work to be done. The setup and research hours were effective at predicting the expected time commitment for each work item. In particular the 2i View Results/Select Convoy to Visualize Page and 2j. Connect UI to backend were some very time intensive tasks with some design changes to be implemented. That design change being the adoption of the React-Plotly module.

While the CMC Detection algorithm was given to us at the beginning, that did not make the visualization process simple. We did not know how much screen space was available and the proper coloring of convoys and contexts was not apparent without client interaction, and some trial and error aka testing.

8 2-week sprints were accurate. There were concrete tasks which needed to be completed and they were distributed amongst team members. Communication was highly encouraged in order to get advancements completed.

4.5.1 Security Concerns and Countermeasures

Throughout all points in our application, we ensured that user-provided data had been sanitized and validated. We also ensured that all libraries used in our project were reputable and maintained consistently. All user credentials were properly handled with proper hashing, salting, and peppering.

5 Testing

In this section we present details of our test methodology that was used throughout the development of the project. We note that the functional and nonfunctional requirements were described in Section 2.3 and they were translated into respective system components in Section 4.3. We will refer to Figure 4.1 and Figure 4.2 of Section 4 when describing the envisioned testing.

5.1 Unit Testing

Currently Selenium is used to test frontend login, account security, web page transitions, data set upload, and other frontend functionality. This is a quick tool which allows for automatic testing of client-side components, which can be updated manually with the addition of frontend features.

For the backend, PyTest is used to test basic functionality of python functions. This is a quick tool which allows for automated testing of server-side components, and can be updated manually with the addition of backend features.

5.2 Interface Testing

With only 7 web pages, user interface testing was conducted manually. Problems with the UI are more qualitative and would be discussed between the client and developers. Backend HTTP API Tests were completed with the Postman API testing application. This application keeps many tools for testing endpoints. Postman HTTP Requests were shared between developers to keep Request/Responses transparent and accessible.

5.3 Integration Testing

There are many integration paths which require testing. User authentication and data retrieval were simpler to test just through observation. However this project is stateful in that the data in the Redis and MySQL servers along with the local files dictate how the project runs. Each and every state was not tested. Cypress was a promising tool to test the frontend with multiple data states. But testing of our code with multiple data states was done manually via observation and manipulation of backend files and database records.

5.4 System Testing

The core processes of the project involve the visualization of large groups of atoms with convoys and the browser retrieval and rendering of these figures. Visualization of the groups of atoms with Plotly's scatterplot 3d was proven in November. Adjustments of the complete figure were established through client interaction with Hasan Anowar MD. Client-side testing of the visualization of Plotly figures is pretty binary, the figure shows, or it doesn't. This can be tested via Selenium like other unit tests.

5.5 Regression Testing

Gitlab provides runners which organize and execute our source code on the VM for the main branch. This preliminary tool provides responsive information on the total code base based on the runner's success. This is an essential, if rudimentary test, that was very beneficial for the acceleration of development.

5.6 Acceptance Testing

For meeting functional and nonfunctional requirements, we directly contacted and worked with our client. The acceptance of the project was decided based on criteria such as: intuitiveness of the UI, accuracy of the implementation of algorithms, and the accuracy of the visualization. We also involved the client in different stages of integration testing.

5.7 Security Testing

Data security testing for this project was a stretch goal. At this time the website is hosted on ISU's network. Passwords are encrypted on the database and credentials are required to use the website. Penetration testing and hardening would be beneficial for future work items. Currently only password verification is tested as a side effect of unit testing.

5.8 Results

Our unit tests and regressions tests ran every time we updated the system to ensure continued functionality of the project. Our client has approved of the state of and appearance of our project at this time. All major bugs discovered during testing have been eliminated. For example, we found issues with the synchronization between the MySQL database and the Redis database. All tests currently perform to a satisfactory level.

7 Implementation

7.1 Overview

Our implementation process was to start by adding the core functionality to our project and gradually implement more adjacent features and functionalities to create our minimum viable product and expand to create a more cohesive and comprehensive project. This allowed us to refine our most complex and important task first to nail the product's core functionality. Our first core features included creating and managing convoy detection jobs, creating and managing hydrogen bond detection jobs, and visualizing said jobs. Less important features include the management of companies and datasets. This process allowed us to work in an agile methodology with simultaneous development.

There were several areas of development that contributed to our final project. The first area was the acquisition and setup of a project infrastructure. This includes the server used in the project which hosted our docker containers, development operations (DevOps), and our database. The first task towards implementing our infrastructure was to acquire a server with sufficient resources from the Electronics and Technology Group (ETG) at Iowa State University. Our team acquired a server that was able to host our Dockered Flask application, our MySQL database, and our CI/CD pipeline for Development Operations. Docker was responsible for deploying our backend tools like Flask, Celery, and Redis. The MySQL database was responsible for storing user data, datasets, and cached results from user jobs. We initially faced challenges on obtaining a server with enough resources as the algorithms our client provided us were resource intensive.

The next area of development was the implementation of the client-provided algorithms towards our Flask API and Celery library. The implementation started out as a 1:1 copy of the provided algorithm into Celery, however this quickly caused issues that needed resolving. Other algorithms required data that was processed in the middle of the algorithm and had no way to export it. We later decided to divide the algorithm into the smallest basic tasks and named those "Jobs" that Celery would queue for our back end to process. With this, users can begin new jobs, query their status, and obtain their end result.

Another area of development was the implementation of visualization. This was implemented as a simple endpoint for the API in which users could request specific visualizations (convoys, hydrogen bonds, or the convoys in context). The visualization was implemented with the Plot.ly library, using a mix of default configuration and custom configurations for the Hydrogen Bond visualization. With this implemented, the frontend is able to request the visualization in which it will embed the returned HTML visualization from the backend into the UI.

The last area of development was the implementation of companies and datasets. This was implemented with JWT authentication and retrieving information from the MySQL database. Users are able to upload datasets with selectable permissions for the public, company, and themselves. They can restrict read and write permissions for datasets and jobs within this.

7.2 Evolution of Design

In this section we note the changes that our project encountered during implementation and why they were needed.

Our original design planned to utilize the React framework for our frontend. After our initial testing, we realized that React was not performant enough for the project requirements. We decided that NextJS would better meet our requirements as it had a better development environment and provided better performance.

Another evolution in our project was the interaction between our two databases, Redis and MySQL. Initially, the jobs from celery would stay only in the Redis database. However, Celery and Redis do not allow for querying the status of jobs in progress. This means we had to create an additional table in MySQL to store intermediary data on these jobs. Ultimately, we reconfigured Celery to use MySQL as the results database and only use Redis for the broker database.

8 Conclusion

7.1 Review

This website should be a convenient tool for chemists and drug researchers to effectively visualize the mechanics of computed chemicals under study. The project has a professional and reactive design which instills confidence from users. Clients should be able to utilize this project as they need as the virtual machine will be kept online running after this semester. All of the core software technologies(Flask Framework, React Framework, Celery, Redis, and MySQL) declared to be used by the design document were used in their intended manner within this project. The Gantt chart was a very useful guide even if it wasn't able to be followed perfectly.

7.2 Value

In this use case, chemists should be able to visit the website on lightweight mobile devices such as tablets and laptops and see a visualization of which chemicals hold proximity to each other. This way chemists with computers that have no graphical hardware, or those with operating systems which do not support VMD can view these complicated processes. Molecular motion at this scale is rarely seen and hopefully this website can make this more well known.

7.3 Future Steps

There are a couple of directions this project could grow in. More customizability in algorithm selection, better compression of visualizations for faster intractability with visualizations on the front end, a public collection of NumPy frame files, even contributing to the Plotly project could be a benefit. There is not one specific direction, but the broad categories for future development of this project are: improving customizability, increasing documentation and education on these processes, improving accessibility, increasing security, and automating server maintenance.

Appendix 1 - Operation Manual

The Remote Repository which holds our source code can be found here:

https://git.ece.iastate.edu/sd/sdmay24-19

To run the project, follow the following guide:

On a Windows, Linux, or MacOS machine ensure the following tools are installed: Docker Engine, Docker Compose, Node.js 20.11.1, Python 3.10, and MySQL Server 8.0 CE.

We recommend opening up sdmay24-19 in an IDE, for these instructions we will be using VScode. MySQL is often most comfortably used with MySQL Workbench.

Database

1. With MySQL Workbench, open and execute the SQL file located at `../sql/table_init.sql`

Backend

1. Open up one terminal and navigate to the directory, `/backend`
2. Run the following command:

   ```
   docker-compose up -d --build.
   ```

   This should create 3 Docker Images and from them 3 containers.

Frontend

1. Open up a concurrent terminal and change directory to `../frontend/convoy`
2. Run the following command:

   ```
   npm run init
   ```

3. Access the web application at `https://localhost:3000/`

Appendix 2 - Other Considerations

Team 33 had this exact same project one year earlier. Our project set out to succeed at some tasks which the other team's project had struggled with. Those being a single responsive UI, and fast download times for large datasets. They also struggled to implement the hydrogen bond detection, which was an emphasis for our implementation.

Appendix 3 - Code

This project is predominantly broken down into 2 servers and 2 databases: Node.js and Flask, MySQL and Redis.

Node, a well supported JavaScript compiler, hosts the browser web pages. All of these are organized by the React Framework. Additional core packages are Next.js, Tailwind CSS and Plotly. Next.js handles web page transitions and URL organization. Tailwind CSS provides styling for web pages from within JSX files. Plotly is the collection of JS modules responsible for visualizing contexts, convoys, and bounding boxes.

Flask, a Python 3 Web Server framework. Crucial to the implementation of the backend API is the use of the Celery, multithreaded task queue, in conjunction with the Redis, in-memory key-value DBMS. Redis acts as the task broker for Celery, and all of these are seamlessly integrated into the total backend.

The MySQL RDBMS is used for secure management of users, and company account information.

References

[1] IEEE SA, “IEEE Standards Association,” IEEE Standards Association. https://standards.ieee.org/ieee/12207/5672/

[2] IEEE SA, “IEEE Standards Association,” IEEE Standards Association. https://standards.ieee.org/ieee/90003/7197/

[3] “ISO/IEC 27001:2022,” ISO, Feb. 02, 2023. https://www.iso.org/standard/27001

[4] “A standard for software documentation,” IEEE Journals & Magazine | IEEE Xplore, Oct. 01, 1997. https://ieeexplore.ieee.org/document/625327

[5] “React.” https://react.dev/

[6] “Welcome to Flask — Flask Documentation (3.0.X).” https://flask.palletsprojects.com/en/3.0.x/

[7] “Plotly.” https://plotly.com/python/

[8] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen, “Discovery of convoys in trajectory databases,” Proceedings of the VLDB Endowment, vol. 1, no. 1, pp. 1068–1080, Aug. 2008, doi: 10.14778/1453856.1453971.

[9] J. Gudmundsson and M. Van Kreveld, “Computing longest duration flocks in trajectory data,” Nov. 2006, doi: 10.1145/1183471.1183479.

[10] “Introduction to Celery — Celery 5.3.6 documentation.” https://docs.celeryq.dev/en/stable/getting-started/introduction.html

[11] “Redis,” Redis. https://redis.io/

[12] “Docker: Accelerated Container Application Development,” Docker, Oct. 18, 2023. https://www.docker.com/

[13] “MySQL.” https://www.mysql.com/