# Detecting convoys of molecular and atomic trajectories

FINAL REPORT

33
Iowa State University
Goce Trajcevski
Claira - Documentation, Testing
Jeremy - Frontend Design
Daryl - Backend Design
Jason - Integration
sdmay23-33@iastate.edu
http://sdmay23-33.sd.ece.iastate.edu/

## Table of Contents

# Table of Figures

## Figures

# 1 Team

## 1.1 Team Members

Jason Guo

Daryl Kay

Claira Springer

Jeremy Lewis

## 1.2 Required Skill Sets for Your Project

Skills required:

- Web Development
- Testing
- Documentation
- Web Server
- Database

## 1.3 Skill Sets covered by the Team

Documentation - Claira, Daryl

Web Development - All Members

Testing - Claira, Jeremy

Web Server - Daryl, Jeremy

Database - All Members

## 1.4 Project Management Style Adopted by the team

For our project we went with the agile development life cycle due to the fact that it works very well in an all software based project. This will allow us to divide our project into sprints with specific purposes and will also allow us to incorporate feedback from the client very easily.

## 1.5 Initial Project Management Roles

Claira- Administration, client interaction, project management, documentation

Jeremy- Individual component design

Daryl- Testing

Jason- Overall design

# 2 Introduction

## 2.1 Problem Statement

Chemists responsible for drug manufacturing have an interest in detecting convoys of molecules and finding interesting interactions between simulated atoms, such as hydrogen bonding. Current methods include looking through data by hand and using algorithms with interpretations, which is borderline impossible with the amount of data that is simulated. We sought to solve this issue by building a graphical interface that would allow a user to quickly search for and find interesting events as they define them, making it easier for chemists to complete their work.

## 2.2 Requirements

Functional Requirements

- Scale well with multiple users
    - Multiple users are able to access the system at one time and request individual queries
    - Users only have access to their specific results and not those of other users using the system
- Hard computations on the server
    - No computations occurring on the front end, all computations are to occur on the backend
    - Front end should  send parameters to the backend and also display outputs of the queries
- The system does not crash with certain input parameters (too large)
    - Frontend has input sanitization to make sure that values imputed are in the proper ranges
    - Backend can contribute to this as well with checking for variables being outside of acceptable ranges

Non-functional requirement

- Intuitive UI
    - The UI is be very user friendly and require little to no assistance for the users to be able to perform their desired tasks on the system
- When bulk inserting new data, the downtime for users should be minimal
    - New data is to be queued to be downloaded/uploaded during off hours which can be set by the users but will most likely be during the night

## 2.3 Use Cases

User 1: Chemists
Characteristics: Intelligent, Stressed, Busy, Detail-Oriented
Needs: Needs a way to easily filter through data and get easy to understand and visualize results, reduce costs to further extend the funding that they receive, expedite results to secure more future funding
Benefits: Saves time and work for the chemists as they will have a program to do these tasks for them, reduce costs in materials, speeds up the drug development process

User 2: Experts from other fields, collaborating with chemists
Characteristics: Less Knowledgeable about the specific chemistry, more experts in data analytics.
Needs: This user needs a way to have a better understanding of the results of simulations without having to understand the data specifically
Benefits: Will have the ability to look at visualizations of the specific data that is interesting and allows for easier communication between the chemists and them.

User 3: Managers in the pharmaceutical industry/business decision-makers
Needs: They need a high-level interactive tool to be able to understand the potential benefits of turning a simulation into actual experiments.
Benefits: Will be able to minimize the risk for investments into experimental setup(s) which could lead to eventual production.

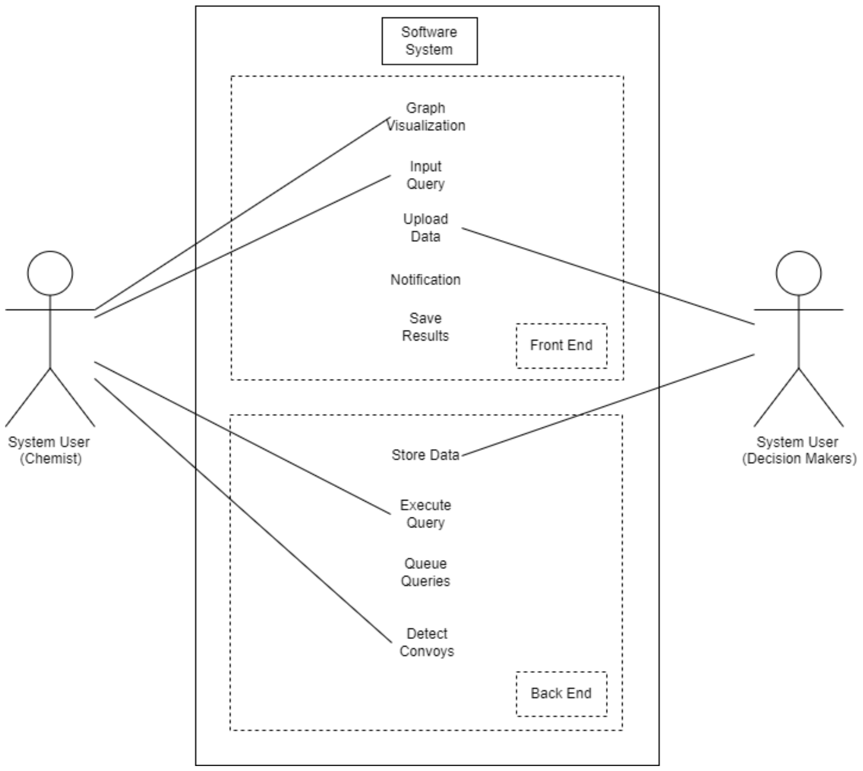


**Figure 1:** Use case diagram

### 2.4 Engineering Standards

IEEE/EIA 12207

Covers the common framework for the software development life cycle

IEEE/ISO/IEC 90003-2018

An extension of ISO 9001:2015 which is also includes the software development cycle but also other factors that will boost customer satisfaction

ISO/IEC 27001

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.

Based on  ANSI/ANS 10.3-1995 a standard used for the documentation of computer software and development

# 3 Project Design

## 3.1 Design Context

Our design problem is going to be used in the realm of chemistry, more specifically to better understand molecular convoys. This includes chemists who will be able to use our software, as well as people who don't work in chemistry but need to use our software to complete their job that is associated with chemistry. Potentially, people who don't understand chemistry at all will need to use our software.

There has been some prior work done in the realm of studying molecular convoys. Primarily, simulations have been used but then studying the simulation data is very manual. There are also products that visualize the outcome of the simulations, however, there hasn't been work to then visualize the convoys occurring. On top of the previously stated methods, there has been some exploration of how machine learning can be used to see the outcomes of chemical reactions.

## 3.2 Technical Complexity

1. Frontend
    a. Graph visualization
        i. Use of a third-party library/framework to visualize data that is given to us by chemists
    b. Scaling with users
        i. Ensure that all users are able to access and view data assigned to them
    c. Scaling with data
        i. Ensure that data of any size is able to be handled and displayed
2. Backend
    a. Convoy algorithms
        i. Have algorithms that detect and mark molecular convoys correctly according to specifications given
    b. Upload data
        i. Be able to upload data to process and use
    c. Execute queries
        i. Execute queries from the front end and send data requested in a timely manner
    d. Queues
        i. Queue query requests from the front end
3. Chemist data
    a. Individual motion of atoms vs collective molecules
        i. Most of this information will be given to us in a file and it will be our job to process this information and sort it into various categories for those using the system to look into

## 3.3 Design Decisions

The decision was made early on in ideation to employ React.js for our frontend, and a sql database for our backend, in addition to VMD. These decisions were made by weighing other potential software that we ultimately decided were not as suited towards our project.

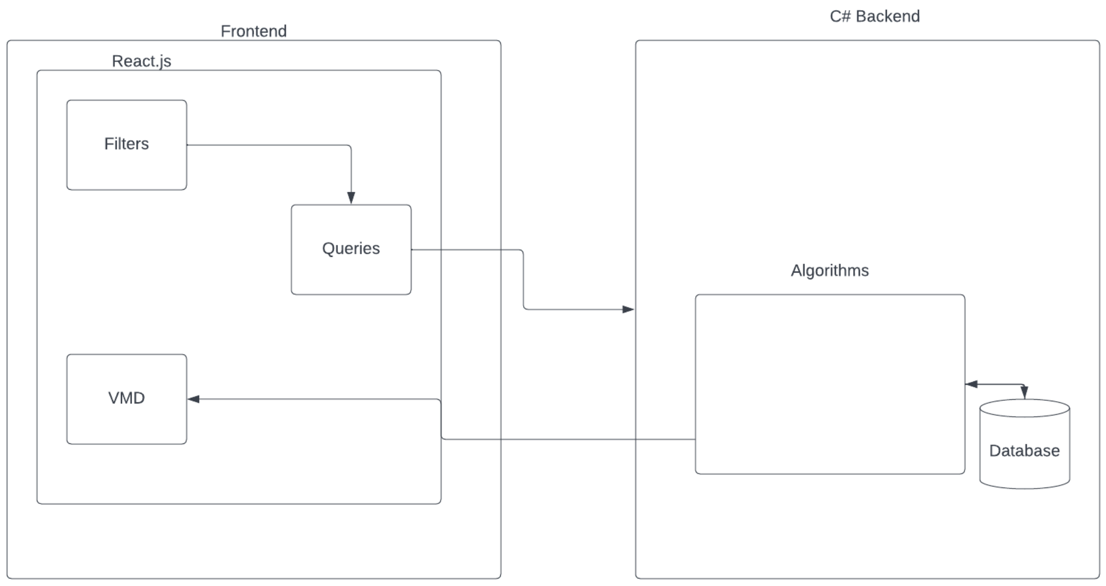


**Figure 2:** Domain Diagram

This diagram above was our initial design which had VMD housed inside of the frontend. After starting to work with it we realized that it would be difficult to embed it into the frontend as there isn't a library for VMD that would allow us to do this easily. We could attempt to stream an open VMD application to the user but this wouldn't be efficient. Due to this we decided to make the backend make an output file that could be turned into a script to then be run in VMD as a separate application. The new design is shown below. This change is discussed in section 3.5 as well.

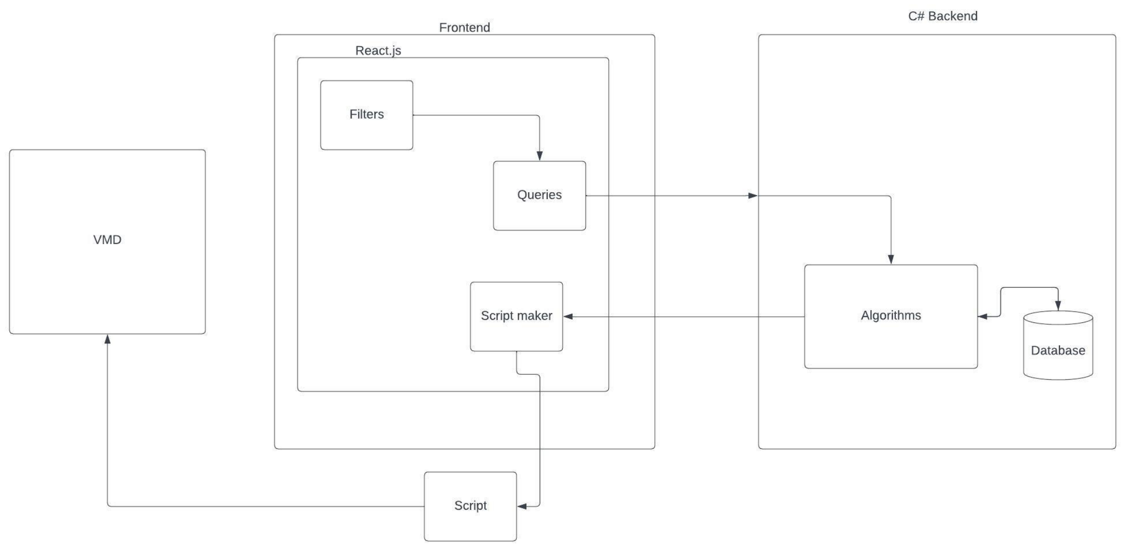


**Figure 3:** New Domain Diagram

## 3.4 Components

Filters/Queries - We consider this part the first step in the process. During this step, the user is inputting data that they want to see back in the VMD module. These two modules are part of our front end in react.

Algorithms - This part of our design is where the CPU-intensive algorithms are run. We find out what queries we need to run from the filters/queries module.

React - This part will be our vessel to communicate with the API and backend to get the results form the queries and also to make the scripts that will run in VMD.

VMD - Will run our scripts and visualize the queries

Database - This section stores our information from the simulations in order to be able to be used by the algorithms.

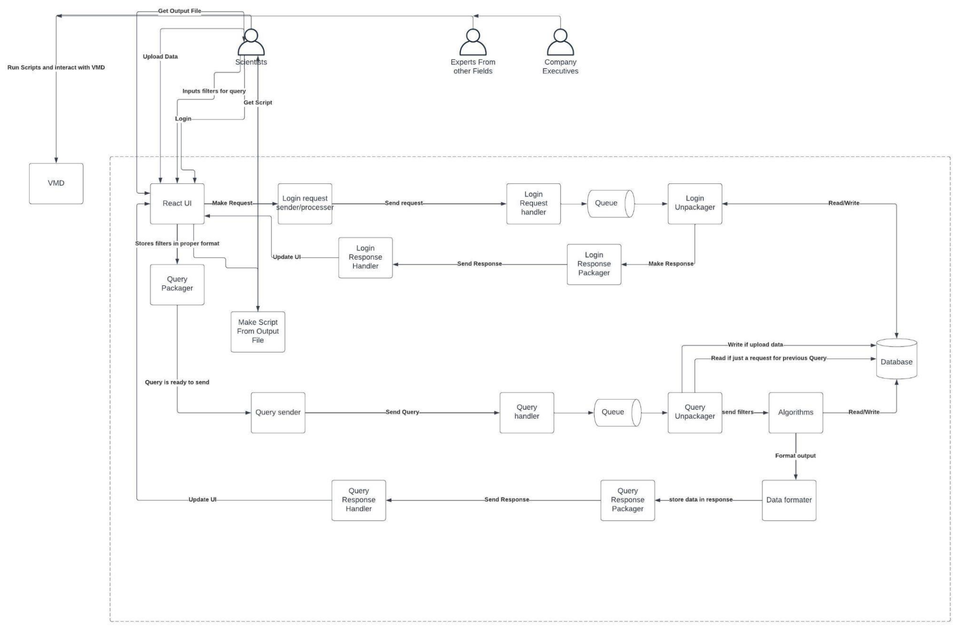


**Figure 4:** Component Diagram

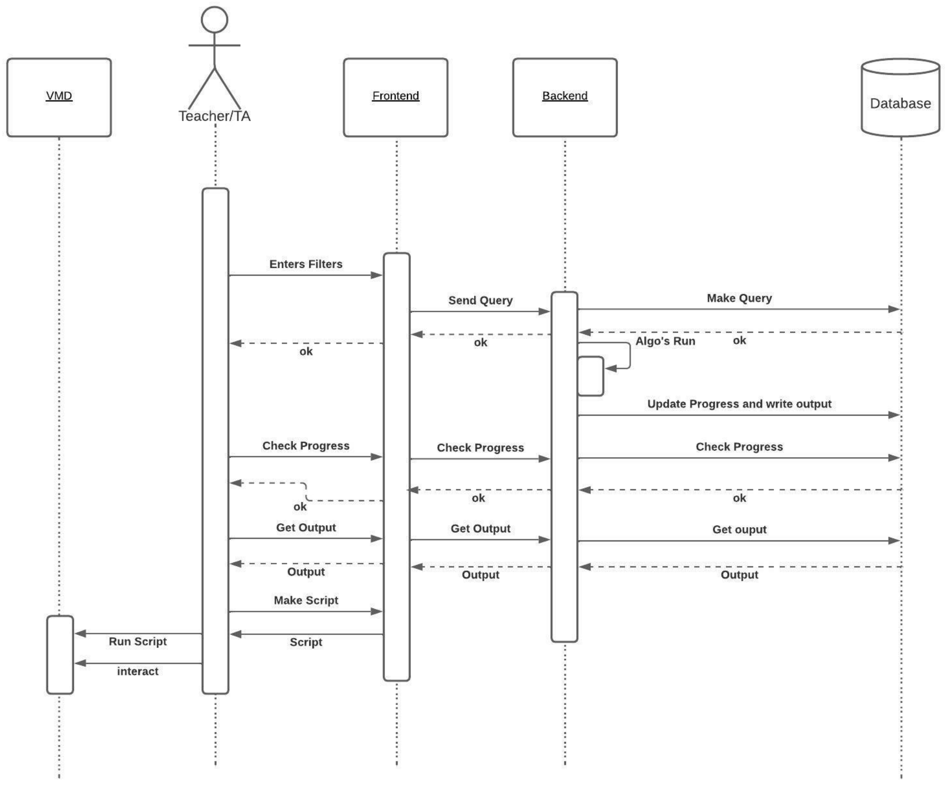


**Figure 5:** Sequence diagram of a user submitting a query, getting a result and making a script, then visualizing

### 3.5 Overall Changes

Our project is largely unchanged from its initial design, but there was one major change that did occur during development. Originally, the plan was to use React to embed VMD into the rest of the UI for visualization purposes, until we realized that VMD isn't able to be embedded into other software programs. As such, once we had discovered that VMD didn't have a library we could use, we changed our model so that instead of embedded VMD into React, our backend returns an output file to our frontend which can then be turned into a tcl script. This is then able to be used to separately visualize queries into VMD. Furthermore, we initially planned on using python for scripting, but changed that to .tcl as well. We also went from having a notification style system to include a progress page where users can track their progress on the algorithms running on the backend.

### 3.6 Security Concerns and Countermeasures

Due to this project being used as a tool by graduate students, security wasn't a primary concern of our application, however we do have some things in place that can provide some security. Firstly the application is running on a university VM meaning that only people inside of the university's domain will be able to access the website or access the VM itself even if they have the username and passwords associated with each of them. The application itself also asks for a username and password before allowing users to submit queries. With those in mind only users from the university will be using this running version of our application at the moment. If more people were to use it more authentication would most likely be needed to ensure no access to the API was being used for malicious purposes.

# 4 Implementation

### 4.1 Backend

The implementation of the backend was as planned. We used .Net which is written in C# to implement the API for the project. We wrote functionality to handle user account management, users uploading multiple files, or simulation data, and being able to run multiple queries on each of the uploaded files. To process the query, as it would take close to 30 minutes to run, we fire off a background task that works on the query and updates the status of the query over time. As this is a proof of concept, we currently only return a hard coded, valid convoy file, however it functions like it would in production with it only being "generated" after a certain period of time. For the sake of time, it is generated after approximately a minute.

The database was managed through Microsoft SQL Server Management Studio so the SQL statements used are slightly different to default SQL. The files uploaded are being stored in the file system with references to the files stored in the database.

We ran into some challenges on the VM while trying to get the code running and the database setup. Microsoft SQL Server Management Studio as well as Visual Studio were too large to run on the VM, however we were able to use tools that Microsoft provided in order to build and deploy the code. We were also able to create the database through the command prompt.

## 4.2 React.js

When it came to implementing the react application there wasn't really anything out of the normal that was used. It was created using the create-react-app command which added all the dependencies needed. The only extra packages that were really added was a dependency that allows for using a proxy to access the backend while avoiding some errors that come from CORS which is active in browsers for safety purposes. As for the application all of the main features of the application are split up into individual components that are housed in a components folder. Almost every component has a feature that can make a call to talk to the backend API discussed in 4.1. Due to the application being implemented on a ISU Vm the application can only currently be accessed while being on the ISU network or through the ISU VPN. Below are a couple examples of the react application including a create user page, and a couple other pages that can be accessed after logging into the application.

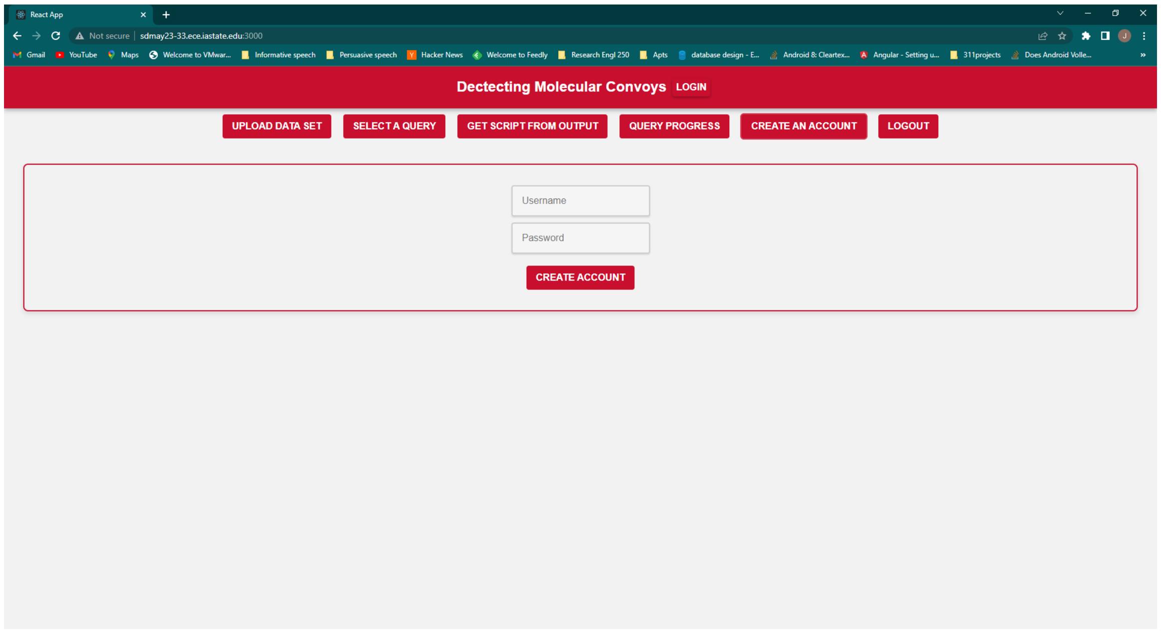


**Figure 6:** Create Account

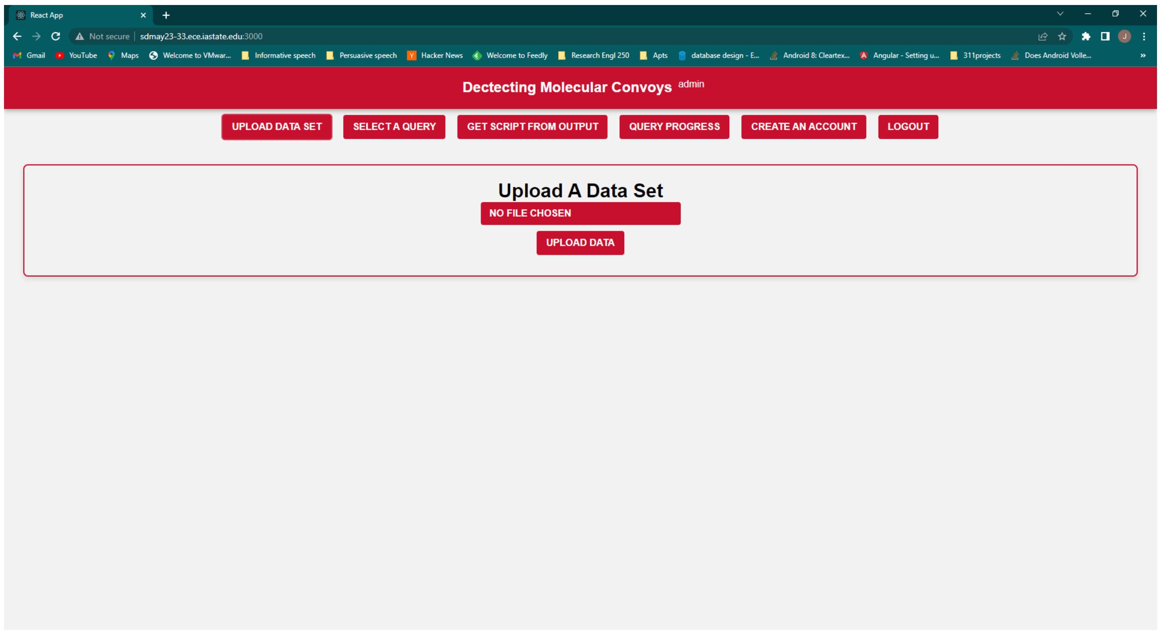


**Figure 7:** Upload Data

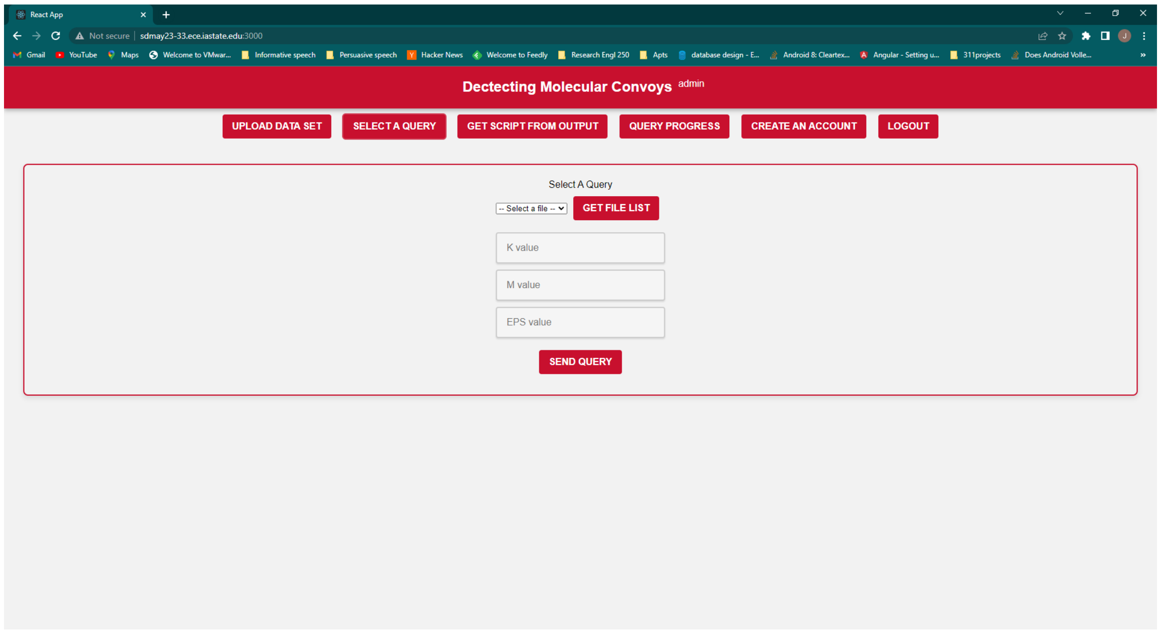
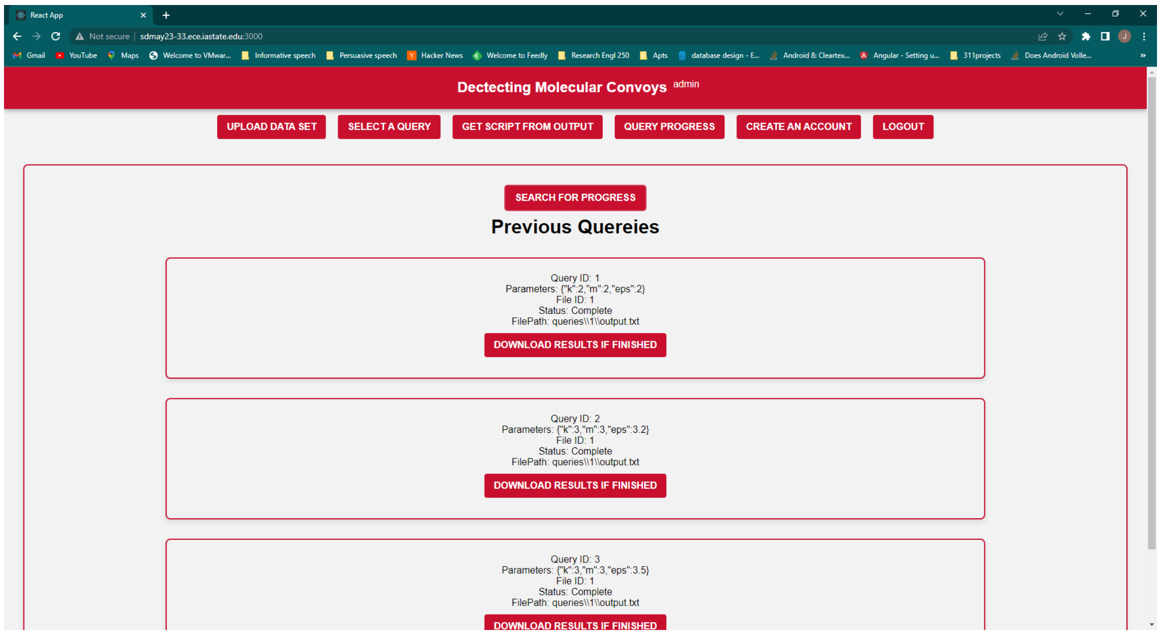


**Figure 8:** Create Query



Dectecting Molecular Convoys admin

Previous Quereies

Query ID: 1
Parameters: {"k":2,"m":2,"eps":2}
File ID: 1
Status: Complete
FilePath: queries\\1\\output.txt

Query ID: 2
Parameters: {"k":3,"m":3,"eps":3.2}
File ID: 1
Status: Complete
FilePath: queries\\1\\output.txt

Query ID: 3
Parameters: {"k":3,"m":3,"eps":3.5}
File ID: 1
Status: Complete
FilePath: queries\\1\\output.txt

**Figure 9:** Progress Page

## 4.3 VMD

When implementing VMD into our project we needed to first install VMD, and make sure that we could get it running with the provided simulation data from our clients, an example of this is below in figure .

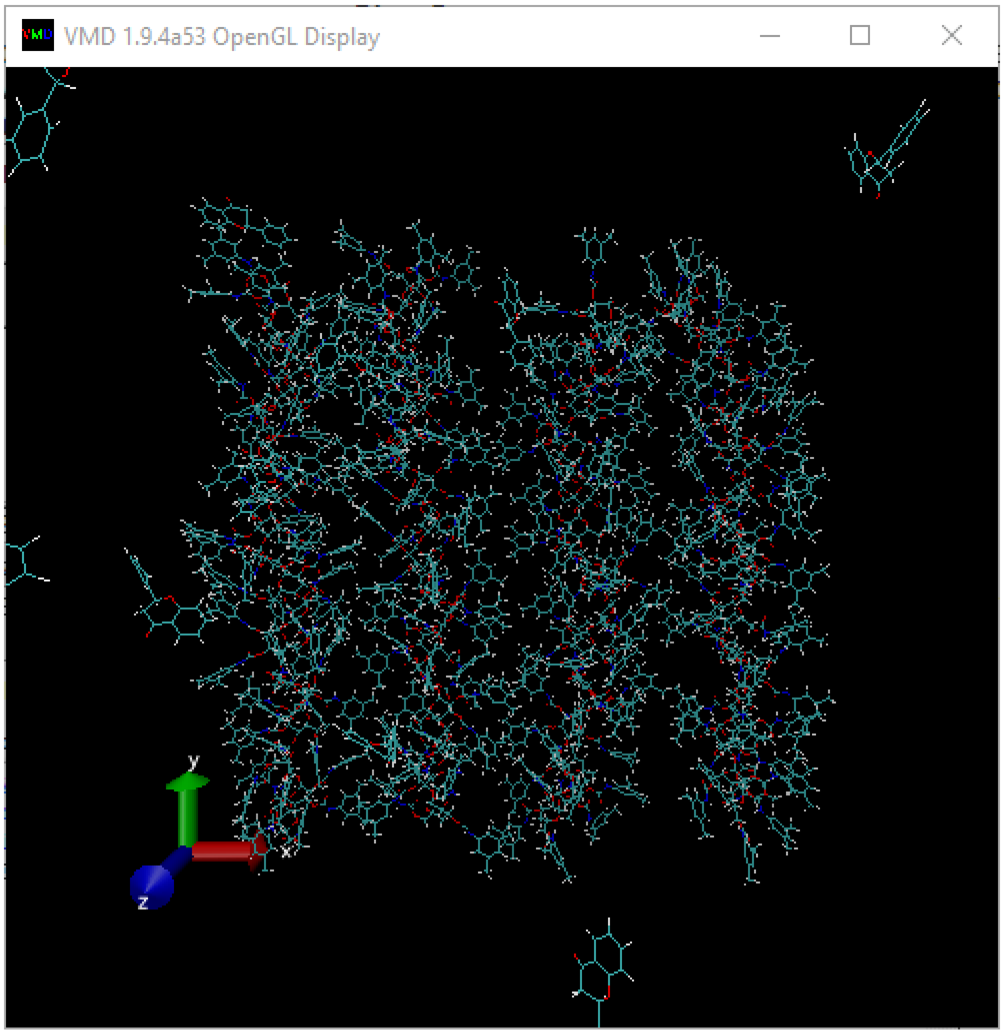


**Figure 10:** VMD Display

After that it was our object to learn about how we could use .tcl scripts to interact with molecules that we loaded into it. When interacting with the molecules our main goal was to be able to filter the molecule to only show data we wanted to see, which in this case is the convoys that are found from running the backend algorithm. Implementing this kind of feature took a lot of trial and error and reading through the user manuals of VMD to figure out their custom console commands. One feature that helped was a way where any command used through their interface would also show you the script version of the command meaning if we could perform the task manually we could also turn it into a script. Using this we were able to discover a script that would perform the task we wanted but that was the first part since then we need the script to be reflective of the output of the convoys provided by the backend algorithm and not just work for one case but all cases. We were able to make our React.js application take the outfile from the algorithm and make a script based on that individual output allowing us to filter the convoys out from any output file. With this we decided to make it so the script will open two separate molecules, one being the filtered and one being the unfiltered, allowing the user to swap between filtered and unfiltered views, a filtered example can be seen below in figure .

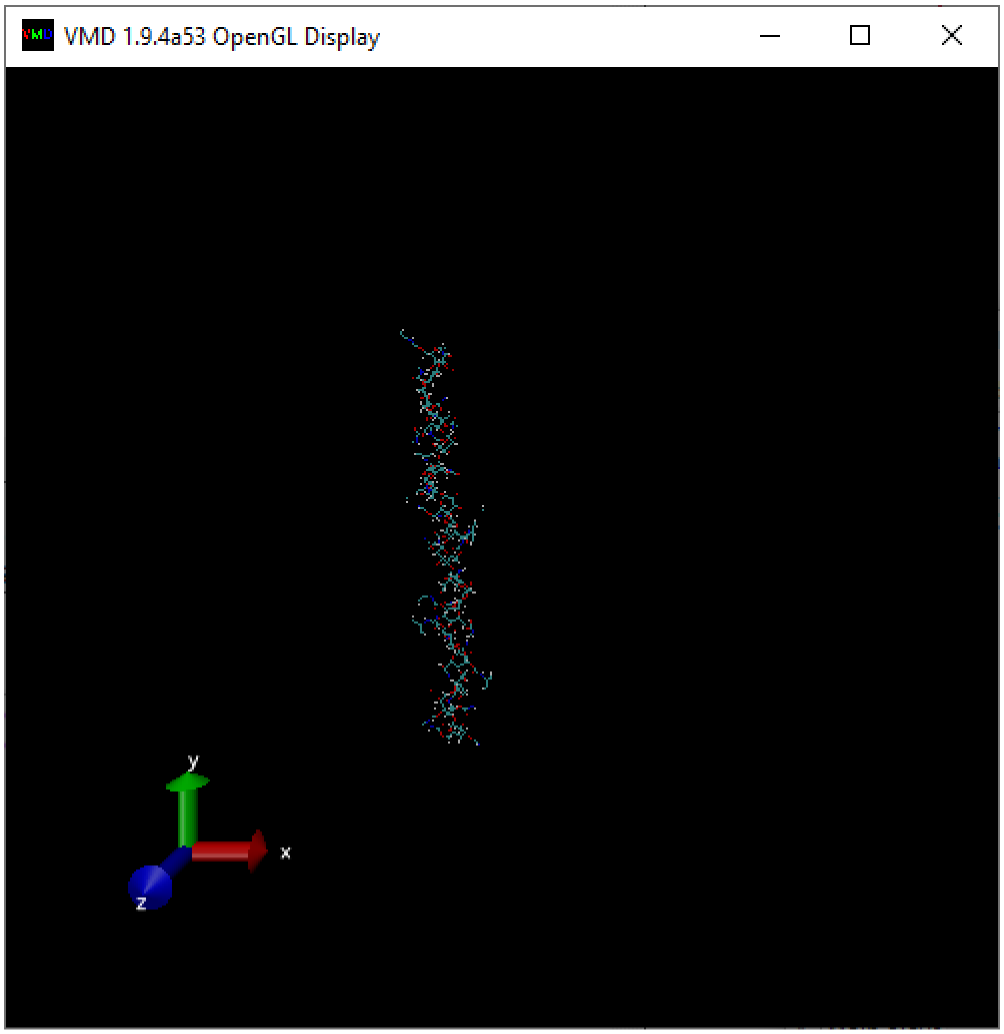
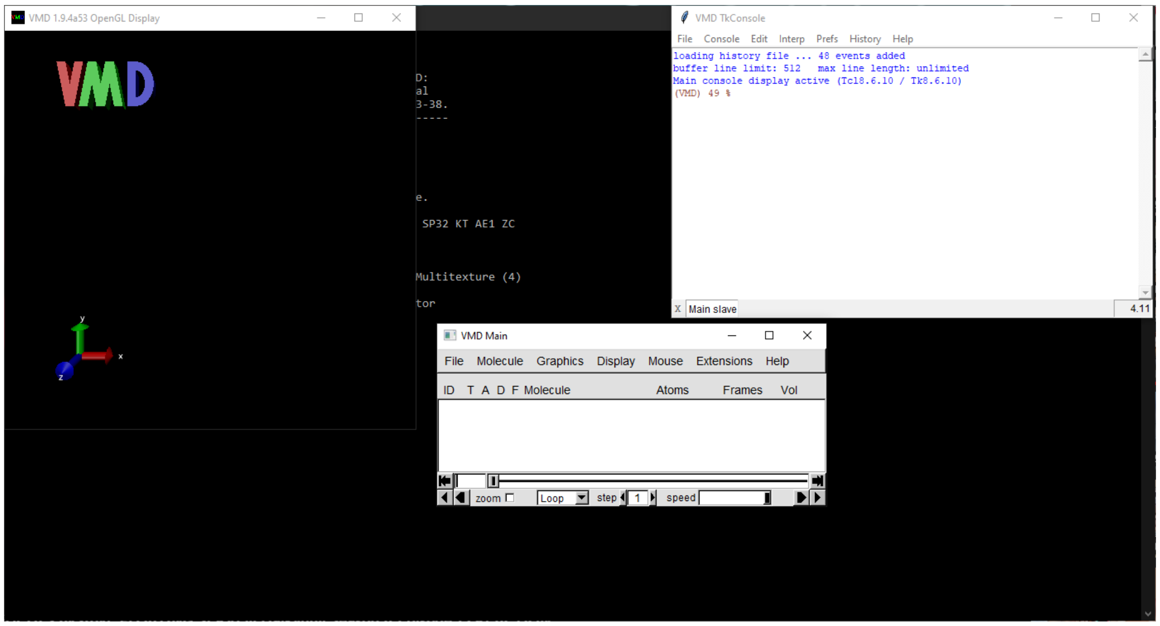


**Figure 11:** Filtered VMD

A full example of what VMD looks like when open and running with no molecules is shown below in figure . This shows their viewing window, the molecule console, and their command prompt. Directions on how to use the system will be provided later in this document.


VMD 1.9.4a53 OpenGL Display
VMD
VMD TkConsole
File Console Edit Interp Prefs History Help
Main slave
4.11
VMD Main
File Molecule Graphics Display Mouse Extensions Help
ID T A D F Molecule Atoms Frames Vol
zoom Loop step 1 speed

```
D:
al
3-38.
-----

e.

 SP32 KT AE1 ZC

Multitexture (4)

tor
```

```
loading history file ... 48 events added
buffer line limit: 512   max line length: unlimited
Main console display active (Tcl8.6.10 / Tk8.6.10)
(VMD) 49 %
```

**Figure 12:** VMD Example

## 4.4 VM

Both the React.js application and the API require a place to run and house them, we request a VM through the university for our project, currently the VM will be active through the end of the semester and will be left to be updated at the client's discretion. This VM is hosted on sdmay23-33.ece.iastate.edu allowing the React App to run on port 3000 and the api to run on port 7145. The SQL server is also housed on this VM and works with the API. Accessing the VM can be accomplished through ssh or RDP with a provided username and password if in the future any edits need to be made.

## 4.5 Testing

Testing was done locally, using the resources Jest and .Net for formal testing, and by hand for less formal testing cases. Jest is a testing framework for React that was employed on the frontend, while the other services were used on the backend. Unit testing provided evidence of each component working properly on its own, and integration testing concluded in late stages that components operated well together.

# 5 Conclusion

This project will be able to help automate and visualize some of the things our clients have been working on. This application is something that hasn't really been made before or at least used in their specific purpose so it could prove useful in their future research. This project could also be taken a step further by analyzing how the convoys are formed or what kind of bonds there are since our clients are currently looking specifically at hydrogen bonding which could be something the application could be adapted to try to cover. As for this project, it is more a proof of concept that demonstrates that this is an application that can be used as a tool during research to be able to visually display convoys for analysis and interaction.

# 6 Appendix I (Operation Manual)

## 6.1 Accessing The VM

When it comes to accessing the VM the easiest way to do it would be to use the windows application Remote Desktop Connection which can be installed from the windows store. This will allow you to login to the system like any other computer and have a visual display. The username and password will be passed on to our clients but when prompted about the username and password those will be entered and it should login to the VM.

## 6.2 Running the API and the React App

First we will talk about the API. First you want to open a command prompt and then change directories to “C/MolecularConvoys/Server/MolecularConvoysAPI”. If this is the first time this API is being ran on the machine you will want to use the command “dotnet build” but in this case it is already built on the VM. To run it you will use the command “dotnet run” and this should run the API and it should say it is listening. From there we will leave that command prompt alone and open a new one to run the React application. This time we will change directories “C/MolecularConvoys/Client/molecularconvoys”. From there again if it is the first time it is being ran you can run the command “npm run build” which will build it, if there are any packages missing those can be installed with “npm install ‘missing package name’”. On this VM all of it has been set up and the command “npm start” should start the application running on port 3000.

## 6.3 Accessing the Site and Functionality

To access the site after running it you have to be on the ISU network or using the ISU VPN and go to the url http://sdmay23-33.ece.iastate.edu:3000/. This will bring you to the initial page of upload data but you won’t be able to use anything until logging in except for one page which we will discuss later. The first page of interest will be the create user page.

### 6.3.1 Create User Page and Logging in

This page is very straightforward and will allow users to make a new account by putting a unique username and then a password. The system will let the user know if it has made the account or if there was an issue with an alert. A picture of the page can be seen below.

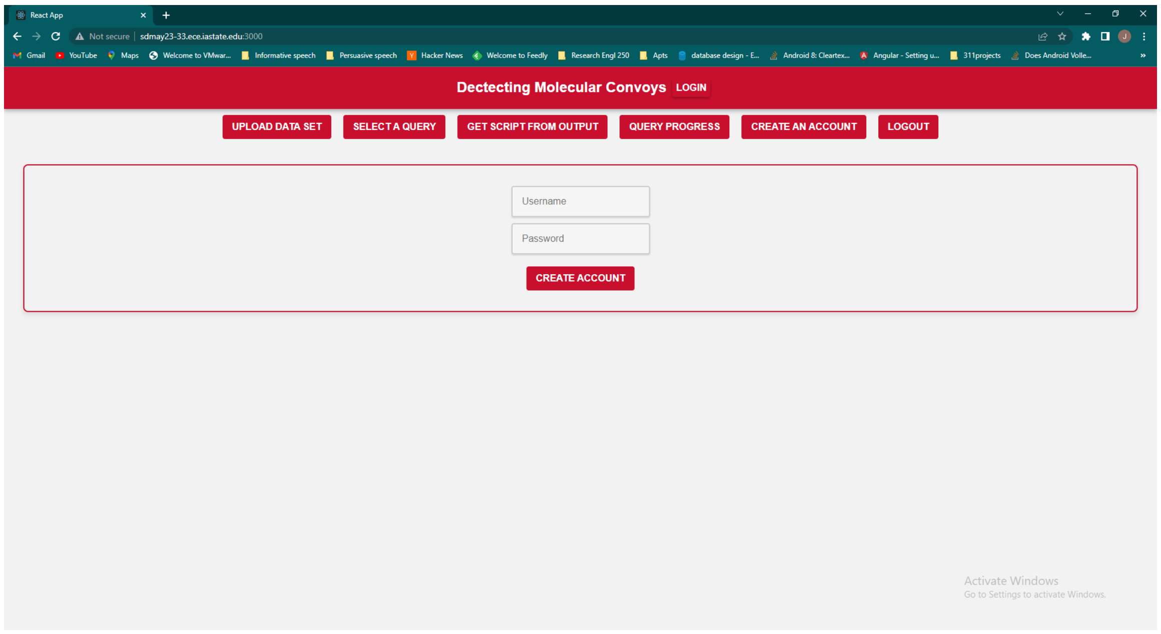


**Figure 13:** Create Account

After a user has made their account they can login using the login button and entering their credentials which will tell them the result in an alert as well.

### 6.3.2 Data Upload

This is the next page that will be covered and this is where a user will upload their .npy file that they would like to analyze. This page just has an upload and a submit button. When uploading this process can take time so we urge the user to be patient as depending on the size of the file and the connection it could take a while. An alert will let the user know if there is an error or if it got uploaded so the user should wait until either of those have happened.

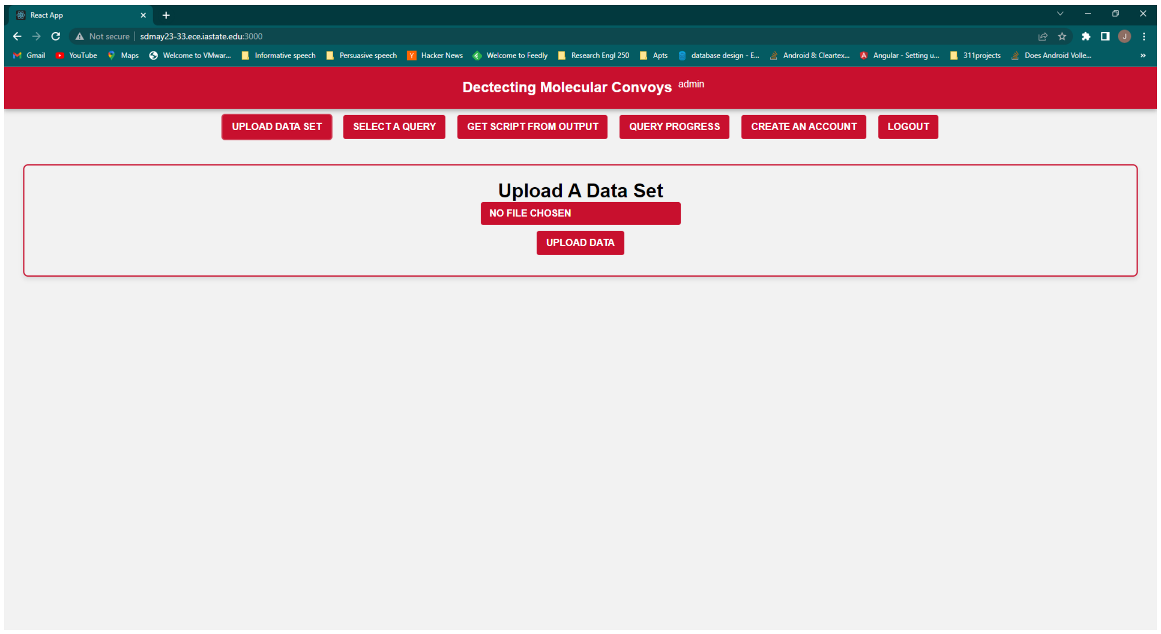


**Figure 14:** Upload Data

### 6.3.3 Creating a Query and Query Progress

The Create Query page is next and this will allow the users to start the execution of the backend code on their chosen file. When a user goes to that page they will hit 'get file list' to refresh the file list and they should be able to see their uploaded files and select one. From there they can enter a K value which is the "Min number of consecutive timestamps to be considered a convoy" and a M value which is the "Min number of elements to be considered a convoy" as well as an EPS value which has to do with how the backend code is run. The user can then hit 'send query' and should get a response in the form of an alert, the page is shown below.

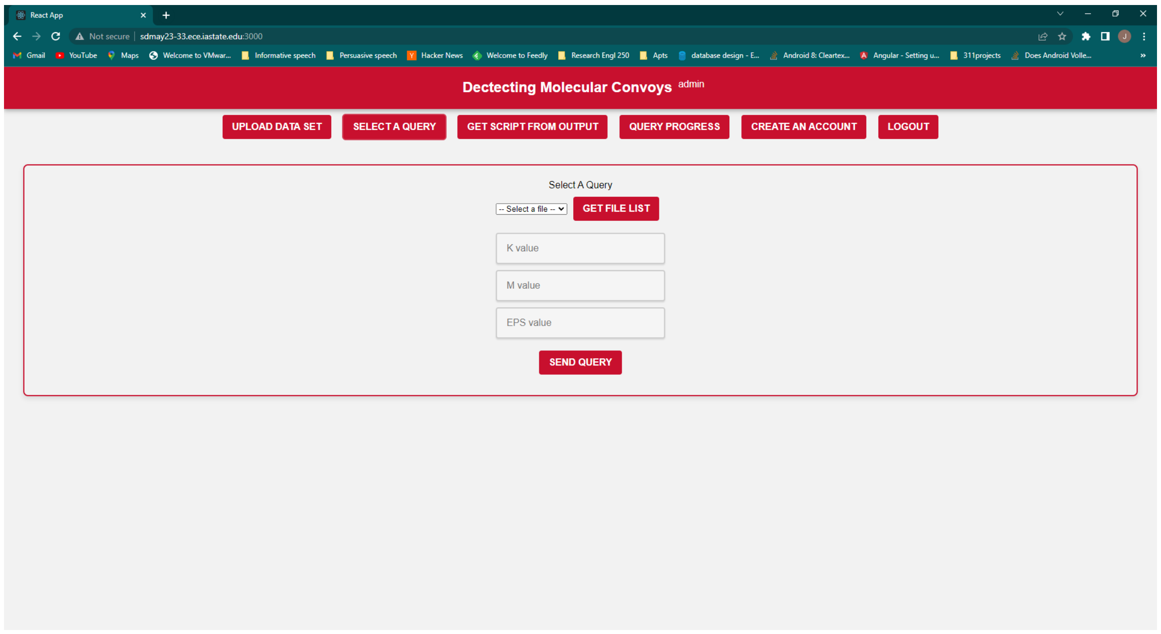
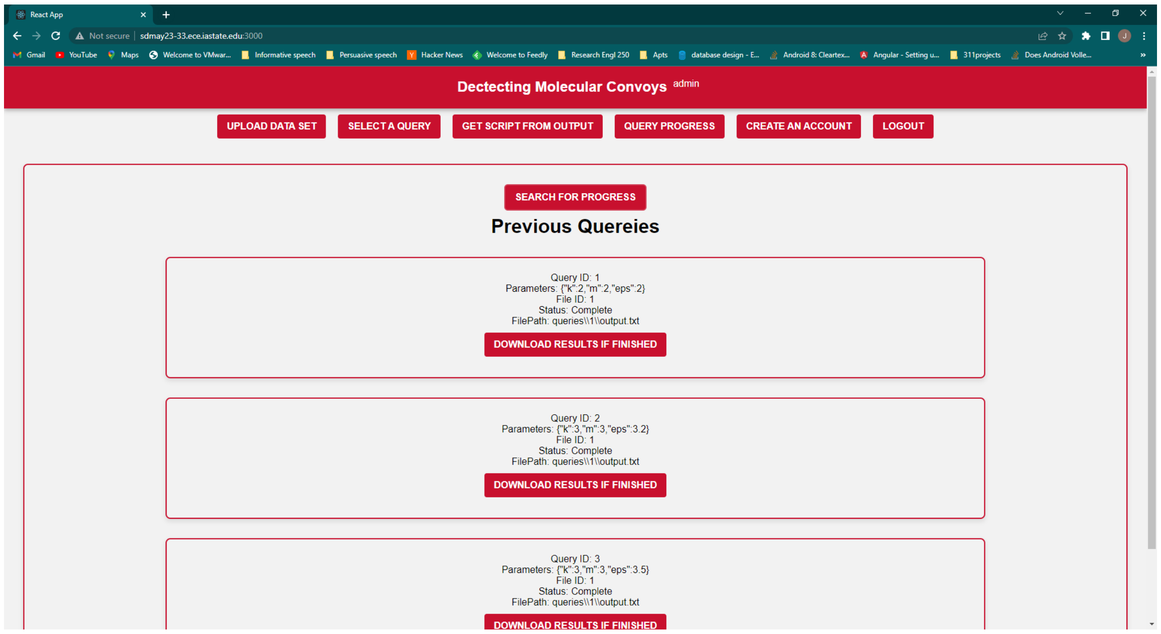


**Figure 15:** Make Query

Next you can go to the query progress page and hit the Search for Progress button to then see the progress of those queries. In here you can see that there are three stages we check to show the progress of the query as it goes through the algorithm. From here when it is finished you can also download the output file in the form of a .txt file. The page is shown below.

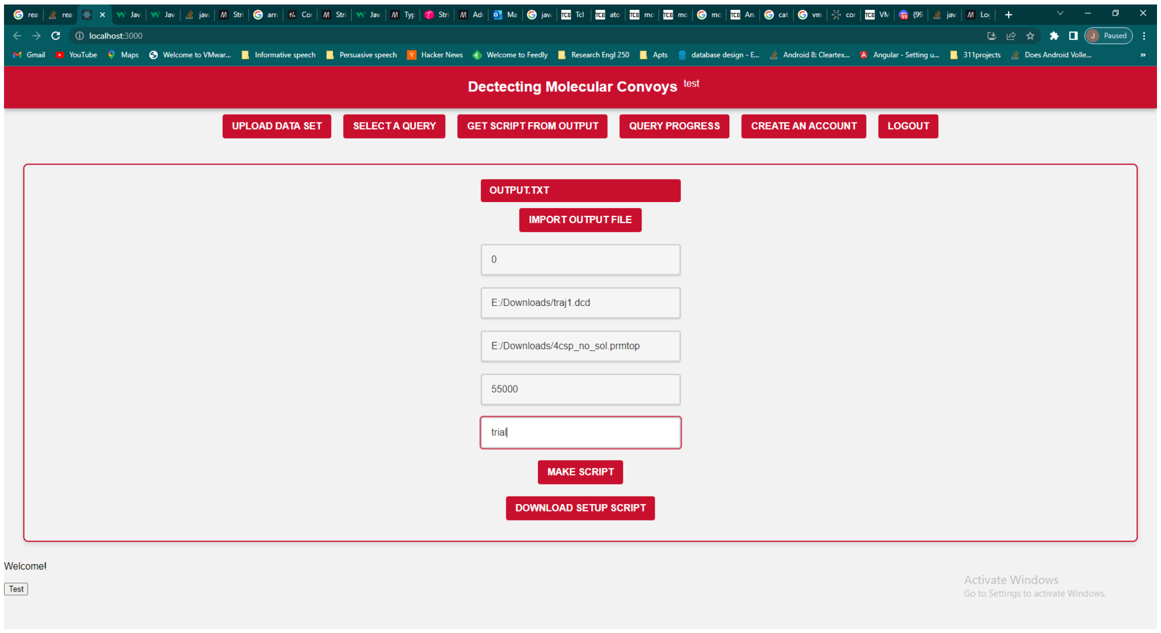

React App
Not secure | sdmay23-33.ece.iastate.edu:3000
Dectecting Molecular Convoys admin
UPLOAD DATA SET
SELECT A QUERY
GET SCRIPT FROM OUTPUT
QUERY PROGRESS
CREATE AN ACCOUNT
LOGOUT
SEARCH FOR PROGRESS
Previous Quereies
DOWNLOAD RESULTS IF FINISHED
DOWNLOAD RESULTS IF FINISHED
DOWNLOAD RESULTS IF FINISHED

```
Query ID: 1
Parameters: {"k":2,"m":2,"eps":2}
File ID: 1
Status: Complete
FilePath: queries\\1\\output.txt

Query ID: 2
Parameters: {"k":3,"m":3,"eps":3.2}
File ID: 1
Status: Complete
FilePath: queries\\1\\output.txt

Query ID: 3
Parameters: {"k":3,"m":3,"eps":3.5}
File ID: 1
Status: Complete
FilePath: queries\\1\\output.txt
```

**Figure 16:** Progress

### 6.3.4 Making the Script

This is the final page and is the one that uses the output provided by the query progress page. The first step is uploading that output and hitting the import button. Then from there you can select which convoy you would like to look at in that prompt. Next you will need to provide the file path to the .dcd which is the simulation data and then the .prmtop file which tells VMD made the query from which should be part of the name of the .npy and then name the output file something like "molecule x" or whatever you want. Then you can hit the 'make script' button. If this is your first time using the application you should also download the setup script by hitting download setup script. This page is the only page that can be accessed without logging in since to use this page you only need to have an output so if you already have an output and would like to make a script to visualize it you can. A picture is shown below of the page and this page also contains the directions to run the scripts after downloading them.



0
E:/Downloads/traj1.dcd
E:/Downloads/4csp_no_sol.prmtop
55000
trial

**Figure 17:** Make Script

## 6.4 VMD

VMD can be installed from "https://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=VMD" and once installed can be run like any other application. Once you receive a script you can run it by following the steps provided on the make a script page of the application but it boils down to opening the console, changing the directory to the place where the scripts are, running the setup script with "source setup.tcl" and then running the downloaded script the same way with "source 'name'.tcl". More interactive details are provided on that page along with pictures if there is any confusion.

# 7 Appendix II (Other Versions and Design Considerations)

## 7.1 Graphing Method

Our initial designs had the idea of using other graphing methods that would allow us to use a javascript library rather than an application, this would allow us to embed it in the application easier but would also make it much more difficult to translate the simulation data into data that could be graphed accurately. Some of the options we considered heavily were three.js, Aframe, and babylon.js. This changed when our clients said we should use VMD which is what they already used in the past to graph them as their data file goes right into it and it would handle the graphing. This would allow us to focus on the other parts of the project rather than remaking something that already exists and just filter the data as needed.

## 7.2 Embedded VMD

Another change we had was the idea that the initial plan was to embed VMD into our application with a window to display the graph for the user. This idea quickly changed when we realized that VMD doesn't have a library that works with javascript and you aren't able to embed an application into a site especially if there are multiple windows that would need to be streamed if we wanted it to somehow still display from the application. We decided to take an approach that made a script that could then be run in VMD as a separate application which ended up being a working solution to allow us to still use VMD and get close to the same result. An example of the old design is shown below and looks very similar to the current one.

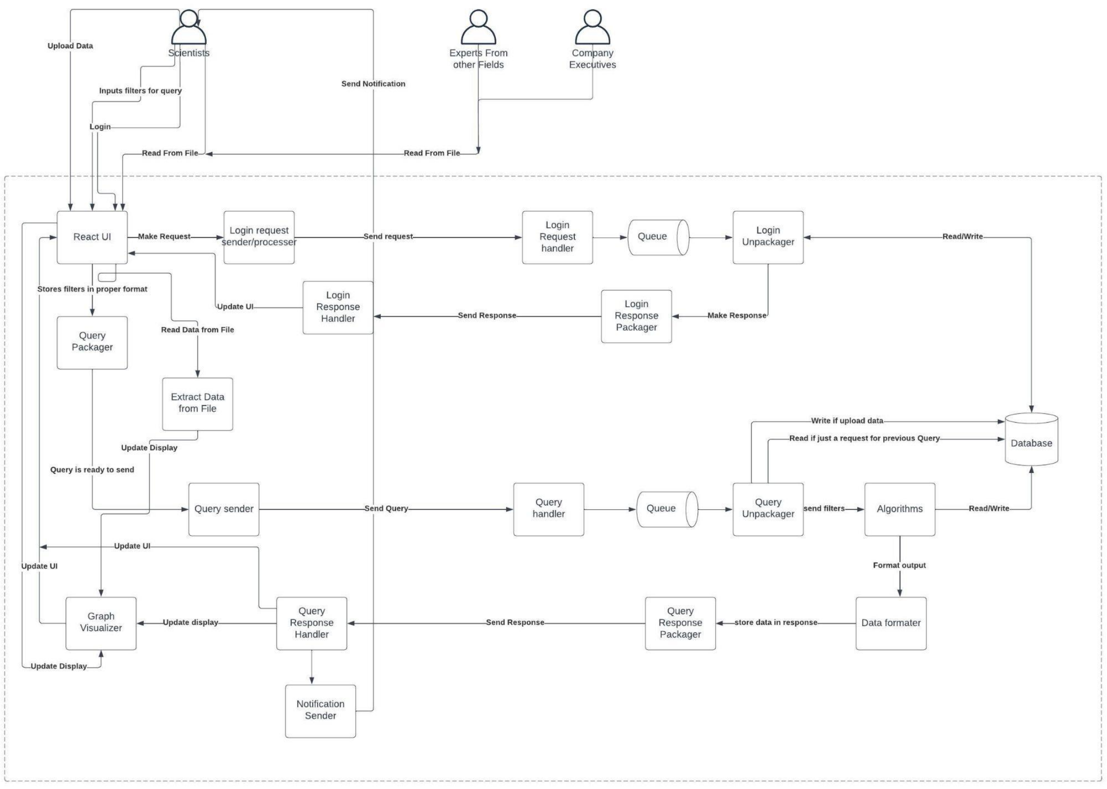


**Figure 18:** Old Component Design

## 8 Appendix III (Code)

The source code is available at https://git.ece.iastate.edu/sd/sdmay23-33 and with to operation manual could be ran on a local machine provided the correct systems such as javascript, Microsoft SQL server and C# packages are installed.