Virus Database Final Report

By Matt Heffel, Sam Moylan, Mitchell Slavens, Lev Kavs

I. Overview

Our project is a biological virus database. The application is based around a database which contains various species of biological viruses. Multiple strains for each species of virus including each strain's entire sequenced genome, open reading frames on the strain's genome, and the proteins that each open reading frame encodes. The open reading frames are then further categorized into either structural or non-structural proteins. The database also includes publications for which each strain was published in or used as a citation along with the publisher and the researchers who contributed to each publication. The applications has multiple capabilities to display, organize and manipulate the database which we will discuss in further detail below. Our target users are students, researchers, teachers and anyone who is interested in studying and exploring viruses and their properties.

II. Design

A. Technical Description

The core elements in the database are the species of viruses which have the properties of: family, order, and genus that they belong to, as well as their name and abbreviation. Each species of virus has multiple strains, though each strain is limited to being a part of exactly one species. Strains are instances of a species of virus the are genetically diverse though still belonging to that species. Each strain contains the information of its entire sequenced genome, the genome's length which is referred to as a base pair, and the species which it belongs to. Open reading frames are shared among the strains of a species, though they may, and are likely to be genetically diverse from the same open reading frame within a different strain of the species. Open reading frames contain information on their start and stop index which reference a segment of the genome of the strain that the open reading frame is a part of. They also reference a protein that they code for. An open reading frame may only code for one protein, though if the database were expanded outside of just viruses this may not always hold true. There may be multiple different open reading frame codes for the same protein. Proteins are categorized into either structural or non-structural regarding whether they are a part of the virus's structure or if they serve a catalytic function. An important thing to know as if one is aware of the structural proteins that a virus is composed of. One may use that knowledge to recognize antigens that can be targeted as identifiers for the virus and use those to develop a vaccine that allows the body to produce antibodies that will bind to the virus in an immune response. Publications are linked to strains that are described within a publication. They may also be cited, which will contain the information of their title and the year published and are also linked to researchers that contributed to them and the publisher who published them. Researchers contain information of their name, email, and organization. Publishers contain information on their name.

Due to the great diversity among the existing virus species in the world we have limited the range of virus species to only those within the Order Nidovirales which is quite a diverse group itself including all of the Families that are within Nidovirales and the Genus' within those Families. On a larger scale for what the application's primary functions are capable of, the database could be expanded to support not only every known virus but also any species of life that has a sequenced genome, or even things that

are not considered as life such as prions, viroids, or free forming RNAs. Essentially anything that has a DNA, RNA, or amino acid sequence is compatible with the database. I will remark that if eukaryotic organisms were to be entered into the database, while the functions of the application will still perform correctly, the complexity of introns in eukaryotic DNA may cause output data to be obscured in some instances.

B. Features

The application contains useful functions for a user to be able to search the database for specific species, strains, open reading frames, proteins, researchers, publishers, and publications. As the user selects items from a populated list, query requests are executed on the database to retrieve the data for populating further list boxes. On startup the leftmost listbox is populated with the names of all species in the database. Upon selecting a species the following strains listbox will be populated with only the strains that belong to the selected species. The user will then have to select from from the three options in the dropdown to populate the following listbox. The process of selecting an item in a listbox is then repeated. If the user would like to revert a listbox selection they have made, they select and item from a previous listbox and all the listboxes following the new selection will be cleared and repopulated if possible. Through the selection process, the user may at any time chose to click the display information button to see all the details of the most recent selection they have made. The query used to gather the information from the database to populate the listbox already contains all the necessary information needed and a new query request is not required.

The user may also chose to delete an item at any point during their search process. This will delete the most recently selected item from the listbox display as well as the database itself. Through the database, if an item is to be deleted that has items in other tables referencing it, a CASCADE ON DELETE property has been added to all foreign keys to ease the deletion process. Since this is a database for researchers as well as students of viruses we are giving the user total control and allow them to remove items from the database as they please.

In addition to deleting data, the user may also choose to edit or add data. If the user selects to edit using the edit button, the rightmost selected item in the display is the item being edited. A new window is shows, containing textboxes for the user to enter new data. To avoid complications, the user does not have the ability to edit IDs of different different items (speciesID, proteinID, researcherID, orfID, etc.). They only have the ability to edit properties such as researcher names, publication titles, etc, attributes that will not complicate relationships between tables as changing the IDs would. The user can add a new Species if they would like. They can also add a new Strains to a specific species. These can be added either by entering all of the information for a publication (and therefore a publication will be added to the database as well) or they can simply enter a strain with no publication information. They can then add proteins and ORFs to any given Strain.

The two most exciting features of the database application that make it stand out are the alignment and consensus sequence functions. The alignment function allows the user to select two different open reading frames from any two strains within any two species in the database, though aligning two completely unrelated open reading frames may not be useful, it is possible to do, and perform a pairwise sequence alignment on

them. The pairwise sequence alignment automatically saves the alignment output to a text file and opens the file which contains the two aligned sequences, each labeled with their species name, strainID, and orfID, as well as a third annotation component that shows where insertion and deletions, labeled by '-', as well as mismatches, labeled by 'X', between the two sequences. This is an incredibly powerful tool as knowing the differences between two genetic sequences is the key to understanding why they are being expressed and behaving the way that they do. It is important to note that while sequence alignment is possible for genomes of any length, the longer the genome gets the more computationally heavy the alignment algorithm is as well as the more RAM that it uses. The typical personal computer generally has enough RAM to handle genomes of about 16000 base pairs in length being aligned.

The consensus sequence function is capable of generating, as the name suggests, a consensus sequence for either a species as a whole, or a specific open reading frame within a species. The user is able to choose a species and from there can either generate a genome wide consensus sequence or further choose an open reading frame within that species and generates a consensus sequence for specifically that open reading frame within the species. The particular consensus sequences being generated are those for translation initiation sites, 10 nucleotides upstream and 10 nucleotides downstream from all of the species' open reading frame start sites or the start site for the specific open reading frame that was selected. The consensus sequence is generated by taking all of the strains in the species and then all of the start indexes in the open reading frames or chosen open reading frame in that species and looks at each position in the 20 nucleotide sequence surrounding each start index and determines the percent probability that a given nucleotide, adenine, thymine, guanine, or cytosine, will appear in that position. It then takes the nucleotide with the highest percentage for each position for the consensus sequence. As in the alignment function, the application automatically saves the output to a text file and opens it showing the output which includes the nucleotide percent breakdown for each position as well as the developed consensus sequence. This feature could potentially be further developed into an open reading frame finder for a user who is entering a strain into the database that is of a species that is already existing in the database where the developed consensus sequence or sequences would be used to predict where the known open reading frames for that species are in they newly inserted strain's unannotated genome.

C. Database Schema and E/RDiagram

Proteins (pID, pName)
- -pName cannot be null

StructuralProteins(pID, struct)
- -pIDis a foreign key referencing pID in Proteins
- -struct is not null

NonstructuralProteins(pID, funct)
- -pID is a foreign key referencing pID in Proteins
- -funct is not null

Species(specID, sName, sAbbreviation, sGenus, sFamily, sOrder)
- -sName is unique and not null
- -sGenus is not null
- -sFamily is not null
- -sOrder is not null

Strains(strainID, basePairs, specID, genome)
- -specID is a foreign key referencing specID in Species
- -genome is not null

OpenReadingFrames(strainID, orfID, pID, startIndex, stopIndex)
- -strainID is a foreign key referencing strainID in Strains
- -orfID is not null
- -pID is a foreign key referencing pID in Proteins
- -startIndex is not null
- -stopIndex is not null

Publications(pubID, pYear, title)

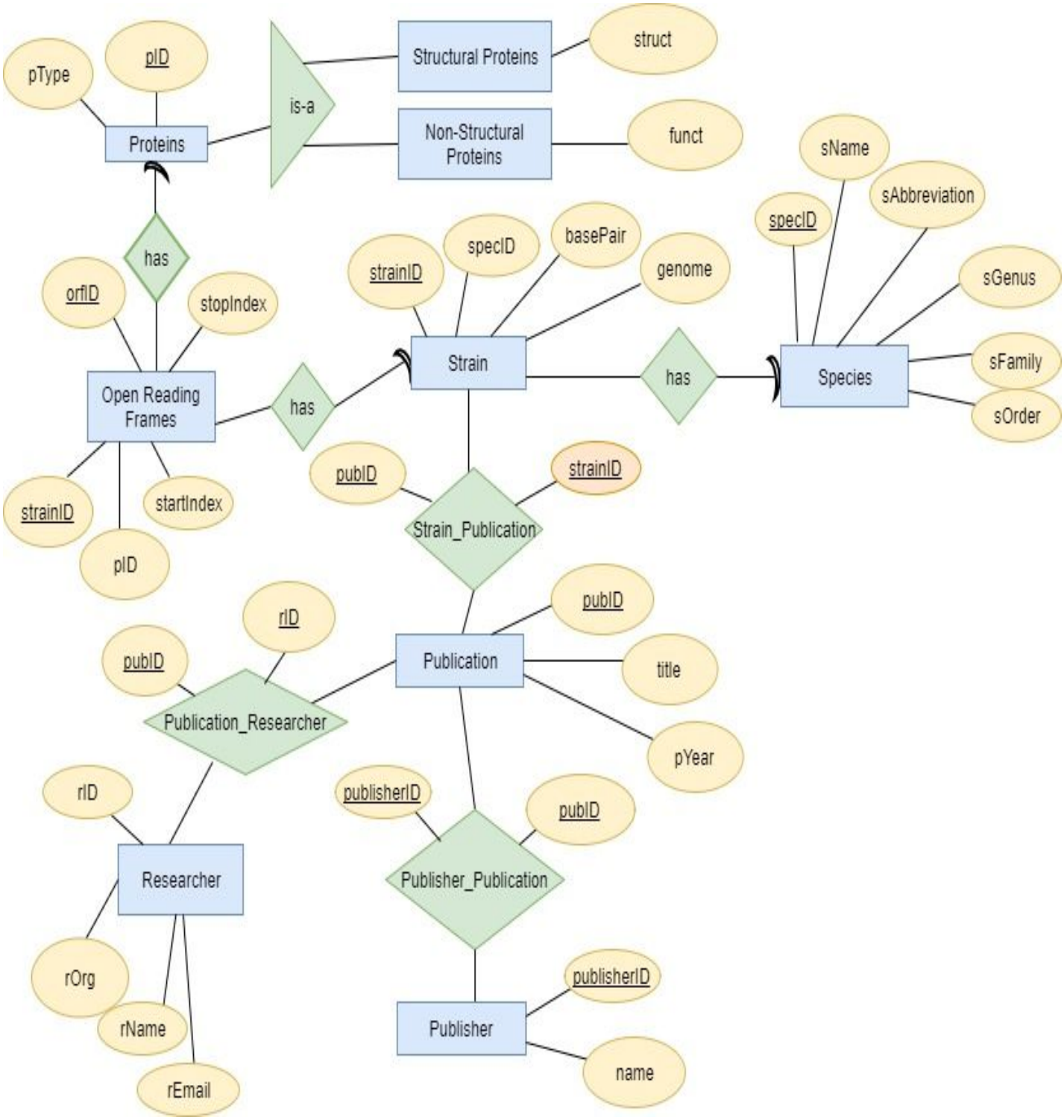
Researchers(rID, rEmail, rName, rOrg)
- -email is unique and not null
- -name is not null

Publication_Researcher(pubID, rID)
- -pubID is a foreign key referencing pubID in Publications
- -rID is a foreign key referencing rID in Researchers

Publisher(publisherID, name)
- -name is not null

Publisher_Publicaition(publisherID, pubID)
- -publisherID is a foreign key referencing publisherID in Publishers
- -pubID is a foreign key referencing pubID in Publication

Strain_Publication(pubID, strainID)
- -pubID is a foreign key referencing pubID in Publication
- -strainID is a foreign key referencing strainID in Strains


pType
pID
Proteins
is-a
Structural Proteins
struct
Non-Structural Proteins
funct
has
orfID
stopIndex
Open Reading Frames
has
strainID
specID
basePair
genome
Strain
has
specID
sName
sAbbreviation
sGenus
Species
sFamily
sOrder
strainID
startIndex
pID
pubID
strainID
Strain_Publication
rID
pubID
Publication_Researcher
pubID
Publication
title
pYear
rID
Researcher
rOrg
rName
rEmail
publisherID
pubID
Publisher_Publication
publisherID
Publisher
name

III. Database Implementation:

All files for database implementation are included in the folder titled Data. This includes table creation file, files used to populate the database, script used to gather the data, as well as any other files used to insert, generate, or update data.

IV. System Implementation

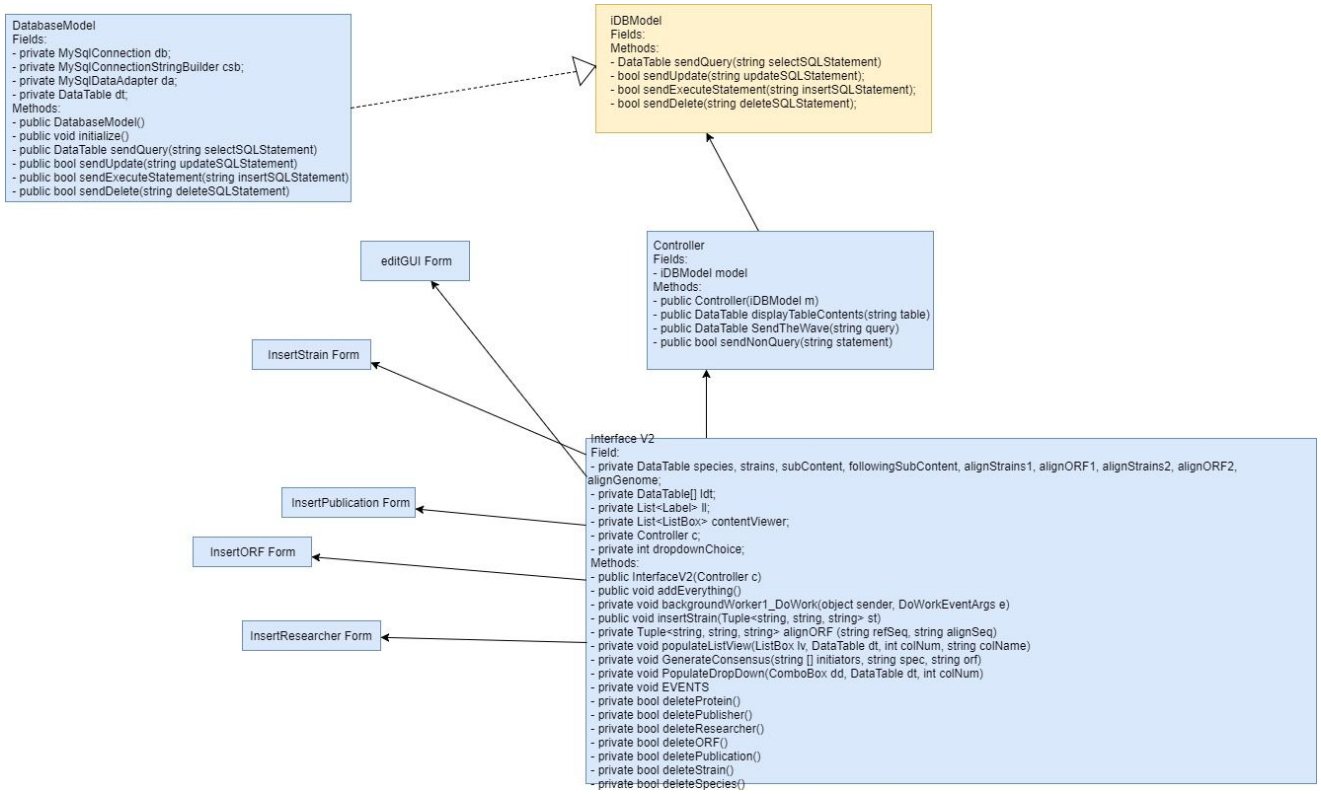
We modeled our software architecture by the Model View Controller(MVC) design. The three primary classes are Database Model, Controller, and Interface V2.

The Database Model class is used to connect to the MySQL Database using .NET/Connector. It maintains an open connection to the database throughout the duration of the applications life. It has methods which execute queries passed to it from the Controller. It returns either a DataTable object containing the returned query result which it received from the MySQL database or it returns a boolean if it executed a transaction. The boolean is true if atleast one row was affected by the transaction, otherwise it is false. The Database Model implements the iDBModel Interface which contains the four methods used to execute queries and transactions.

The Controller class on creation takes an iDBModel object in its constructor. It is given a Database Model, but it cannot access other methods or fields of the Database Model because it only knows the methods of the iDBModel. This provides security and the ability to create other classes that implement iDBModel, which the Controller could use. The Controller sends SQL statements to the Database model and what it returns from the Database Model it gives to InterfaceV2 Form.

InterfaceV2 Form is the main GUI/Window Form which is seen when the application starts up. The InterfaceV2 form owns/contains a Controller, which is given to its constructor. From the user interactions it can make the requests and gives them to the Controller to execute its requests. Then from the Controller it receives the updates it needs to display on the Form. Interface V2 incorporates and runs multiple other helper forms which it uses to get user input. It runs and displays different helper forms from the different interactions the user performs on the InterfaceV2 Form. The five helper forms which it uses are: editGUI, insertStrain, insertORF, insertPublication, insertResearcher. The editGUI is used for editing pre-existing data in the model that the user selects from the listboxes. The other four insert forms are used to gather the information to perform the insert that the user would like to make into the database. Below is the diagram of the architecture of our software application.

The flow of the application goes as such. After the Database Model connects successfully to the MySQL database and the InterfaceV2 Form is created the form immediately requests the Species table. The form calls a method from the Controller and the Controller calls a method from the iDBModel(DatabaseModel) which then executes what was passed through each object. The iDBModel(Database Model) returns to the Controller the output which the Controller then returns to the InterfaceV2 Form to display. Every user interaction follows this order when processing user interaction.



V. System Features

All features described below come with screenshots included in the "project_screenshots" folder. Titles of the images are as they appear in the subsequent paragraphs.

When the application loads the main GUI is loaded.This populates the appropriate lisboxes based on which items are selected in the previous listboxes. A drop-down box is present in the middle of the form that decides data to populate the last two listboxes with, mainGUI.PNG conveys this process. In the screenshot, a species is selected, followed by a strain. Once an item is selected in the second box, the drop down is enabled. In the provided example, OpenReadingFrames is selected from the, subsequently displaying the proteins for the selected ORF in the last listbox. The contents of the listboxes change depending on the selected items, but the functionality remains the same.

- The display details button functionality is shown in displayDetails.PNG. It displays the details of the most recently suggested item, as the name suggests.
- The edit button edits the most recently selected item. When the button is pressed, an edit GUI window is displayed. The textboxes are populated with the data from the most recently selected item in the main GUI. This can be seen in editGUI.PNG. The delete button deletes the most recently selected item from the application display as well as from the database. Nothing is shown for this process other than a message box so a screenshot is not included for this functionality.

Once the Insert tab is selected, the main GUI changes display. The user can choose to add a species, strains, ORFs, proteins, publications, or researchers. The insert tab can be seen in insertTab.PNG.

- If the Add Species button is clicked, a new GUI is opened prompting the user to add a new species. Shown in insertSpecies.PNG
- If Species drop-down is selected, user can select Add Strain. Upon button click, new GUI is opened to add strain. Shown in insertStrain.PNG.
- If Species and Strain drop-downs are selected, user can selected Add ORFS/Proteins. When clicked, new GUI is opened. Shown in insertORF.PNG.
- If Species and Strain drop-downs are selected, user is able to add a publication. Within the new GUI that is opened upon button click, user can add mew publication. In addition, user can add a researcher, a publisher, and/or a strain. These are all associated with the publication. Shown in insertPublication.PNG.

When the align tab is selected, the main GUI switches displays. The align functionality is shown above, so it will not be alaborated on here. User populates all necessary drop-downs and clicks align button. A text file is created with the alignment that the user is prompted to save. Screenshot of the Align tab is shown in align.PNG. The text file for this will be included in the folder titled generated_data as AlignedORFs.txt..

.When the Consensus tab is selected, user is prompted to perform a species consensus or an ORF consensus, based on which button is selected. A text file is generated and the user is prompted to save it. A species consensus will be included in the generated_data folder titled Consenus1.txt. Consensus2.txt will show an ORF consensus. The Consensus tab display is shown in consensusTab.PNG.

Overall, the system is very effective in terms of its core functionalities: the alignment and consensus features. These are very practical applications for someone involved in research. The system does have limitations, however. The user does not have complete control due to the difficulty that would cause in implementing the app. For example, the user can only edit non-crucial data such as names, years, titles, etc. The system does not give the user the ability to edit IDs of different items as this would create major problems in database implementation. In addition to this, the application is limited to the features mentioned above. While these are very flexible, more than you would expect for an application of this type, it is somewhat structured.

VI. Queries and Reports

Sample queries along with screenshots of their results can be seen in Queries.PDF.

VII. Evaluation

The application was successfully implemented with all the necessary requirements and constraints. The overall esthetics of the GUI/Form is simple and easy to maneuver if the user is familiar with viruses, species, strains, ORFs, etc. First time users only hiccup they may experience would be when starting to explore and familiarise themselves with the application. When the user is interacting with the search, edit, and add functionalities they must have the data they need to input on hand, which may be a bit off a hassle to collect, but that goes with any database. The performance of the application run smoothly and only takes a few seconds to execute the alignment and

consensus sequence functions if they are executed on large amounts of data. The wait time of the two functions could be drastically reduced if the application was ran on a more powerful machine instead of our laptops.

VIII. Summary

In conclusion, our project followed our initial design relatively closely. In planning, we knew we wanted to implement the alignment and consensus functions for sure, these are the key features of the entire application. Minor improvements were also made to our schema. The three members who were not well-versed in biological terms and properties of viruses were able to pick up a decent amount of information on the topic, at the very least learning the terminology. We learned how to work together, both remotely and in person. If we had a second go at the project, we would like to spend more time actually working on the project together rather than remotely. Working remotely did not cause any major problems but it was much easier to solve problems while the team was together. Future development for the application could include making the functionality more flexible. Our processes are currently relatively structured as far as what the user can do. There is a wide variety of functionalities, but little flexibility within these application. Further developments in flexibility could prove to make the application even more useful for researchers. Other species of viruses as well making the application have more than just viruses. The development could include everything with a genome. This would require a major change to the database as well as the application itself. But with the iDBModel, the Controller would not have to change, it would only have to be passed a new object which implements that interface. This could be creating multiple databases for multiple different variations.

IX. Team Work Experience

Overall, our team worked very well together. As was mentioned above, we did a lot of coding remotely. This complicated problem-solving but we were able to manage. Given that most a large portion of work in the industry is done remotely, this project was a very good learning experience and improved our skills for the future.