HELM Editor 2.0 Final Report

CSCI E-599 Software Engineering Capstone Project

Harvard Extension School

May 2016

By

Simili Abhilash
Stephanie Dube
Thankam Girija
Sarah Leinicke
Chinedu Okongwu
Justin Sanford

Contents

1. Introduction

1.1. Abstract

The Hierarchical Editing Language for Macromolecules (HELM), enables the representation of a wide range of biomolecules through a hierarchical notation that represents complex macromolecules as polymeric structures with support for unnatural components (e.g., unnatural amino acids) and chemical modifications. As per the documentation, HELM notation allows monomers to be depicted in a standard atom bond representation, such as SMILES. Each representation is given an identifier type, such as peptide or nucleotide. Monomer sequences can be linked, producing biomolecules that have multiple monomer types. For more information about the HELM notation, see items 1 and 2 in the References section.

A Java Swing based HELM Editor exists allowing users to take advantage of its functionality to edit and display sequences of monomers and/or monomer fragments. This project ported the existing HELM Editor to an open-source, web-based architecture. The core functionality is now hosted on a server and can be accessed through modern web browsers. The pre-existing dependency on commercial software such as MarvinBeans and yFiles are also removed.

Through this web application, users will be able to:

1. Convert biomolecular sequences to HELM notation
2. View graphical representations of biomolecules
3. Edit and modify biomolecules
4. Search and retrieve from the monomer library
5. View molecular properties of biomolecules

1.2. Screenshots of Primary Use Cases of the HELM Editor 2.0

Below are screenshots capturing several primary uses of the HELM Editor 2.0.

1.2.1. Convert biomolecular sequences to HELM notation and view graphical display.

Figure 1 below shows the Load Sequence page, in which a user enters a HELM notation, RNA/DNA sequence, or a peptide sequence.

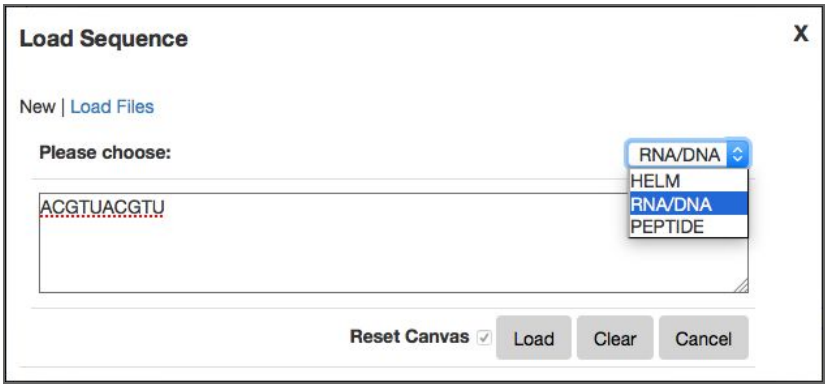

Figure 1: HELM Editor 2.0, showing the dialog where users enter HELM notation, RNA/DNA sequences, or peptide sequences

Once the user has loaded a sequence, a graphical image will appear on the upper canvas, as displayed below in figure 2. The HELM notation also appears, on the lower canvas. If the user loads a RNA/DNA or peptide sequence, it will be converted to HELM notation. If the user inputs a HELM notation, the HELM Editor 2.0 will verify that the input is correct HELM notation. If the notation is valid, the graphical representation will be displayed.

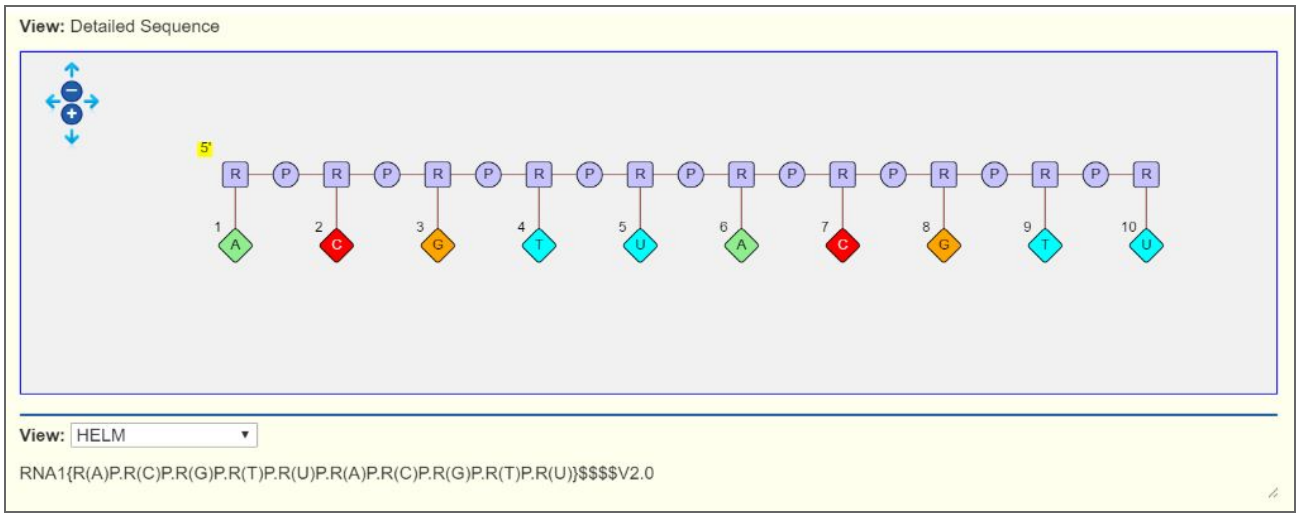

Figure 2: HELM Editor 2.0, displaying a graph of a biomolecule

Users can also view the biomolecular structure. In figure 3 below, a user has loaded a HELM notation, right clicked on the diagram, and clicked “Show Molecular Structure” from the context menu. A graphical representation of the biomolecular structure (retrieved from an external webservice) appeared.

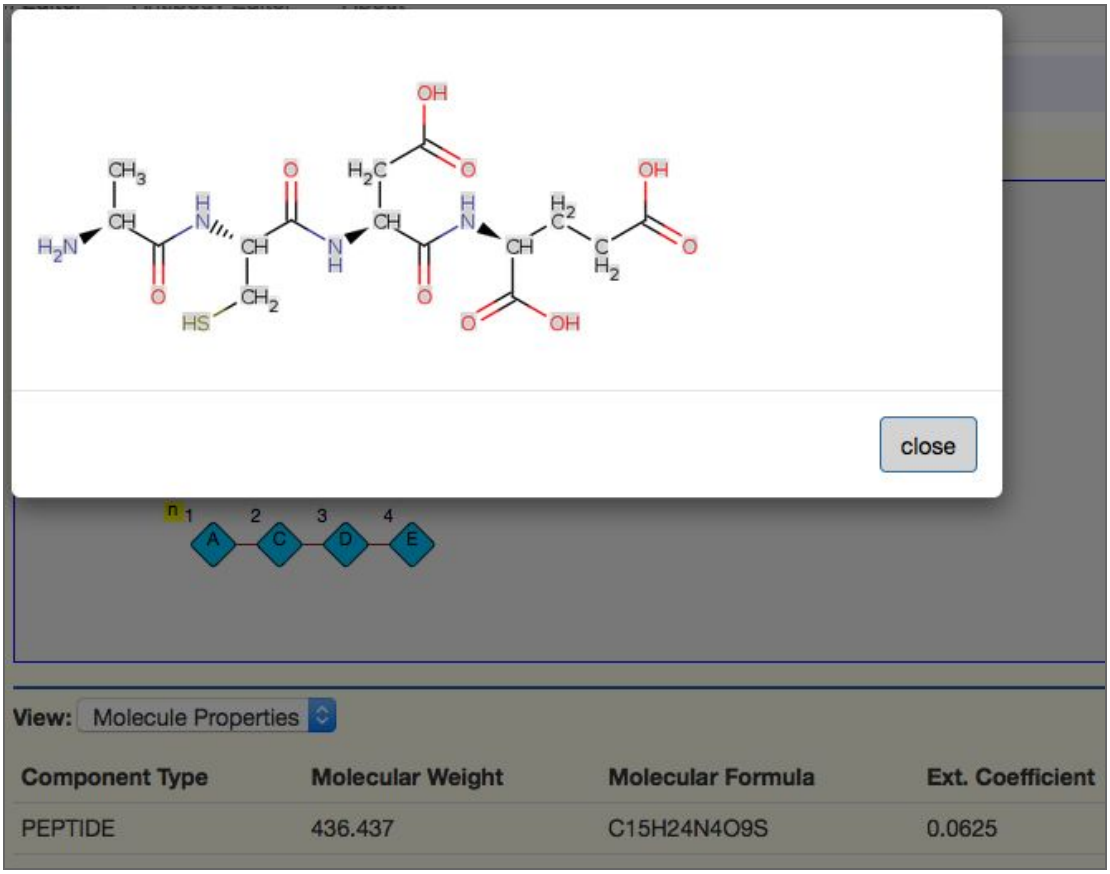

Figure 3: The HELM Editor 2.0 showing a biomolecular structure

1.2.2. Edit nodes on the canvas

Users can edit and delete biomolecules portrayed on the canvas. For example, figure 4 shows the ability to select nodes and delete them from graphical display by using the trash-can icon at the top right corner of the editor (circled in red).

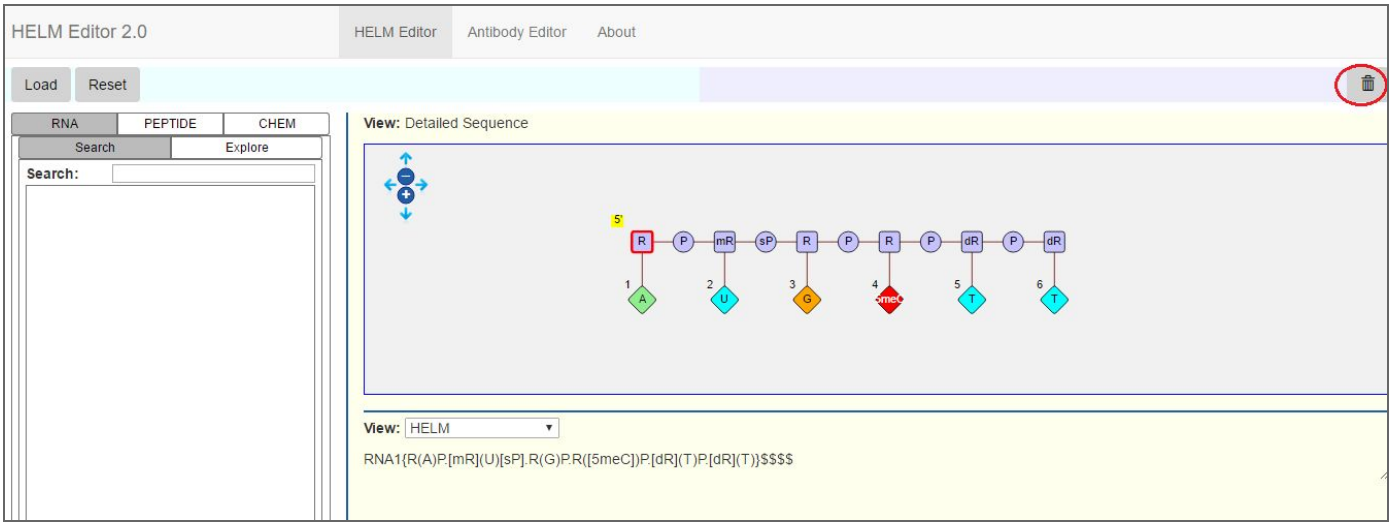

Figure 4: After selecting nodes, users can click the trash-can icon (upper right, circled in red) to delete them.

1.2.3. Search and retrieve from the monomer library

In figure 5 below, a user has searched for the letter ‘g’ on the left panel. Monomers with that letter are listed beneath the search box. Users can then click and drag a monomer to the upper canvas on the right.

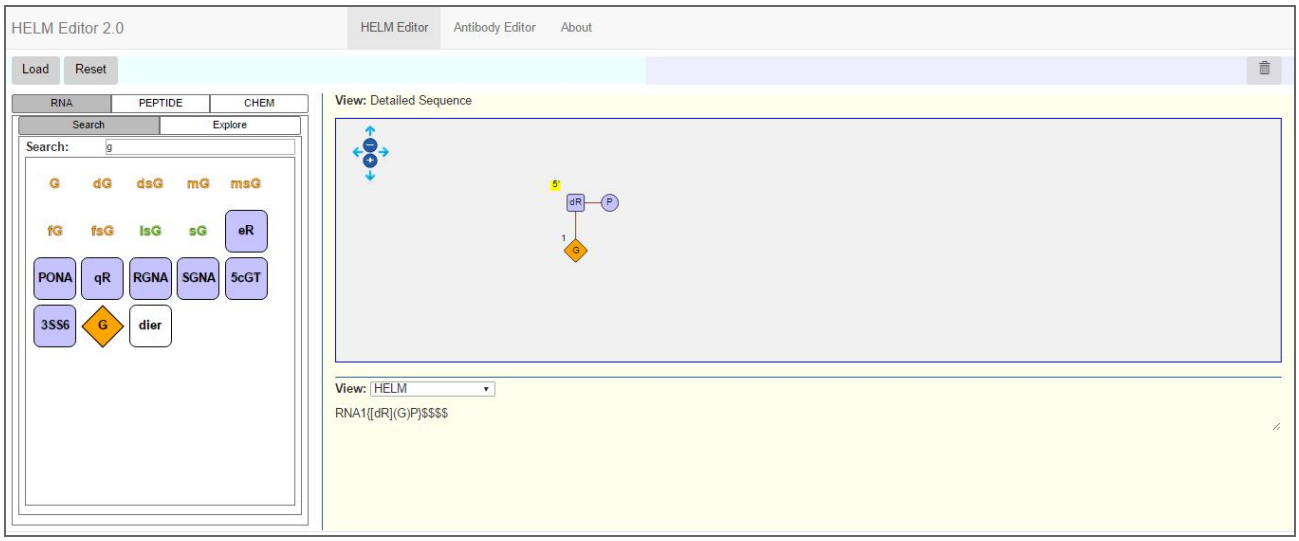

Figure 5: A user retrieved a biomolecule from the monomer library on the left and dragged it to the canvas

1.2.4. View molecular properties of biomolecules

Figure 6 below shows two ways to view molecular properties. A user can select 'molecular properties' from the drop-down menu on the lower pane, or from the right-click context menu options on the upper canvas. The molecular properties include the component type, molecular weight, molecular formula, ext. coefficient, and molecular structure.

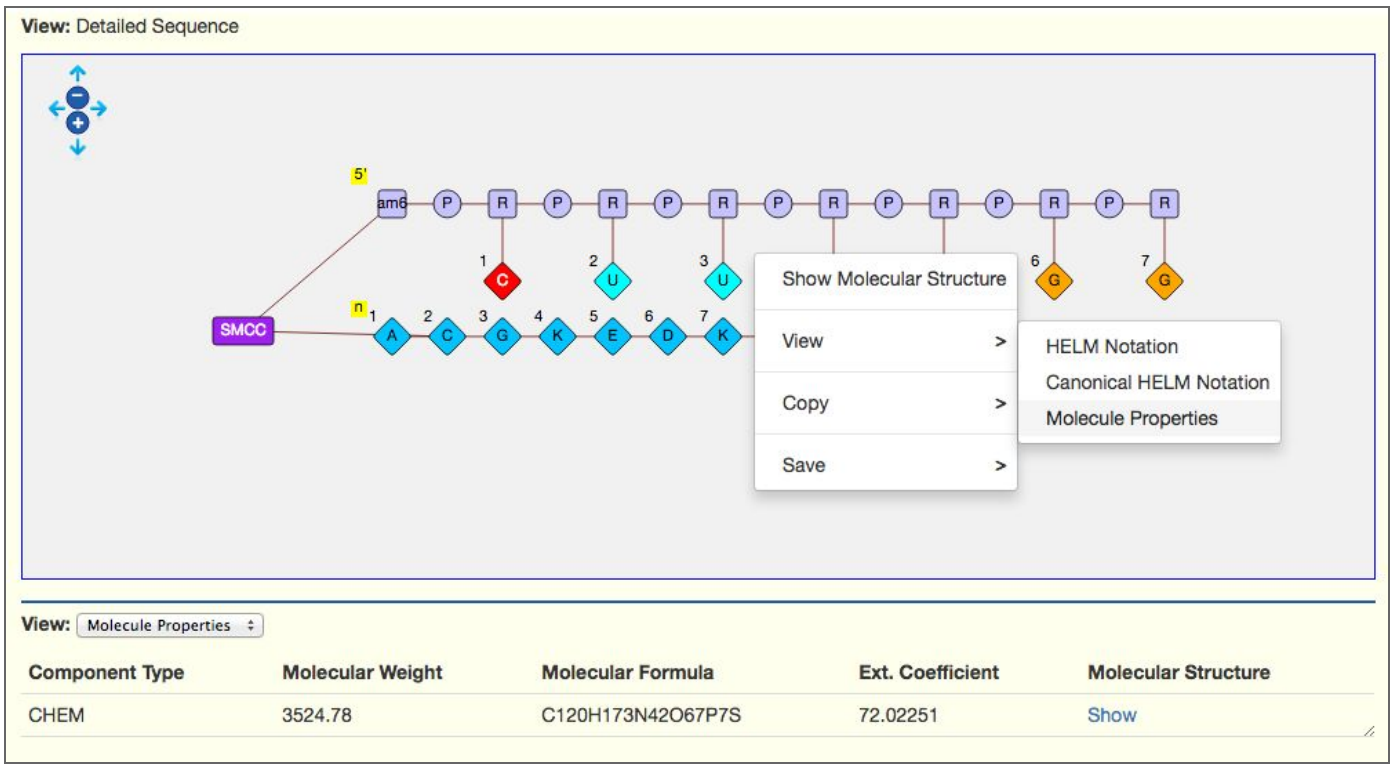

Figure 6: Users can view molecular properties on the lower pane, or through the right-click options on the upper canvas

1.2.5. View molecular properties of monomers

Users can view the monomer details by double-clicking the monomers displayed on the monomer palette. In figure 7 below, a user has double clicked the monomer "A". The monomer details, such as the ID, name, type, molecular structure, HELM notation, Molfile, and SMILES string, are displayed as a popup.

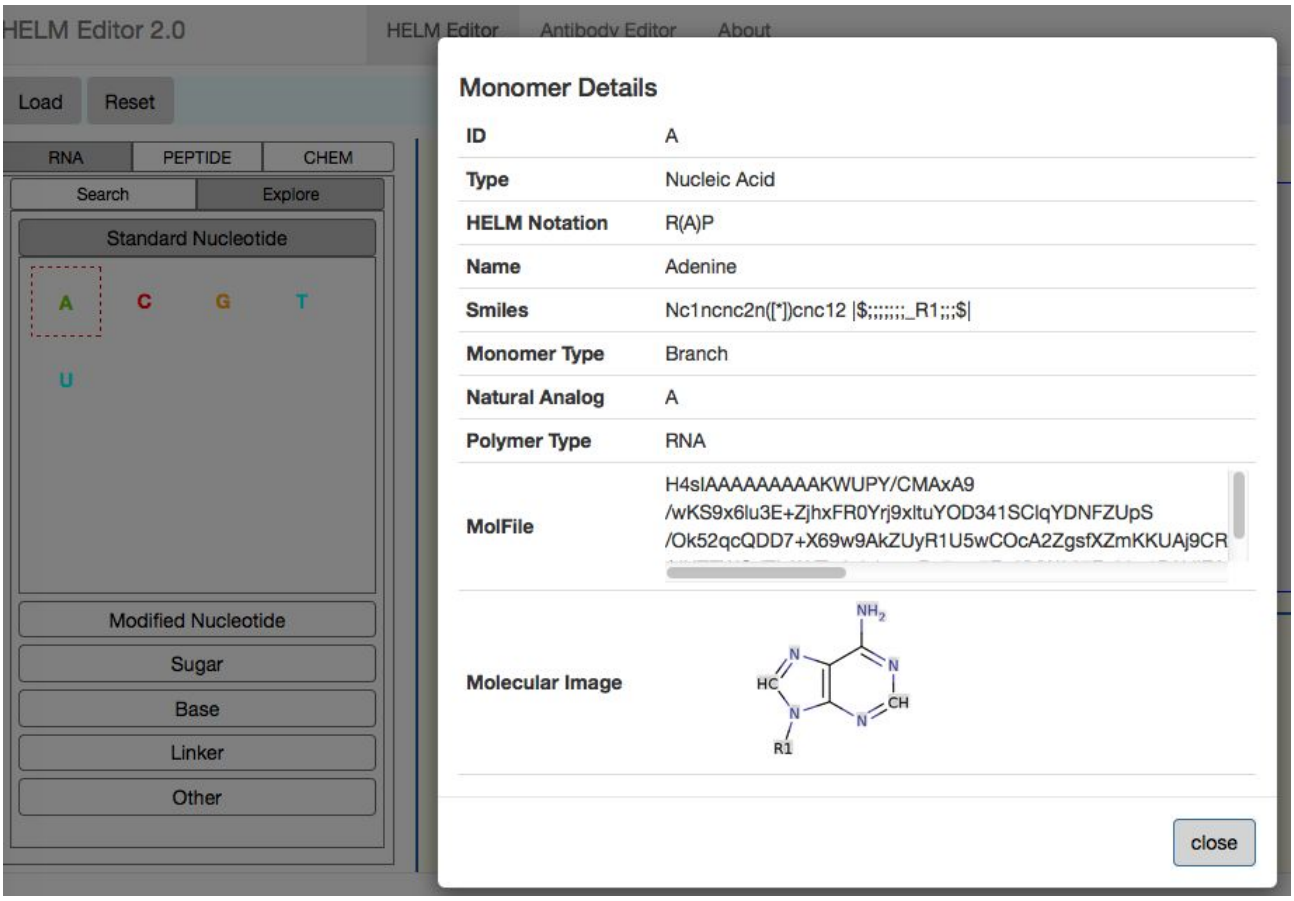

Figure 7: Users can view monomer properties by double-clicking the monomer on the monomer palette

2. Design

2.1. Application Components

HELM Editor Application

- The core framework for the web application is **AngularJS**, utilizing **HTML5, CSS,** and **Bootstrap**.
- All graphical representation of the polymers and monomers is accomplished through the use of **SVG**. This has replaced the use of the commercial tool yFiles that was used in the previous Java application.
- The web application is hosted on a DigitalOcean droplet for the purpose of this project. This server is accessible here: http://104.236.250.11/editor/dist/.

- There is no 'backend' for the application -- all files are served from the hosted version and everything is handled client-side, aside from the functionality provided by the HELM2WebService.
- The hosted instance of the HELM2WebService is used to translate RNA/PEPTIDE sequences into HELM notation, compute the molecular weight of a polymer, retrieve the molecular image, and provide a searchable monomer database, among other functions. These functions are detailed in the section below.

Hosted version of HELM2WebService

- The HELM 2.0 Toolkit initial implementation is available as an open-source web service.
- The web service exposes a RESTful API and has the following functionalities:
    1. Conversion from Peptide or RNA string sequences into HELM notation
    2. Conversion from HELM notation into canonical HELM notation
    3. Conversion from HELM notation into JSON output
    4. Conversion between HELM notation and Fasta Sequences
    5. Generation of image representations of monomer and HELM molecules
- The HELM2WebService is hosted on an Ubuntu server running on a DigitalOcean Droplet, running Apache Tomcat 8. This server is accessible here: http://104.236.250.11:8080/WebService/DocumentationHELMNotationToolKitRestAPI.pdf.

2.2. Technical Architecture

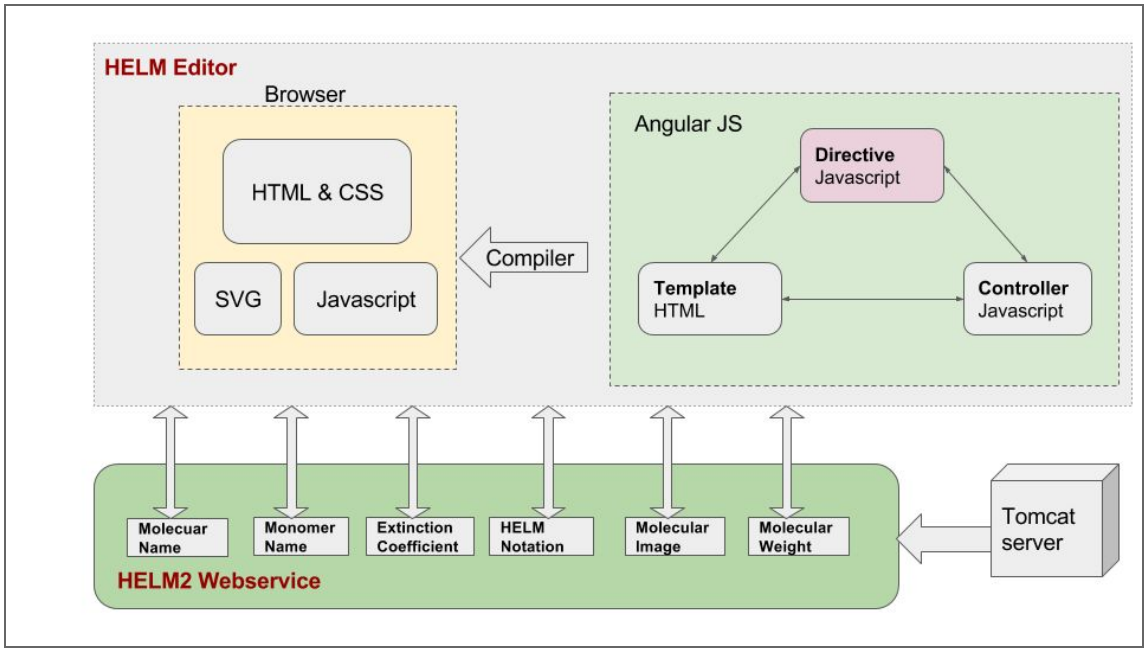

Figure 8: The HELM Editor 2.0 Technical Architecture

2.3. Use case diagram

2.3.1. Use Case diagram for HELM editor web application

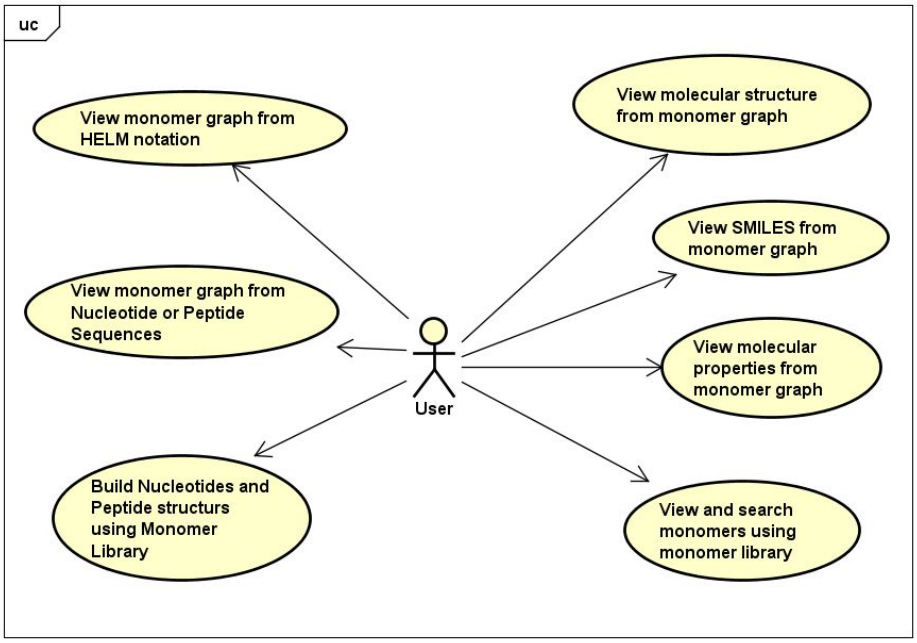

Figure 9: The HELM Editor 2.0 Use case Diagram

2.4. Component Diagram

2.4.1. Component diagram for HELM editor web application

The component diagram shows the interactions between the main elements of the system: Views/Templates, Controllers, Services, and Directives.

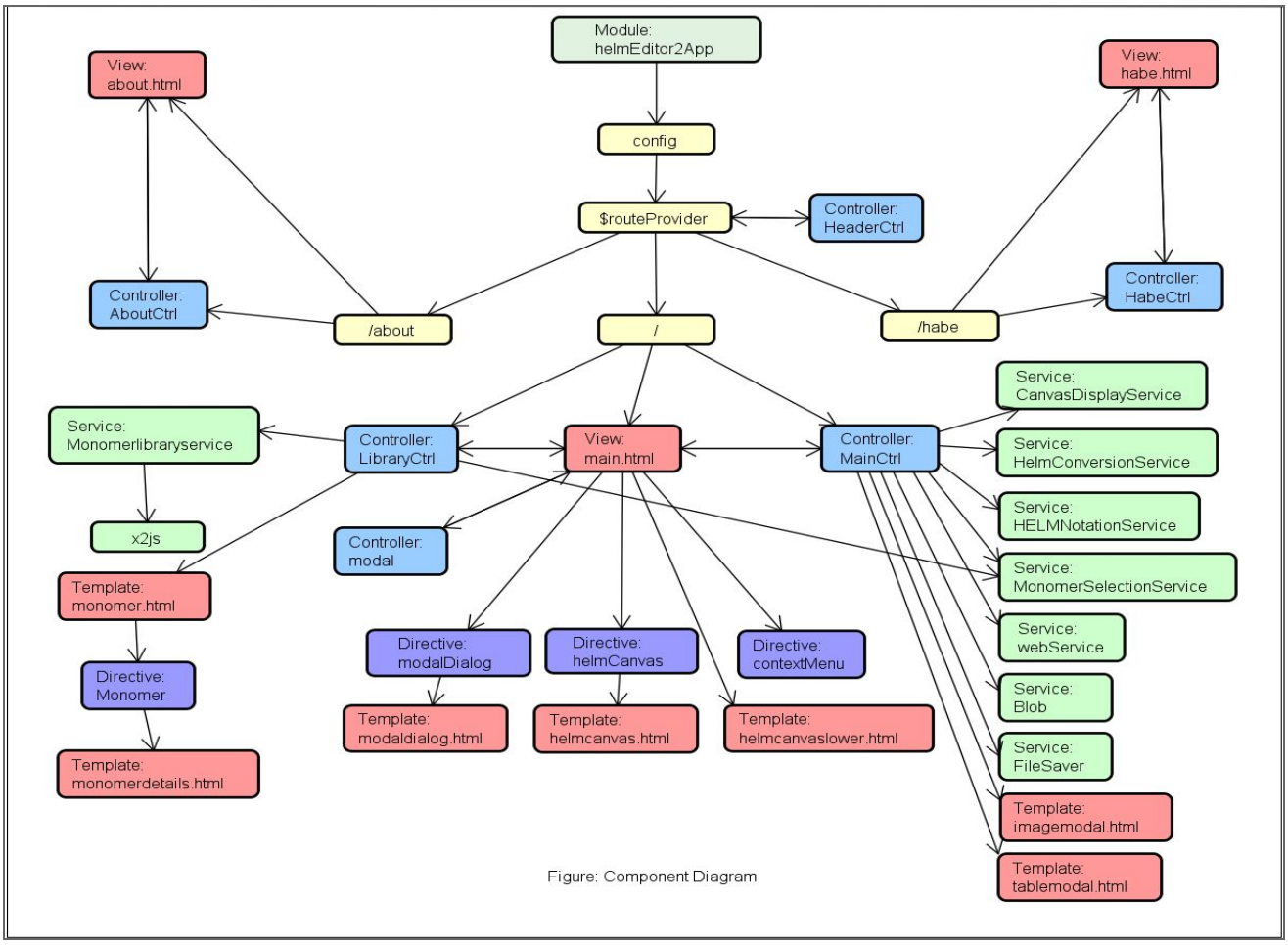

Figure 10: The HELM Editor 2.0 Component Diagram

2.5. Sequence Diagrams

2.5.1. Load Sequence Functionality

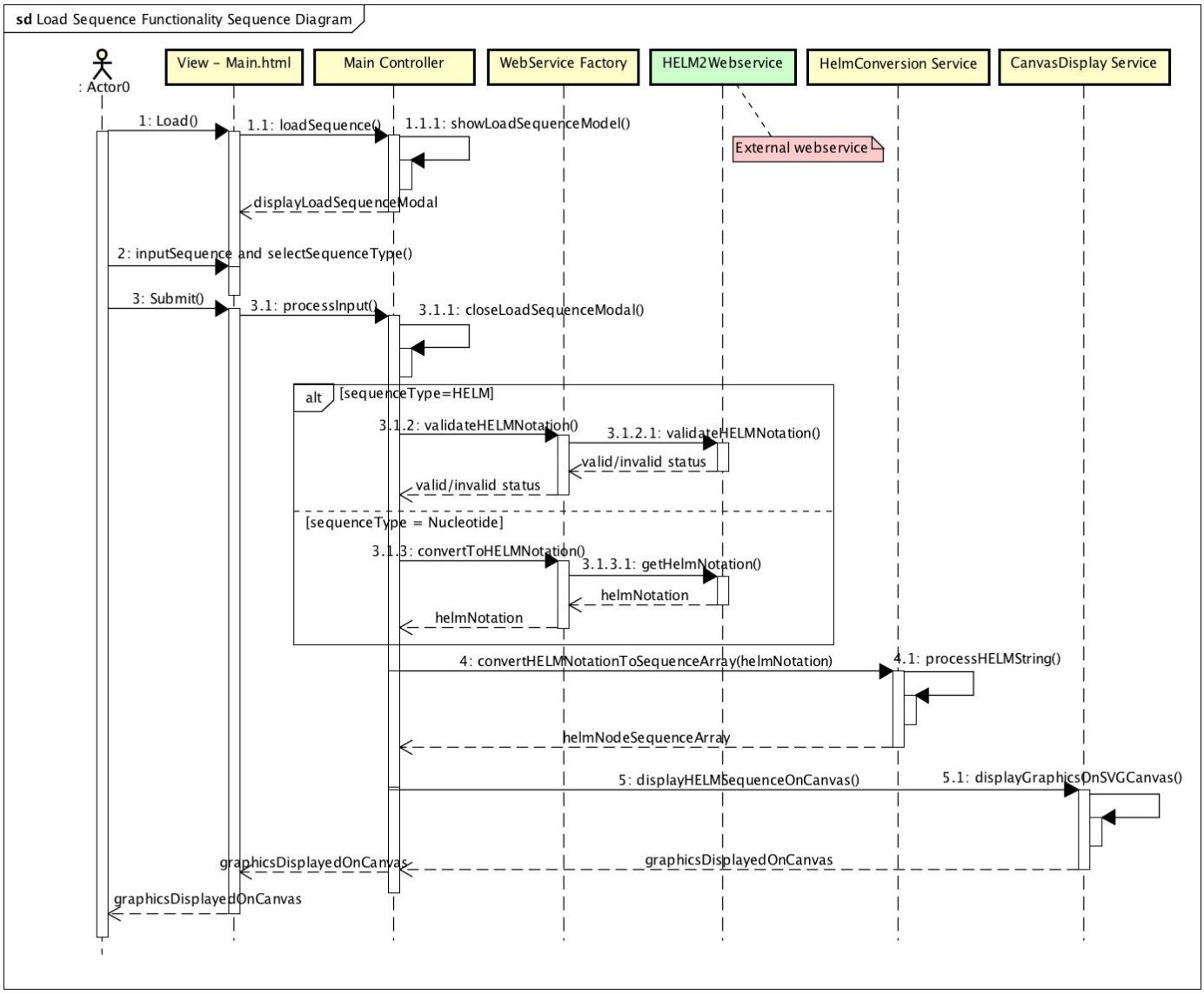

Figure 11: The HELM Editor 2.0 Load Sequence Functionality Sequence Diagram

2.5.2. Monomer Library - Initial Launch - Sequence Diagram

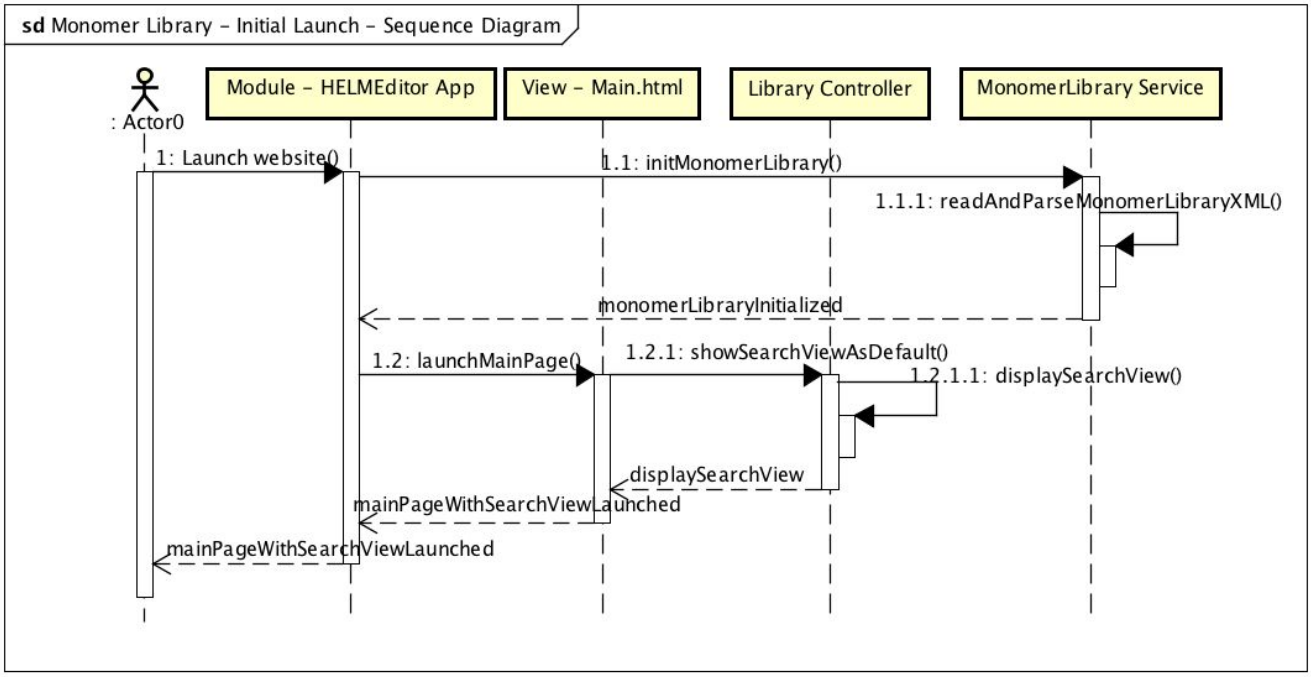

Figure 12: The HELM Editor 2.0 Monomer Library Initial Launch Sequence Diagram

2.5.3. Monomer Library - Search Functionality - Sequence Diagram

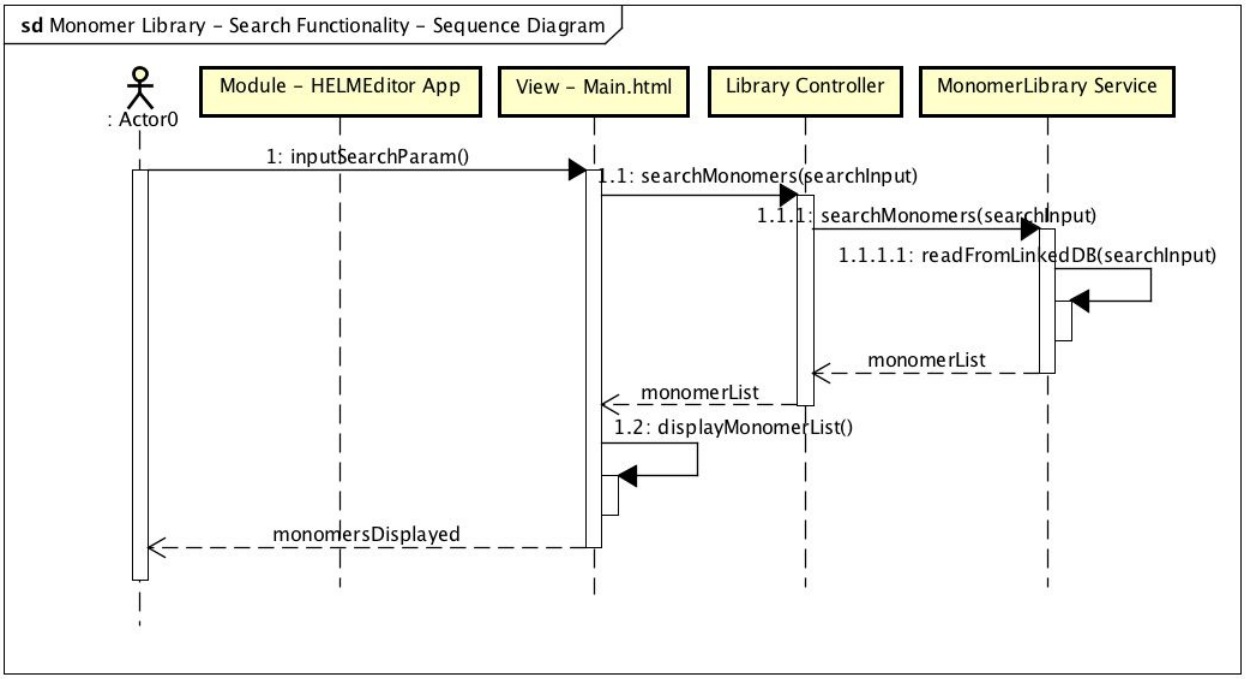

Figure 13: The HELM Editor 2.0 Monomer Library Search Functionality Sequence Diagram

2.5.4. Monomer Library - Explore and Select Monomer Functionality - Sequence Diagram

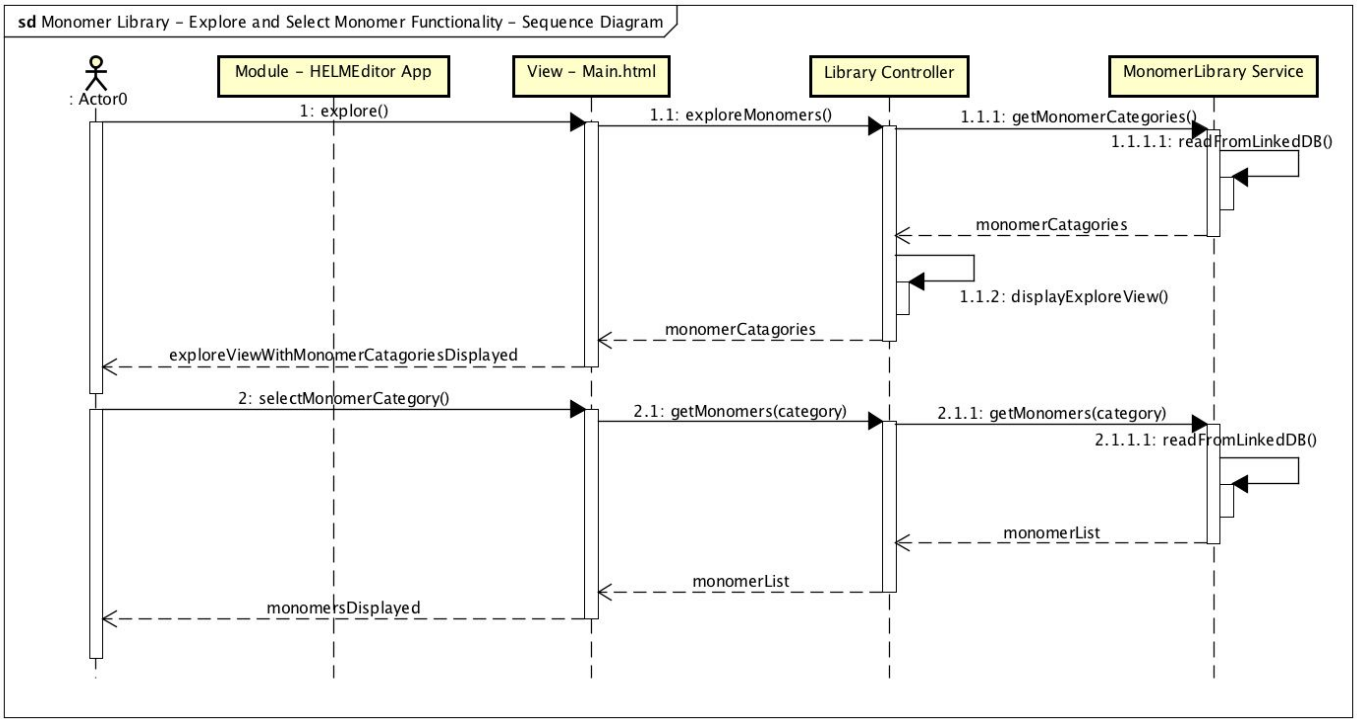

Figure 14: The HELM Editor 2.0 Monomer Library Explore and Select Monomer Functionality Sequence Diagram

2.6. Development Environment

All the code is stored within a public GitHub repository, which can be found here: https://github.com/CSCIE-599/HELM-Editor-UI. After cloning the repository to a workstation, the user must install several tools globally in order to have a fully-functioning development environment. Explicit instructions are listed in Appendix 5.1. The tools that are needed globally are: NodeJS, NPM, Grunt-CLI, and Bower. Once these are installed, the rest of the dependencies can be installed locally via Bower and NPM, from the root directory of the project that has been cloned.

There is no required (or suggested) text editor or IDE, and the team has employed a number of text editors with no adverse effects. As the master branch is locked within GitHub, all development and bug fixes must occur on separate branches, and the development environment is configured to only allow Pull Requests to be merged into master once an automated build and test process (run on Travis-CI) completes with all tests passing successfully. See the Testing section below for more detail on the test setup and how to run the tests.

The original scaffolding of the application was provided through Yeoman, and it is *strongly* recommended that you use Yeoman when adding Controllers, Routes, Directives, or any other Angular components to the application. This will ensure that the files are placed in the appropriate

location within the application directory, and provide the initial scaffolding for each component and the appropriate test file.

In order to run the web application, there are two supported methods. First, a user could host the files on any Apache web server, and run the dist/index.html as the root page of the application. Otherwise, during development, a grunt target has been provided to run a lightweight web server to host the Angular application. Users can start this with the task $grunt serve.

2.7. Build and Deployment

All building, deployment, and testing is handled through the Grunt task manager. Grunt is provided with the default Yeoman Angular scaffold and used as the means of building, testing, and deploying the application. Everything that is done by Grunt can be viewed by inspecting the file Gruntfile.js within the root directory. The short form of how Grunt is configured is as follows: to build the application (i.e. the default grunt target mentioned above), Grunt performs the following steps:

1. Run jshint (see section below on Code Validation for details)
2. Run jscs (see section below on Code Validation for details)
3. Run tests, which includes the following steps
    a. Clean all working directories
    b. Run wiredep, which injects Bower components into the application
    c. Run the test web server
    d. Run the unit tests through Karma
4. Build the application
    a. Clean dist/ directory
    b. Run wiredep, which injects Bower components into the application
    c. Compile and minify all source code into single javascript and HTML pages (multiple steps utilizing Grunt plugins UseMin, ngTemplates, ngAnnotate, concat, cdnify, cssmin, ugliy, filerev, and htmlmin) - this was included with the base Yeoman scaffold, though it was slightly modified to handle the application structure changes we were forced to make.
    d. Copy all files into the dist/ directory.

It is expected that anyone trying to build the application on their own use the Grunt default task directly, which should build the application fully once it passes all code validation and unit tests. End-to-end tests have been removed from the default build task due to the lack of a common web browser to test this on. End users should run these on their own before making any commits. However, the end-to-end tests (on Firefox) are run on all Travis builds, so they must pass before any Pull Request may be merged.

Additional Grunt targets have been provided to build, run unit tests, run end-to-end tests, run a test server, and are repeated in the appendix, but are listed here for convenience:

- $ grunt - run all tests, build, minify, and distribute
- $ grunt build - build app, without running tests
- $ grunt serve - build app and run server locally to test manually
- $ grunt test - run Karma unit tests one time
- $ grunt test-continuous - run Karma continuously, testing with every file that's saved
- $ grunt protractor-chrome - run Protractor tests only on Chrome
- $ grunt protractor-firefox - run Protractor tests only on Firefox
- $ grunt protractor-all - run Protractor tests on Firefox and Chrome, simultaneously

In addition to the Grunt targets that have been provided, the project is also set up with Travis-CI for continuous build, test, and deployment. For each commit to any user branch, Travis pulls the repo into a clean environment, installs all dependencies, and runs the default target (unit tests and build), as well as the end-to-end tests on the Firefox browser. This build process also runs on any Pull Requests to master, and *must* complete successfully before the code can be merged into the master branch. This process ensures that both the code quality and test coverage are tracked, and limits the number of breaking changes that can potentially make it into master.

Lastly, for this project we have hosted the latest stable build of the application here: http://104.236.250.11/editor/dist/#/. This build and deployment is *also* automated through Travis, where any commit to the stable branch will result in a build, which, if it succeeds building and end-to-end tests, will automatically deploy this latest version to the above URL. The previous build (i.e. the build last deployed to stable before this one) is copied to http://104.236.250.11/editor/backup/ and kept as a backup. This backup is overwritten with any new push to stable and successful build.

3. Development Process & Lessons Learned

3.1. Meeting the requirements

We met our Milestone 2 and 3 goals. We used 2-week agile sprints, and allocated work based on user stories listed on Github as 'issues'.

The lists below provide more detail regarding our progress towards meeting the project requirements:

Interfacing with HELM2WebService

- The Application provides all the API methods required to interface with the HELM2WebService to carry out the following functionalities:
    - Validate HELM notation,
    - Convert peptide or RNA string sequences into HELM notation,

- Convert HELM notation into canonical HELM notation,
- Convert HELM notation into JSON output,
- Convert HELM notation into Fasta sequences, and vice versa.

User Interface to input a sequence

- Users can input either a RNA, PEPTIDE, or HELM sequence. The application will invoke appropriate web service API to generate or validate the HELM sequence.

Canvas display of HELM strings

- After loading a RNA, PEPTIDE, or HELM notation sequence, users can view a graphical representation of the molecule on the upper canvas.
- Users can modify graphical representations by adding, removing, and replacing monomers.
- On the lower canvas, users can select to view the HELM notation, the sequence, or the molecular properties of the sequence loaded.
- Users can zoom in and out, and pan right and left, on the upper and lower canvases.
- Users can right-click on the upper canvas to 1) view the molecular structure image, HELM notation, or canonical HELM notation in a modal window, 2) save notation to a local file, 3) copy the notation to the clipboard, or 4) view molecular properties, such as molecular weight and formula.
- The HELM Editor 2.0 does not graph biomolecules with two or more linked cyclical sequences (a rare instance). If a user loads such a biomolecule sequence, a warning will appear on the upper canvas. The cyclical sequences will be drawn in the upper canvas, but they will not be linked to each other.

Monomer Library

- Users can look up a monomer using ID, name and type, to get details about the monomer.
- Users can obtain a list of polymer types (Peptide, RNA/Nucleic Acid, Chem).
- Users can retrieve a list of monomer groups for a polymer type.
- Users can get a list of all monomers, categorized by polymer type and monomer group.
- Retrievable objects are exposed internally with shape and color pallet information.

3.2. Estimates

The process we followed for estimating the effort was relatively straight-forward. We first estimated the effort on tasks by analyzing the existing HELM editor functionality, reviewing the HELM editor 2.0 RFI and other documents and through customer interviews. Then we prioritized the tasks based on the input from the HELM experts. Next we researched on the tech stack and the implementation effort associated with the technology. Since all team members were new to the tech stack that we chose, we added additional time to account for the learning curve. We also identified functionalities that can be worked on independently and assigned that to individual resources. Also, we identified the particularly complex functionalities, divided them into sub-tasks, and assigned multiple resources to worked on them together. Since the team members worked

on multiple tasks and did not necessarily live in the same regions, we accounted for additional integration time in our initial estimate.

We have accomplished all of the expected user stories set in our Project Plan and included some of the optional features. We set up a web server running a hosted version of the code, and an automated build and test process. The HELM Editor graphs basic HELM sequences, as well as multiple sequences and connections between sequences. It also includes a monomer library that is searchable and returns some monomer and pallet information. Internally, these monomer objects are exposed in a way that allows access to color pallet and other information.

Sprint Schedule

Sprint	Description	Start	End	Status
Sprint 1	Ensure development environment is complete; complete estimation of tasks, ensure all tasks are in Github under the appropriate milestone (if not already complete); begin development of application framework.	03/04/2016	03/20/2016	Done
Sprint 2	Be able to show the application to stakeholders with very basic functionality.	03/21/2016	04/06/2016	Done
Sprint 3	Continue to add functionality.	04/08/2016	04/17/2016	Done
Sprint 4	Finalize application (quality assurance and testing).	04/18/2016	05/04/2016	Done

Completed Tasks and Estimates

The table below, lists the major tasks that were carried out for the project. In it, '1 week' corresponds to 14 hours per week per resource. A resource corresponds to an individual developer.

Task	Details	Initial Estimate	Actual

Learning and ramp up	Learn AngularJS and SVG and the development environment tech stack. Gain indepth knowledge on the HELM domain and existing application.	1 week 6 resources	1 week 6 resources
GitHub Issue 3 - Development environment setup	The base application has been created and added to the project repository in Github.	1 week 1 resource	.75 weeks 1 resource
GitHub Issue 4 - Automated build system setup and deployment to web server	Setup an automated build and test process that is run on each Pull Request from a development branch to main, and on each commit to main. Publish the application to a web server.	1 week 1 resource	1.25 weeks 1 resource
GitHub Issue 5 - Canvas Pane graphical display of HELM string	Application can parse a HELM sequence. Canvas pane can render the graphical representation of a HELM sequence.	2 weeks 2 resources	2 weeks 2 resources
GitHub Issue 6 - Load peptide/RNA/ HELM sequence	The peptide or RNA string that is entered into the modal dialog is converted to HELM notation, and validates HELM notation.	2 weeks 1 resource	2 weeks 1 resource
GitHub Issue 8 - Internal support for monomer library	The existing monomer library and categorization information is converted into a single store of data, to facilitate integration with the web application	2 weeks 1 resource	2 weeks 1 resource
GitHub Issue 9 - Monomer pallet initial support	Monomer palette is implemented and added to the web application. Monomer palette supports exploration by monomer type only.	2 weeks 1 resource	2 weeks 2 resources
GitHub Issue 10 - Ability to add new monomers to existing graphical sequence	As a user of the web application, I am able to drag and drop elements from the monomer palette onto the Canvas to add them to the current sequence or replace existing monomers, so that	1 week 1 resource	1 week 1 resource

	I can begin to modify the existing molecule.		
Github Issue 11 - Ability to remove elements from graphical sequence	As a user of the web application, I can select an existing element from the Canvas and delete it, so that I may remove unwanted elements.	1 week 1 resource	1 week 1 resource
GitHub Issue 12 - Alternate representations supported in Canvas	The canvas pane can render alternate sequence representations (eg, PEPTIDE sequence, HELM, Canonical HELM, image). Dropdown added to the canvas pane to allow user to select desired representation. When new representation is selected, application makes a call to the HELM2WebService to retrieve the new representation.	1 week 1 resource	1 week 1 resource
GitHub Issue 13 - Contextual menu (replace right-click menu)	After right-clicking on a biomolecule, users now see a contextual menu, with drop-down options to view the biomolecular structure in a modal, view the notation in a modal, copy the notation to a clipboard, or save the notation to a file.	1 week 1 resource	1 week 1 resource
GitHub Issue 14 - Monomer and Sequence Details (Bonus Deliverable)	Monomer detail view added to the web application and contextual menu. Monomer detail view accessed when a user double clicks on a monomer in the monomer library. View includes information that can be retrieved from the HELM2WebService, such as an image of the chemical representation, molecular weight, etc.	1 week 1 resource	1 week 1 resource

GitHub Issue 15 - Search for monomer by ID and name (Bonus Deliverable)	There is a search field within the monomer palette allowing the user to enter a monomer ID or name. Monomer palette displays any search results from the monomer library.	1 week 1 resource	1 week 1 resource
GitHub Issue 20 - Interfacing with HELM2Webservic e	Added functionality in HELM Editor AngularJS application to interface with the HELM2Webservice to define APIs to all web service methods.	2 weeks 1 resource	2 weeks 1 resource
GitHub Issue 26 - Constants and properties should be retrieved from a config file	For Webservices, added a config array to specify the base url and all webservice API urls.	0.25 weeks 1 resource	0.25 weeks 1 resource
GitHub Issue 28 - Create a main layout	The main layout/frame of the landing page has been created. It consists of: Left Pane - to hold monomer palette; Right top pane - for graphical display/editing; Right bottom pane - for displaying helm notation/ monomer info etc.	0.25 week 1 resource	0.25 week 1 resource
GitHub Issue 39 - Modify layout to make the bottom "pane" only support alternate viewings	In the lower canvas, users can select from a dropdown menu to for alternate views, including of HELM notation, sequence, or molecular properties such as molecular weight and formula.	1 week 1 resource	1 week 1 resource
GitHub Issue 40 - Remove all unnecessary views from the navigation	Unnecessary alternate pages have been removed, to improve UI.	0.25 week 1 resource	0.25 week 1 resource
GitHub Issue 41 - Update and About views to	The ‘About’ view has been updated with data about the HELM project and the development team. The ‘Antibody	0.25 week 1 resource	0.25 week 1 resource

have meaningful placeholders	Editor' view states that that tab is 'under development'.		
GitHub Issue 42 - Modify Canvas display to warn when multiple cycles are detected to show a warning	The canvas display does not correctly display sequences that have multiple, linked cyclical sequences. When a user loads such a sequence, an alert message appears to warn the user. The cyclical sequences are drawn, but they are not connected.	0.25 week 1 resource	0.25 week 1 resource
Unit testing (GitHub Issues 43-49)	Unit tests for each controller and directive and service are complete.	1 week 6 resources	1 week 6 resources
GitHub Issue 50 - End to end tests	A full set of end-to-end tests for the main helm editor has been implemented. Specifically, the following functionalities are now tested: - clicking load displays the modal dialog, - clicking the x hides the modal dialog, - clicking outside of the modal dialog hides the modal dialog, - loading in an incorrect PEPTIDE sequence results in an error message, - loading in an incorrect NUCLEOTIDE (RNA/DNA) sequence results in an error message, - loading in a correct/known PEPTIDE sequence results in valid HELM and valid SVG display, - loading in a correct/known NUCLEOTIDE sequence results in valid HELM and valid SVG	1 week 1 resource	.5 weeks 1 resource Per guidance from Peter and the stakeholders, a fuller suite of end to end tests was deferred in favor of other bug fixes. This is something that can be added on at a later date, as the framework for running these tests exists.

	display,
	- loading in a known-incorrect HELM sequence results in an error message,
	- loading in a known-correct HELM sequence results in the HELM string and valid SVG display for:
	  - a single polymer,
	  - a HELM sequence with multiple polymers,
	  - a HELM sequence with a single cycle,
	  - a HELM sequence with two cycles - should result in warning, and
	- toggling the lower pane flips from HELM to Canonical HELM to Image.		

GitHub Issue 51 - Enable user to zoom in and out on the SVG drawing	Users can now zoom in and out, and right and left, on graphical representations of biomolecules.	2 weeks 1 resource	2 weeks 1 resource
GitHub Issue 52 - Create checkbox when loading new sequence to clear current canvas	Users can now check a box to clear the current canvas.	0.25 week 1 resource	0.25 week 1 resource

3.3. Risks

1. A primary risk was meeting our milestone goals on deadline in the given timeframe. The team had no previous exposure to the HELM project. We estimated tasks based on the HELM RFI and other documents and customer interviews, rather than experience with the HELM suite.

   Mitigation: Given this risk, we have prioritized certain tasks, such as transitioning the HELM Editor to a web-based architecture, and noted that we will only implement other customer

requests, such as representing molecular ambiguities, if time permits.

2. Most of the team members had little experience with AngularJS or SVG.

   Mitigation: Team members have taken AngularJS tutorials and we helped each other to get started with the development.

3. Lack of experience working on a distributed team, in an asynchronous model.

   Mitigation: Team meets often over Hangouts (at least weekly) for progress updates and questions/discussion, and share emails with important updates as they are needed.

4. Lack of familiarity with development environment and tooling (grunt, jshint, protractor, karma, etc).

   Mitigation: Team met to go over the build environment and process early on, and assist each other as needed to commit, merge, and test code.

3.4. Team Dynamic

We met weekly on Google Hangouts to provide each other status updates and ask questions. We also communicated frequently through e-mail.

To allocate work, each team member chose one or more user stories listed on our Github repository. Team members worked together on the more complex user stories. For example, Thankam and Sarah collaborated on “User Story 5: Canvas Pane Can Display Graphical Representations of HELM String”.

3.5. Lessons Learned

1. Continuous collaborations and synch up is important.
2. Learned to understand and build upon another person’s code.
3. Learned to better communicate one's own logic.
4. Learned that working together produces better results than working alone.
5. Integrating code earlier in the process is important.
6. Simply devoting extra time to address issues sometimes results in diminished return on investment and fresh eyes from teammates or other resources can be necessary.

4. Testing

This project has combined the best practices for AngularJS applications with regards to code validation and testing, which are all run automatically on each build and push/Pull Request.

4.1. Code Validation

For code validation, the project uses both JSHint and JSCS. JSHint is a javascript code quality tool, checking for common javascript coding errors such as unused or undefined variables, lack of semicolons at the end of statements, and others. JSCS, on the other hand, is a code "style" checking tool. JSCS ensures things like consistent use of single or double quotation marks. Both configurations for JSCS and JSHint are, for the most part, as included by the base Yeoman scaffold.

Both tools must run on each build (i.e. automated on each push to any branch and Pull Request to master), and the build process fails if either do not exit with success.

4.2. Unit Testing

Unit tests are run utilizing the Karma javascript test runner, in which all unit tests are written in the Jasmine behaviour-driven javascript language, on top of the headless javascript framework PhantomJS. Currently there are 33 unit tests, though this number will raise significantly before Milestone 3, as one of the goals of Milestone 3 is to increase unit test coverage.

There are two Grunt targets for running the unit tests:

- $ grunt test - This runs all unit tests that are included within the /test/spec/ folder of the application a single time and then exits.
- $ grunt test-continuous - This runs all unit tests within /test/spec/ and then continues running, re-running the test suite on every file change in the application.

It is strongly suggested that during development, the user open a terminal window and run the second target above, and keep that window running (in view) during development to ensure no tests have broken during development.

All unit tests are run as part of the automated build process, and builds and Pull Requests are rejected if all unit tests do not pass.

4.3. End to End Testing

End-to-end testing is handled through Protractor, which utilizes the Selenium server and Jasmine language to describe the tests. The end-to-end tests are intended to test the actual use of the application.

There are three Grunt targets for the end-to-end tests, and it is up to the developer to choose which to run depending on the browsers that are installed:

- $ grunt protractor-chrome - This runs all end-to-end tests that are included within the /test/e2e/ folder of the application a single time on the Chrome browser and then exits.

- $ grunt protractor-firefox - This runs all end-to-end tests that are included within the /test/e2e/ folder of the application a single time on the Firefox browser and then exits.
- $ grunt protractor-all - This runs all end-to-end tests that are included within the /test/e2e/ folder of the application a single time on the Chrome and Firefox browsers, simultaneously, and then exits once both complete.

The end-to-end tests are run automatically on each Travis build, to ensure that all known workflows pass before allowing any merges to happen to master. In Travis, because there is no support for Chrome, the tests are only run on Firefox.

4.4. Known Issues

There are a few existing issues with the web based editor. Some of these issues exists in the Swing based editor as well:

1. The editor cannot handle cyclic Nucleotide sequences. Currently it can only support cyclic peptides.
2. The editor will not draw links between multiple cyclic peptides. Instead, the cyclic peptides are graphed without links connecting them.
3. The molecular structure image for cyclic peptides will not show, because the HELM2WebService returns a 500 error status code. It is currently captured and thrown as an exception to the front-end.
4. The editor cannot support polygon shapes with more than 4 sides. Example, Chemical modifier(CHEM) nodes are displayed as a rectangle instead of Hexagon.

5. Appendix

5.1. Keywords & References

Term	Description
HELM	Hierarchical Editing Language for Macromolecules. Active project under the Pistoia Alliance: http://www.pistoiaalliance.org/projects/hierarchical-editing-language-for-macromolecules-helm/
HAbE	HELM Antibody Editor
Monomer	An individual molecule that may bind with other molecules to create a polymer. Examples include ‘A’, ‘T’, ‘G’ etc.
Polymer	Large molecule, or macromolecule, composed of many

	repeated subunits (e.g. monomers). https://en.wikipedia.org/wiki/Polymer
HELM2NotationToolkit	Java toolkit exposing HELM 2.0 functionality such as HELM notation validation, conversion between different notations, and image generation. This is an open source project, available here: https://github.com/PistoiaHELM/HELM2NotationToolkit
HELM2WebService	External Java-based Web Service API exposing HELM2NotationToolkit functionality. This is an open source project, available here: https://github.com/PistoiaHELM/HELM2WebService
Nucleotides	Organic molecules that serve as the building blocks for nucleic acids (such as DNA and RNA). Reference: https://en.wikipedia.org/wiki/Nucleotide
Peptides	Short chains of amino acid monomers linked by peptide (amide) bonds. Reference: https://en.wikipedia.org/wiki/Peptide
FASTA	FASTA is a text-based format for representing nucleotide or peptide sequences. Reference: https://en.wikipedia.org/wiki/FASTA_format
Bootstrap	HTML, CSS, and JS framework for developing responsive web applications. Reference: http://getbootstrap.com/
SVG	HTML5 component named Scalable Vector Graphics. Reference: https://www.w3.org/Graphics/SVG/

5.2. Final User Stories

The following user stories are completed:

1. Development environment setup
2. Automated build system setup and deployment to web server
3. Load peptide, RNA and HELM sequence
4. Interfacing with the external webservice - HELM2Webservice
5. Canvas Pane display of graphical representation of HELM string
6. Graph with multiple sequences and connections between sequences
7. Zooming and Panning of the graphical representation
8. Display of molecular properties and chemical structure corresponding to the HELM string

9. Right click contextual menu on canvas
10. Export of HELM and CHELM strings - copy to clipboard, and save to external file system
11. Monomer library exists and is searchable
12. Monomer from library can be added to the canvas pane by drag and drop
13. Select and remove node from canvas graph
14. Monomer detail can be viewed on double click
15. Units tests and End to End tests

5.3. System Installation

The HELM Editor 2.0 can be accesses through a browser. The hosted version of the HELM Editor 2.0 is available at: http://104.236.250.11/editor/dist/

The project source code can be downloaded at: https://github.com/CSCIE-599/HELM-Editor-UI

5.4. Developer Installation Instructions

This project was generated with yo angular generator version 0.15.1.

The following steps must be done in order to set up your development environment fully.

1. Install NodeJS (version 4.3.2)
    a. https://nodejs.org/en/
2. Upgrade npm
    a. $ sudo npm install -g npm
    b. $ node -v
    c. v4.3.2
    d. $ npm -v
    e. 3.8.0

3. If desired (Yeoman was used to create the scaffolding), install Yeoman (yeoman.io), and the generators we need
    a. $ sudo npm install -g yo generator-karma generator-angular generator-protractor

4. Install bower
    a. $ sudo npm install -g bower

5. Install grunt
    a. $ sudo npm install -g grunt-cli

6. Install node dependencies
    a. $ npm install (from within the project directory)

7. Install bower dependencies
    a. $ bower install

8. Install Selenium (for e2e tests)
    a. $ node ./node_modules/protractor/bin/webdriver-manager update
    b. Note this can take a little while

9. Grunt targets
   $ grunt - run all tests, build, minify, and distribute
   $ grunt build - build app, without running tests
   $ grunt serve - build app and run server locally to test manually
   $ grunt test - run Karma unit tests one time
   $ grunt test-continuous - run Karma continuously, testing with every file that's saved
   $ grunt protractor-chrome - run Protractor tests only on Chrome
   $ grunt protractor-firefox - run Protractor tests only on Firefox
   $ grunt protractor-all - run Protractor tests on Firefox and Chrome, simultaneously

A note on Yeoman

Yeoman was used to generate the scaffolding for the application, which is how all of the folders and files have been created. It is highly suggested that for modifications and additions to components of the application (views, controllers, factories, etc) you use Yeoman. For instance, the following commands might be useful (they should all be run from within the root folder of the application):

1. Create a new view:
   $ yo angular:view <view name>

2. Create a new factory:
   $ yo angular:factory <factory name>

Full information on the angular generator can be found here: [Yeoman generator:angular](https://github.com/yeoman/generator-angular)

5.5. Detail Design and Application Components

Application Module

- *helmeditor2App:* The main module of HELM editor web application is “helmeditor2App” and is defined in app.js. This file defines all application routes using the Angular $routeProvider provider, while also wiring together all views and their respective controllers.

Views

- *main.html*: This is the default view that will be loaded when the web site is launched. It defines the main canvas where the graphics can be displayed and edited.

- *about.html:* This is the simple "About" page of the web application, just listing the project information.

- *habe.html:* This will be the view for the future HELM Antibody Editor, though this has been removed from the scope of the project due to time constraints. Currently only has a placeholder page and route/controller.

Controllers

- *MainCtrl*: This is the main controller. It defines various functions to parse the HELM sequence, generate the graph and to display it on the canvas. It also defines the functions for loading a sequence using a modal dialog, to invoke the webservice to validate the HELM sequence and to convert the Peptide/Nucleotide sequence to HELM notation.

- *HeaderCtrl*: This controller handles the header in index.html, which contains the base template for the web application.

- *AboutCtrl:* Handles the About page data model. This is trivial.

- *HabeCtrl:* Handles the HAbE page. This is trivial, as it has not been developed yet and is not in scope for the project.

- *LibraryCtrl:* This controller interacts with the MonomerLibraryService and MonomerSelectionService providing access to the monomer information.

Services

- *webService*: This Angular Factory provides the functionalities to invoke the HELM2Webservice. It defines the webService APIs and invokes it using $http get or post methods. It processes the response and returns the API specific result.

- *CanvasDisplayService*: This service provides functionalities necessary to convert a sequence of monomers into a graph of nodes and edges. The service is utilized in the controller main.js. All the model objects that are used are defined in this service.

  Following are the primary objects defined for rendering the graphical display:

- **Node**: Any monomer shape that appears in the canvas is a node. A node is defined by the following properties:
  - *Id*: Unique identifier for every node
  - *Name*: The letter displayed inside a node
  - *Node Number*: The number which appears next to the node which shows the position of a node in a given sequence
  - *Type*: Attribute which identifies the type of a node, like Ribose node, Phosphate node or a Base node
  - *Height*: Height of a node
  - *Width*: Width of a node
  - *Color*: Height of a node
  - *Rotate degree*: The degree of rotation which tells SVG to rotate a shape, for example, a square can be rotated 45 degrees to look like a rhombus.
  - *Position* (x, y): The x, y position of the node on the canvas
  - *Corner Radius* (rx, ry): The radius with which the corner can be rounded. A square corner can be rounded to look like a circle

- **Connection**: Any two nodes can be linked together by a connection entity. A connection is defined by the following properties:
  - *Source*: The node from which the connection begins.
  - *Destination*: The node in which the connection ends.

  The length of a connection is determined by the distance between the source and connection nodes.

- **CanvasView**: The canvas view object which holds all the nodes and connections. It provides two methods addNode() and addConnection() which can attach a node or connection to the view.

- **SubGraph**: A sub component of a graph. This is primarily used to connect small graphs together, for example a linear graph to a cyclic graph. It has the following attributes:

  - *First Node*: The first node in the subgraph
  - *Last Node*: The last node in the subgraph
  - *NodesArray*: Array of all nodes in the subgraph, in the order of appearance

- *helmconversionservice*: This service converts HELM notation into an array of monomer sequences and an array of connections between monomer sequences. *MainCtrl* uses these arrays to create a graphical image consisting of nodes and edges, calling helper methods in *CanvasDisplayService*.

- *monomerlibraryservice*: This service imports the monomer library and provides an API for accessing the monomers.

- *monomerselectionservice*: This service allows the selection of a monomer from the monomer palette and adding to the canvas.

- *helmnotationservice*: This service provides the ability to manage interactions HELM notation sequence and connections.

Directives

- *modalDialog*: This is the AngularJS directive for launching the loadSequence modal dialog.

- *helmCanvas*: This is the AngularJS directive registered with the name “helm-canvas”. When AngularJS bootstraps and encounters the “helm-canvas” element in the DOM, it automatically instantiates the directive. The directive is self-contained, reusable and is restricted to use as an HTML element. The directive specifies an AngularJS template and this replaces the “helm-canvas” tag in the HTML. The template is described in detail below.

- *contextmenu.js:* This manages interactions with the canvas through the context menu generated on mouse right-click.

- *monomer.js:* This is the AngularJS directive for displaying the monomers on the monomer pallette. It loads the html content from the monomer.html

Templates

- *helmcanvas.html:* The entire contents of the template is replaced in the DOM when using the directive “helm-canvas”. All the SVG related elements are contained within this template.
- *helmcanvaslower.html:* This template adds a canvas to show alternate view of monomer graph in the lower pane
- *modaldialog.html:* Template used for the ‘Load’ button modal
- *imagemodal.html:* Modal template used for right-click menu option to view a molecular structure image
- *monomer.html:* Template used to select a monomer on a mouse click
- *monomerdetails.html:* Modal template to display monomer details
- *Tablemodal.html:* Modal template for right-click option to view a table of molecular properties
- *viewmodal.html:* Modal template for right-click options that do not require an image or a table

5.6. Directory Structure

Once you have installed all node and bower dependencies, and built the project once, the directory structure is as follows:

- HELM-Editor-UI/ - *root application folder*
  - app/ - *main application folder*
    - common/ - *location of common files*
      - svg_class.js - *support for JQuery-style methods for SVG*
    - images/ - *location for any images*
      - yeoman.png - *default Yeoman image*
      - *arrow-down.png - for pan button*
      - *arrow-up.png - for pan button*
      - *arrow-left.png - for pan button*
      - *arrow-right.png - for pan button*
    - scripts/ - *location of all script files*
      - controllers/ - *location of all controller script files*
        - about.js - *Controller for the About page*
        - habe.js - *Controller for the HAbE page*
        - header.js - *Controller for the navigation bar*
        - main.js - *Controller for the main editor page.*
        - library.js - *Controller for the monomer library*
        - modal.js - *Controller for the modal dialogs*
      - directives/ - *location for all directive script files*
        - helmcanvas.js - *Directive for helm-canvas*
        - modaldialog.js - *Directive for modal-dialog*
        - monomer.js - *Directive for monomer display in the library*
        - contextmenu.js - *Directive for context menu*
      - services/ - *location for all service script files*
        - canvasdisplayservice.js - *Service for CanvasDisplayService*
        - helmconversionservice.js - *Service for HelmConversionService*
        - monomerlibraryservice.js - *Service for MonomerLibraryService*
        - webservice.js - *Service for webService (HELM2WebService interface)*
        - monomerselectionservice.js - Service for MonomerSelectionService
        - helmnotationservice.js - Service for HELMNotationService
      - app.js - *root application script file*
    - styles/
      - helm.css - *HELM-specific CSS rules*
      - main.css - *CSS for the main editor page*
    - templates/
      - helmcanvas.html - *template for the helm-canvas directive*

- modaldialog.html - *template for the modal-dialog directive*
- helmcanvaslower.html - *template for lower pane*
- imagemodal.html - *template for molecular image popup*
- monomer.html - *template for displaying an available monomer or fragment in the monomer library*
- monomerdetails.html - *template for monomer detail popup*
- tablemodal.html - *template for lower pane table display*
- viewmodal.html - *Directive for View-only modal dialogs in the editor*

■ views/
- about.html - *HTML for the about view*
- habe.html - *HTML for the HAbE Editor view (placeholder)*
- main.html - *HTML for the main HELM Editor page*

■ 404.html - *Default error page*
■ DefaultMonomerCategorizationTemplate.xml - *Monomer categorization information*
■ favicon.ico - *favicon*
■ index.html - *root HTML page of full application*
■ MonomerDBGZEncoded.xml - *Monomer library initial file*
■ robots.txt - *for web-crawling*

○ bower_components/ - *Bower dependency install directory*
○ dist/ - *Location of minified and compiled files, for distribution*
○ node_modules/ - *NPM dependency install directory*
○ test/ - *main test folder*
■ e2e/ - *location of all end-to-end (Protractor) test files*
- scenarios/ - *location of actual test files*
  - views.js - *all e2e tests are in this file currently*
- protractor.conf.js - *Protractor config file*

■ spec/ - *location of all unit tests*
- controllers/ - *location of all unit tests for controllers*
  - about.js - *unit tests for the About page*
  - habe.js - *unit tests for the HAbE page*
  - header.js - *unit tests for the navigation bar*
  - main.js - *unit tests for the main editor page.*
  - library.js - *unit tests for the monomer library*
  - modal.js - *unit tests for the modal dialogs*
- directives/ - *location of all unit tests for directives*
  - helmcanvas.js - *helm-canvas directive unit tests*
  - modaldialog.js - *modal-dialog directive unit tests*
  - monomer.js - *monomer directive unit tests*
  - contextmenu.js - *Directive for context menu*
- services/ - *location of all unit tests for services*

        - canvasdisplayservice.js - *CanvasDisplayService unit tests*
        - helmconversionservice.js - *HelmConversionService unit tests*
        - monomerlibraryservice.js - *MonomerLibraryService unit tests*
        - webservice.js - *webService unit tests*
        - *helmnotationservice.js - HelmNotationService unit tests*
        - *monomerselectionservice.js - MonomerSelectionService unit tests*
      - .jshintrc - *jshint configuration file for tests specifically*
      - karma.conf.js - *Karma configuration file*
    - .bowerrc - *Bower configuration file*
    - .editorconfig - *Default editor configuration file*
    - .gitignore - *Git ignore settings*
    - .jscsrc - *JSCS configuration file*
    - .jshintrc - *JSHint configuration file*
    - .travis.yml - *Travis build file*
    - .yo-rc.json - *Yeoman configuration file*
    - bower.json - *Bower dependencies*
    - Gruntfile.js - *Grunt targets and configuration*
    - LICENSE - *MIT License file*
    - package.json - *NPM dependencies*
    - README.md - *README file*

5.7. User Manual

1. Launch the hosted application through any modern browser: http://104.236.250.11/editor/dist/.
2. To load a sequence:
    a. Click the “Load” button to launch the Load Sequence popup dialog.
    b. Input the sequence, and select the sequence type.
    c. Click Submit.
    d. The graphical representation of the sequence will be displayed in the canvas.
    e. Sample sequences are provided, with their results:
        i. RNA/DNA
            1. Input: ATG
            2. Output:

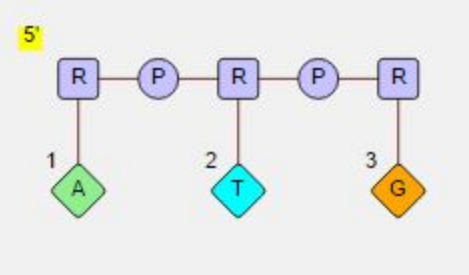

Figure 15: Graphical representation of a Nucleotide sequence

ii. PEPTIDE
    1. Input: ATG
    2. Output:

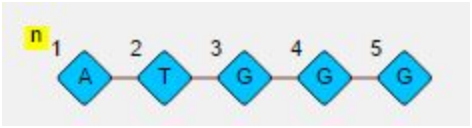

Figure 16: Graphical representation of a Peptide sequence

iii. HELM Sequence - there are various sequences provided on the load screen
    1. Input: PEPTIDE1{R.Y.F.L.W.V.F.P.L}$PEPTIDE1,PEPTIDE1,9:R2-1:R1$$$
    2. Output:

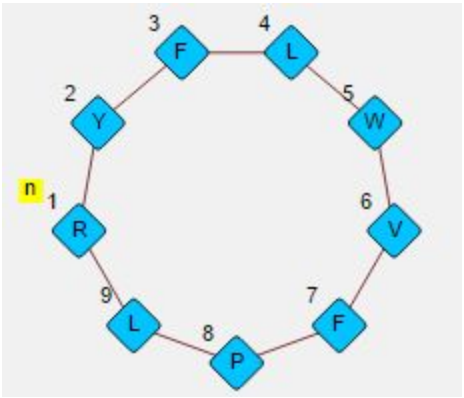

Figure 17: Graphical representation of a cyclic Peptide sequence

3. View Monomer Library (Milestone 3)
    a. Select a polymer from a tab of available types .
    b. Select one or more monomer groups or sub-groups from a corresponding list that is dynamically generated.
    c. Select a monomer from a list of monomers to access its information. The monomer palette displays the monomers that are retrieved from the monomer library.
    d. Double click on a single monomer in the palette to open a pop-up with additional details about the monomer.
4. Edit Sequence (Milestone 3)
    a. Drag or click on a single monomer from the monomer palette to the canvas
5. Save Sequence (Milestone 3)
    a. Right click and select the ‘Save’ option to download the HELM notation or canonical HELM notation to a file.
6. Copy Sequence (Milestone 3)
    a. Right click and select the ‘Copy’ option to copy the HELM notation or canonical HELM notation to the clipboard

5.8. HELM Notation Decoded

The HELM notation renders biomolecules composition and structure in a machine-readable format. This diagram below explains the HELM notation:

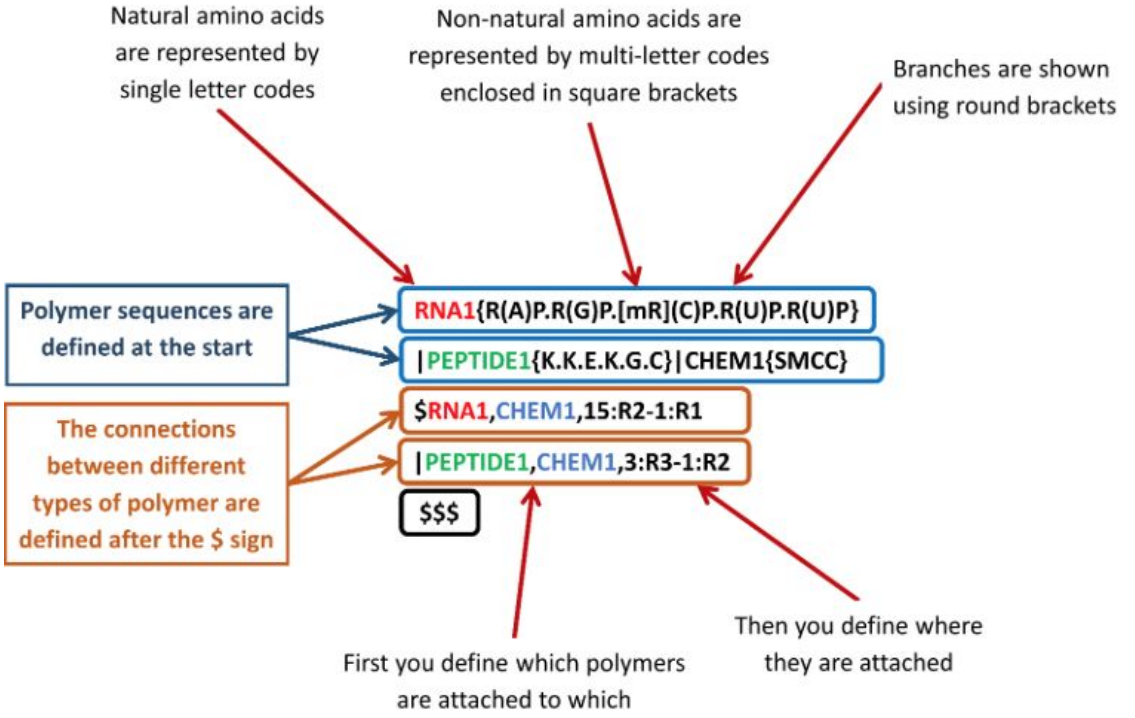

Figure 18: Diagram explaining the HELM Notation

6. References

1. HELM Project overview: https://pistoiaalliance.atlassian.net/wiki/download/attachments/8716303/Short_form_HELM_Leaflet%202015_08_20%20docx.pdf?version=1&modificationDate=1440414807207&api=v2
2. Overview and history of HELM http://pubs.acs.org/doi/full/10.1021/ci3001925
3. AngularJS Tutorials: https://docs.angularjs.org/tutorial/
4. Code Project Article: Implementing-a-Flowchart-with-SVG-and-AngularJS
5. SVG Tutorials: https://developer.mozilla.org/en-US/docs/Web/SVG/Tutorial
6. SVG blog: http://www.petercollingridge.co.uk/svg-tutorial/svgs-web
7. StackOverflow: Placing SVG shapes in a cycle
8. Modal Dialog in Angular by Adam Albrecht: http://adamalbrecht.com/2013/12/12/creating-a-simple-modal-dialog-directive-in-angular-js/

9. XML to JSON:
   http://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html