Reactome Advanced Knowledgebase Search and Pathway Visualization Prototype App

by

Ryan Vereque

A Major Qualifying Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Computer Science

by

May 2020

APPROVED:

Smith, Therese

Abstract

The goal of this project is to prototype an app that allows users to search and visualize the Reactome knowledgebase. The Reactome ontology contains biological pathways, their components, and interactions. The app allows the user to construct search criteria across multiple ontological concepts and their specific relationships, whereas existing Reactome search tools cannot. The app also re-creates pathway visualizations. This work could serve as a basis for visualizing these pathways in custom use cases not met by the official tools.

Contents

List of Figures

Chapter 1

Problem Description

Reactome is a knowledgebase specializing in biomolecular pathways [FKV+18]. The Reactome project also provide a variety of bioinformatics tools including but not limited to a full-text search tool [FSG+16], an API for programmatic access to the knowledgebase [Reab], and a pathway diagram viewer [Reaf].

Searching the Reactome knowledgebase using the available tools for entries that relate to another set of entries via specific ontological relationships can be a laborious task when additional search criteria is placed on either set of entries. If a search task involves identifying a set of entries that are further ontologically related to the results of the previous task, then the number of search queries and manual involvement required quickly grows unreasonable without automation. These search-related limitations of the existing set of tools is addressed in the prototype app developed for this project.

Furthermore, Reactome's pathway diagram viewer does not feature the ability to have a user modify a pathway's contents or stylize them in any customized manner. The prototype app in this project aims to reproduce some of the visual contents from the existing pathway viewer, and in doing so will address challenges associated with gathering and processing diagram data that would be needed for implementing a new and enhanced pathway diagram viewer.

Chapter 2

Related Work

The Reactome knowledgebase follows a formal data schema [VDS+07] than can be interpreted as an ontology. This ontology delineates concepts including chemical reactions, genes and their products, cell compartments, diseases, and pathways and their regulation, among other things [Reac, Read].

2.1 Existing Reactome Search Tools

The full-text search tool provided by Reactome allows the user to search for entries in the knowledgebase by specifying a search string and selecting any number of options for facets [FSG+16]. Some examples of these facets include a general category of ontological concepts the entry belongs to, the compartment within a cell that the entry is associated with, the species that the entry is associated with, and a set of predefined keywords it is associated with.

The search functionality provided by this existing Reactome tool is useful for a variety of use cases, but is lacking in some respects. Many use cases for searching the Reactome knowledgebase may require the ability to explore entries that are ontologically related by some specific concept to a first entry. One such realistic use case is a user wanting to search for pathways that contain a given gene product and that also meet a certain search string. To find entries that are associated with another (set of) entr(ies) in a specific ontological relationship is not possible through the aforementioned full-text search tool.

There are other less user-friendly tools provided by Reactome that may partially aid a user with these kinds of use cases. One such tool is the Content Service API which can find entries that are related to a single entry for a certain limited set of relationships, including the relationship that would be relevant for finding pathways that contain a gene product [Reac]. However, further filtering this set of pathway entries with a search-string is non-trivial because this set of results would have to be intersected with a set of results from the a full-text search in highly manual process.

As the use case increases in complexity, the benefits of automating this process become clearer. Consider a use case where the user wants to search for all pathways that contain any of the reactions from a given initial pathway. This would require one use of the API to find reactions contained in the given pathway, then another use of the API for each reaction to find the pathways that contain that reaction [Reaa]. The search process would be even more complex if either these intermediate reactions or the final pathways needed to satisfy criteria about what species or cell compartment they exist in. While these searches queries could be fulfilled by directly querying Reactome's publicly available relational or graph databases, the technical knowledge of syntax required for forming these queries, along with it taking place outside the web browser, limits its target audience as a search tool.

2.2 Existing Pathway Diagram Viewer Tool

Another tool provided by the Reactome project is the pathway diagram viewer which displays curated visualizations of pathways [Reaf]. These diagrams represent networks of reactions including their inputs, outputs, catalysts, and other related concepts each represented with their own node in the diagram. The reactions in these diagrams are all of the ontological concept "Reaction-Like-Event" (RLE). A RLE is represented via “edges” with multiple tails connected to instances of its previously mentioned related concepts. Not all possible ontological relationships and entries are included in a pathway diagram, instead the diagram mostly consists of RLEs and their constituent Physical Entities (PEs). These diagrams also feature interactivity including the ability to view more information on a selected visual element and search for elements within a diagram using a search string [FKV$^+$18]. It should also be noted that there is a distinction between detailed pathway diagrams (which are also dubbed as “low-level” diagrams in the API documentation) and enhanced high-level diagrams (EHLD) [Reae, Reaa].

These diagrams do not, however, allow elements to be re-positioned, included, or excluded. It also does not allow the customization of visual styles used for different diagram elements. These claims are evidenced by that fact of there being no mention to such features in the collection of papers published by the Reactome project under their regular naming convention of “The reactome pathway knowledgebase” [CMH$^+$14, FSG$^+$16, FJM$^+$18, JMV$^+$20] and in the latest pathway diagram focused paper [FSV$^+$18] which collectively go back 2014. This dates back some time before the last major version release (version 3) of the pathway diagram viewer in 2015 [FSG$^+$16]. It also evidenced by no mention in pathway diagram related documentation [Reae, Reaf]. One use case where such functionality may be useful is where a user wishes to create an alternative interpretation of what reactions are relevant to a given biological pathway. While this use case is beyond the scope of the prototype app, it could theoretically involve the user starting from an existing diagram and then including searched-for RLEs that connect to Physical Entities already present in the diagram, extending the network.

2.3 How this Project Builds upon Existing Work

Developing the aforementioned desirable features in a piece of software may require several iterations of design. To make progress towards this goal, a prototype web app was created in this project. It implements a UI with some relational search features and some underlying logic necessary for future pathway diagram features.

This prototype app is not hosted on a web-server publicly accessible on the web. The source-code is available, however, and the app may be hosted locally. The source-code may be found at:

`https://github.com/kicknickr/reactome-search-vis-prototype`

See appendix chapter A for more information.

Chapter 3

Overall Design

3.1 Development Environment and Utilized Packages

The development environment for this web app is reliant upon Node.js (Node). Node is “an asynchronous, event-driven JavaScript runtime that offers a powerful but concise standard library”. It is has a built-in package manager named NPM which allows for the inclusion of publicly accessible packages into a node project [You17].

Notable packages that provide additional context to the app’s development, without significantly influencing import design choices, include Babel and Parcel. “Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments” including the features to transform syntax, polyfill features that are missing in your target environment, and perform source code transformations [Bab]. Parcel is an asset-bundler for web apps with many automatically configured features for optimization and supports JavaScript transformers and transpilers such as Babel [Par].

Some notable packages aiding the construction of the UI and its logic include D3.js (D3) version 5 and Material Web Components version 5. D3 “is a JavaScript library for manipulating documents based on data” [D3]. Material Web Components is package designed for web apps that wish to execute Material Design, a UI design system developed by Google [Mata]. “Material is an adaptable system of guidelines, components, and tools that support the best practices of user interface design” [Matb]. The Material Design system was not strictly executed in this project’s app, however UI components provided by the package where utilized wherever possible. Aside from the pathway diagram viewer portion of the UI, the remainder of the app has a look and feel that could be primarily associated with Material Design.

3.2 General User Interface Layout

The web app divides the screen space in two. A large and central pathway diagram viewport, and a header containing access to general information . Upon the user initiating a context menu on empty space within the viewport, this new menu gives the user a choice to open a search pane. This search pane allows a user to construct search criteria for entries in the knowledgebase.

Update

3.3 User Interface and Features

3.3.1 App Header

talk about header

3.3.2 Pathway Diagram Viewer

The app’s diagram viewport is where a rough re-creation of any Reactome pathway that has a low-level diagram can be displayed. Figure 3.1 shows the app’s recreation of pathway R-HSA-69239 or “Synthesis of DNA”. The app’s diagram differs from the existing Reactome pathway diagram viewer (Figure 3.2) in a few key ways. Firstly, RLEs are represented with their own nodes, rather than by multi-tailed edges. Edges now represent a link between a RLE and just one of its constituent PEs, rather than representing an entire RLE itself. These new edges do not denote in what way a PE contributes do a given RLE. Secondly, groupings of diagram elements denoted by shadows, seen as translucent boxes, featured in the existing diagram viewer have not been implemented in this app, although it should be noted the data for their positions was found to exist in the API. Thirdly, this app features no interactivity with diagram elements. The two diagram viewers do, however, share some features. Notably including similar styling for PE nodes, the same relative locations for elements in the diagram, and the ability to zoom and pan through the viewport.

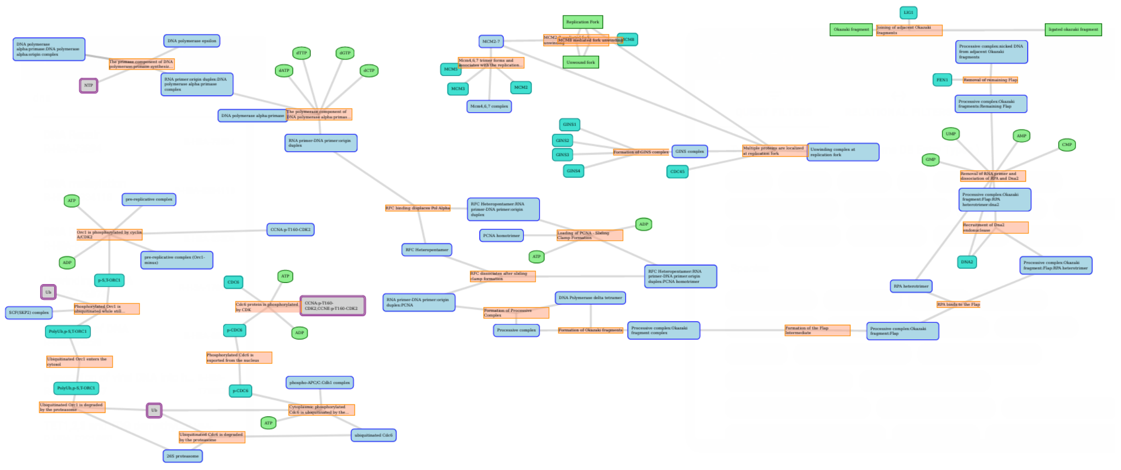


Figure 3.1: Pathway diagram viewer opened to pathway R-HSA-69239, known as “Synthesis of DNA”

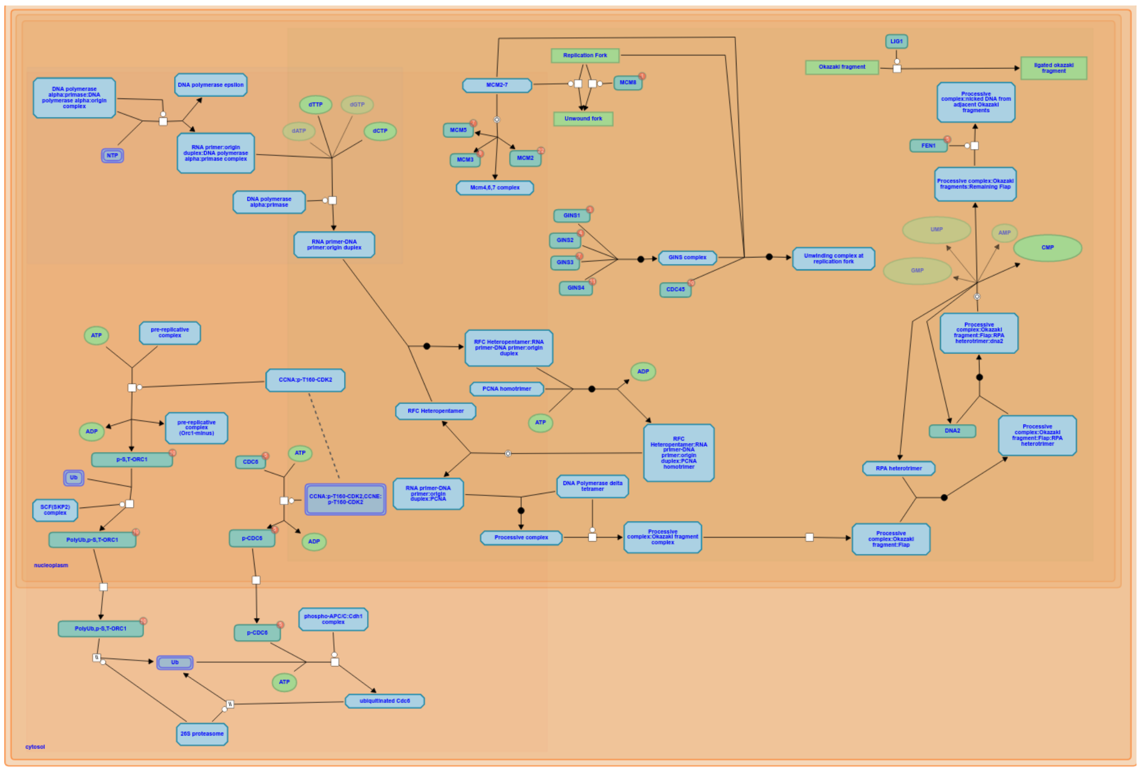
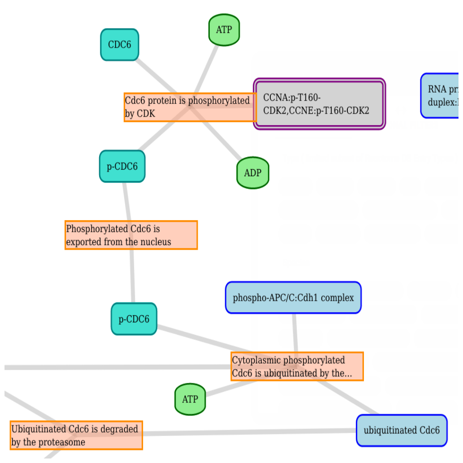
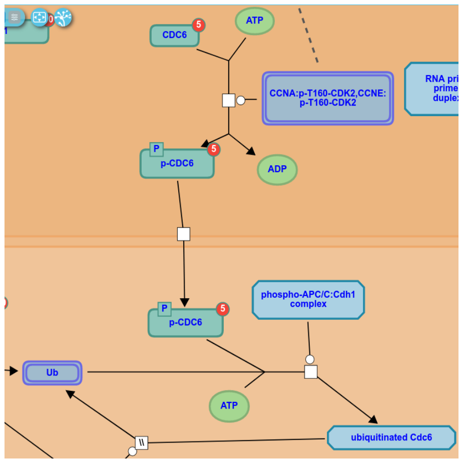


Figure 3.2: Reactome’s pathway diagram viewer opened to same pathway as figure 3.1


CDC6
ATP
Cdc6 protein is phosphorylated by CDK
CCNA:p-T160-CDK2,CCNE:p-T160-CDK2
RNA pr
duplex:
p-CDC6
ADP
Phosphorylated Cdc6 is exported from the nucleus
phospho-APC/C:Cdh1 complex
p-CDC6
Cytoplasmic phosphorylated Cdc6 is ubiquitinated by the...
ATP
Ubiquitinated Cdc6 is degraded by the proteasome
ubiquitinated Cdc6

(a) This project's app



(b) Reactome's existing tool

Figure 3.3: A side-by-side comparison between closeups of pathway diagram viewers from this project's app and Reactome's existing tools. The portion of pathway being displayed is from the same pathway as figures 3.1 and 3.2

3.3.3 Search Feature

The search pane appears as a window that can be re-positioned, resized, closed, and pinned. Through user interaction in the search pane, more search panes may be produced for further refining search criteria and for specifying relational search criteria. The simple search pane can be accessed via a context menu on the diagram viewport (see Figure 3.4). The simple search pane serves two functions. To specify a search string, and to view a list of results for the search. To expedite debugging and testing of the simple search pane, a hard limit of 100 entry search results are accepted from queries to the Reactome API.

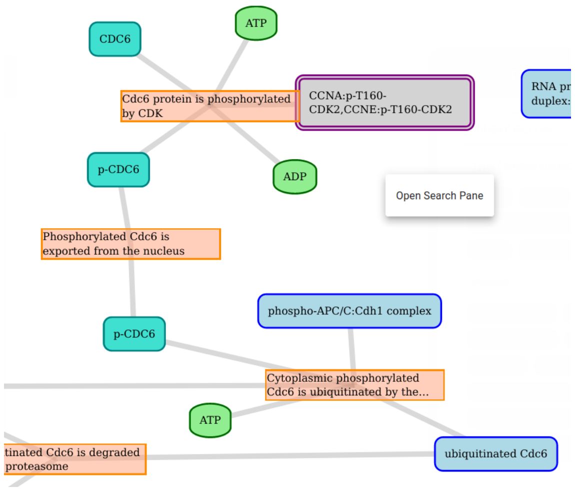

CDC6
ATP
Cdc6 protein is phosphorylated by CDK
CCNA:p-T160-CDK2,CCNE:p-T160-CDK2
RNA pr duplex:
p-CDC6
ADP
Open Search Pane
Phosphorylated Cdc6 is exported from the nucleus
phospho-APC/C:Cdh1 complex
p-CDC6
Cytoplasmic phosphorylated Cdc6 is ubiquitinated by the...
ATP
tinated Cdc6 is degraded proteasome
ubiquitinated Cdc6

Figure 3.4: The same closeup from Figure 3.3 of this project's pathway diagram viewer, but with a context menu having been opened on empty space

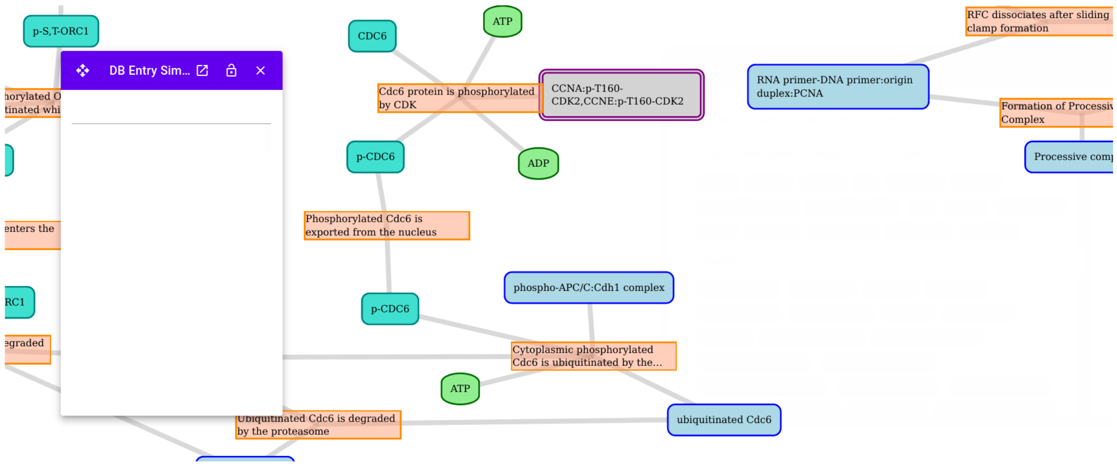

DB Entry Sim...
p-S,T-ORC1
CDC6
ATP
Cdc6 protein is phosphorylated by CDK
CCNA:p-T160-CDK2,CCNE:p-T160-CDK2
p-CDC6
ADP
Phosphorylated Cdc6 is exported from the nucleus
p-CDC6
phospho-APC/C:Cdh1 complex
Cytoplasmic phosphorylated Cdc6 is ubiquitinated by the...
ATP
ubiquitinated Cdc6
Ubiquitinated Cdc6 is degraded by the proteasome
RFC dissociates after sliding clamp formation
RNA primer-DNA primer:origin duplex:PCNA
Formation of Processiv Complex
Processive comp

Figure 3.5: The simple search pane appears once the “Open Pane” option has been selected

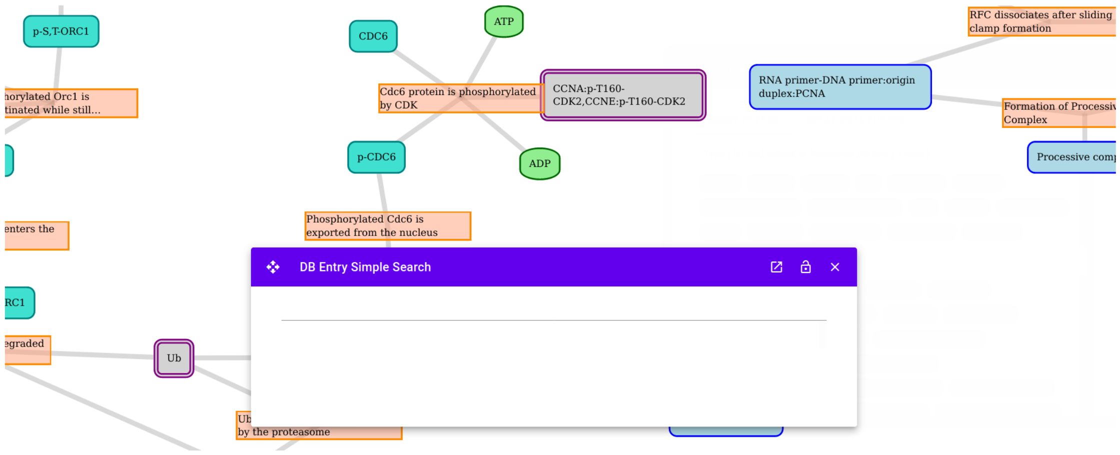
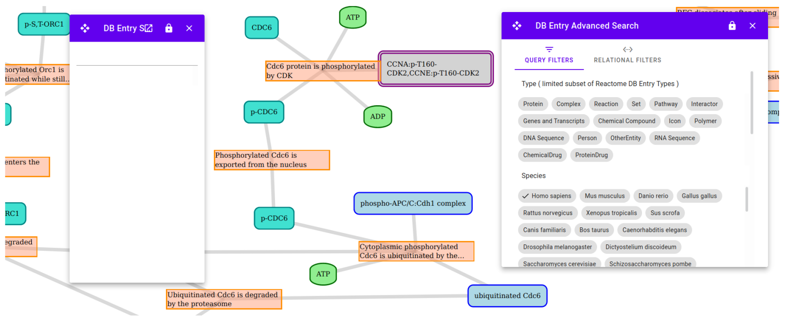

p-S,T-ORC1
horylated Orc1 is tinated while still...
CDC6
ATP
Cdc6 protein is phosphorylated by CDK
CCNA:p-T160-CDK2,CCNE:p-T160-CDK2
p-CDC6
ADP
Phosphorylated Cdc6 is exported from the nucleus
DB Entry Simple Search
Ub
by the proteasome
RFC dissociates after sliding clamp formation
RNA primer-DNA primer:origin duplex:PCNA
Formation of Processiv Complex
Processive comp

Figure 3.6: Search panes can be resized and dragged

From the simple search pane, a button to open the advanced search pane is available. The advanced search pane provides two mechanisms which work simultaneously for further constructing search criteria.

The first mechanism appears on the advanced search pane’s “Query Filters” tab. This tab allows the selection of options from multiple search facet categories. This tab, along with the simple search pane, replicates the functionality of the existing

Reactome full-text search tool. For examples of a multi-choice selection of facet options from a set of different facet categories, one may see them featured in the product search interface for many online shopping websites. For each category of facet, the user may select any number of options. If none are selected, no additional search criteria are imposed. If one or more options are selected, the returned search results must belong to one of the selected options. Search results must satisfy this for all categories of facets, and not just any of them. The categories of facets implemented include: ontological concept category (displayed as "Type"), associated species, associated cell compartment, and associated predefined keywords.



p-S,T-ORC1
DB Entry S
CDC6
ATP
Cdc6 protein is phosphorylated by CDK
CCNA:p-T160-CDK2,CCNE:p-T160-CDK2
p-CDC6
ADP
Phosphorylated Cdc6 is exported from the nucleus
phospho-APC/C:Cdh1 complex
p-CDC6
Cytoplasmic phosphorylated Cdc6 is ubiquitinated by the...
ATP
Ubiquitinated Cdc6 is degraded by the proteasome
ubiquitinated Cdc6
DB Entry Advanced Search
QUERY FILTERS
RELATIONAL FILTERS

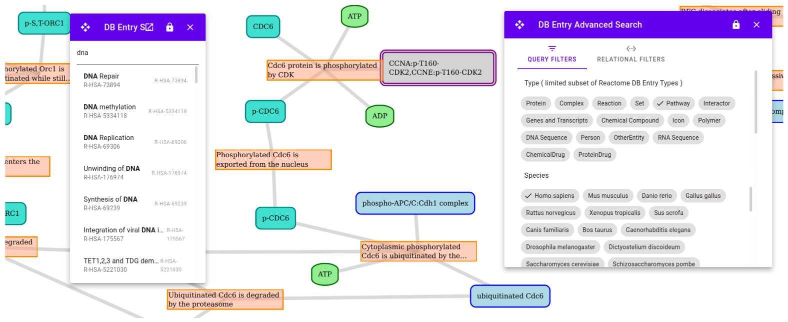
Type (limited subset of Reactome DB Entry Types)

Protein, Complex, Reaction, Set, Pathway, Interactor, Genes and Transcripts, Chemical Compound, Icon, Polymer, DNA Sequence, Person, OtherEntity, RNA Sequence, ChemicalDrug, ProteinDrug

Species

✓ Homo sapiens, Mus musculus, Danio rerio, Gallus gallus, Rattus norvegicus, Xenopus tropicalis, Sus scrofa, Canis familiaris, Bos taurus, Caenorhabditis elegans, Drosophila melanogaster, Dictyostelium discoideum, Saccharomyces cerevisiae, Schizosaccharomyces pombe

Figure 3.7: The initial state of the advanced search pane and its "Query Filters" tab



p-S,T-ORC1
DB Entry S
CDC6
ATP
Cdc6 protein is phosphorylated by CDK
CCNA:p-T160-CDK2,CCNE:p-T160-CDK2
p-CDC6
ADP
Phosphorylated Cdc6 is exported from the nucleus
phospho-APC/C:Cdh1 complex
p-CDC6
Cytoplasmic phosphorylated Cdc6 is ubiquitinated by the...
ATP
Ubiquitinated Cdc6 is degraded by the proteasome
ubiquitinated Cdc6
DB Entry Advanced Search
QUERY FILTERS
RELATIONAL FILTERS

dna

DNA Repair R-HSA-73894 — R-HSA-73894
DNA methylation R-HSA-5334118 — R-HSA-5334118
DNA Replication R-HSA-69306 — R-HSA-69306
Unwinding of **DNA** R-HSA-176974 — R-HSA-176974
Synthesis of **DNA** R-HSA-69239 — R-HSA-69239
Integration of viral **DNA** i... R-HSA-175567 — R-HSA-175567
TET1,2,3 and TDG dem... R-HSA-5221030 — R-HSA-5221030

Type (limited subset of Reactome DB Entry Types)

Protein, Complex, Reaction, Set, ✓ Pathway, Interactor, Genes and Transcripts, Chemical Compound, Icon, Polymer, DNA Sequence, Person, OtherEntity, RNA Sequence, ChemicalDrug, ProteinDrug

Species

✓ Homo sapiens, Mus musculus, Danio rerio, Gallus gallus, Rattus norvegicus, Xenopus tropicalis, Sus scrofa, Canis familiaris, Bos taurus, Caenorhabditis elegans, Drosophila melanogaster, Dictyostelium discoideum, Saccharomyces cerevisiae, Schizosaccharomyces pombe

Figure 3.8: The facet category for ontological type has had the "Pathway" option chosen

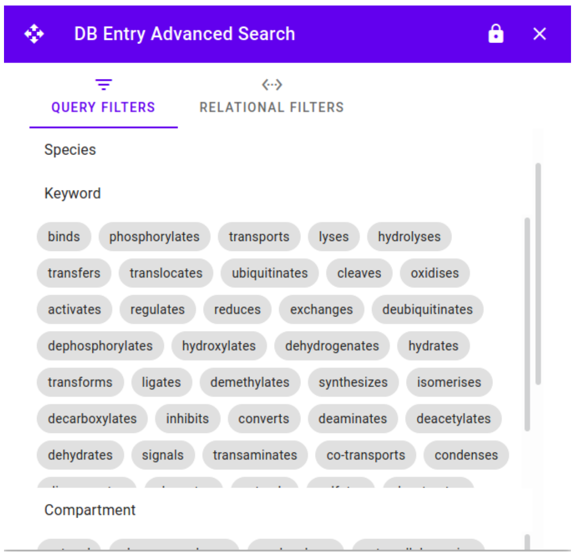
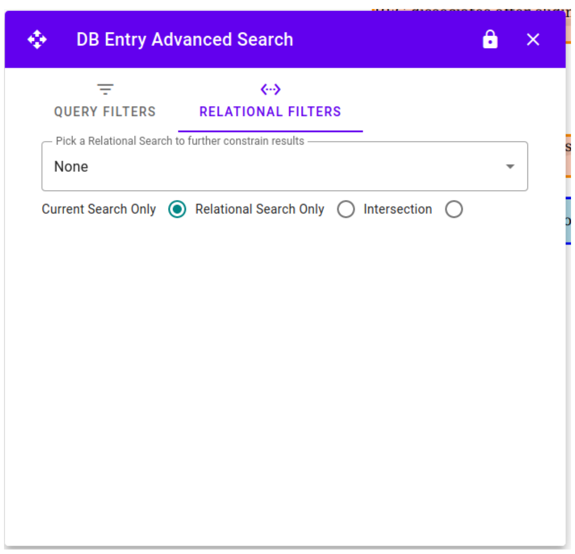


Figure 3.9: Facets categories can be expanded/contracted and scrolled-through

The second mechanism appears on the “Relational Filters” tab (see Figure 3.10). The functionality on this tab and the functionality provided by the additional UI components that it generates is not available in the existing Reactome search tools with UI. It constitutes the bulk of this project’s contribution to advancing search functionality for Reactome’s knowledgebase. This tab provides the ability for a set of search result entries returned from the current search pane’s criteria to be further narrowed down by whether they satisfy a specified ontological relationship with one of the search results from another search pane.



Pick a Relational Search to further constrain results: None

Current Search Only [✓ ●] Relational Search Only ○ Intersection ○

Figure 3.10: The initial state of the “Relational Filters” tab

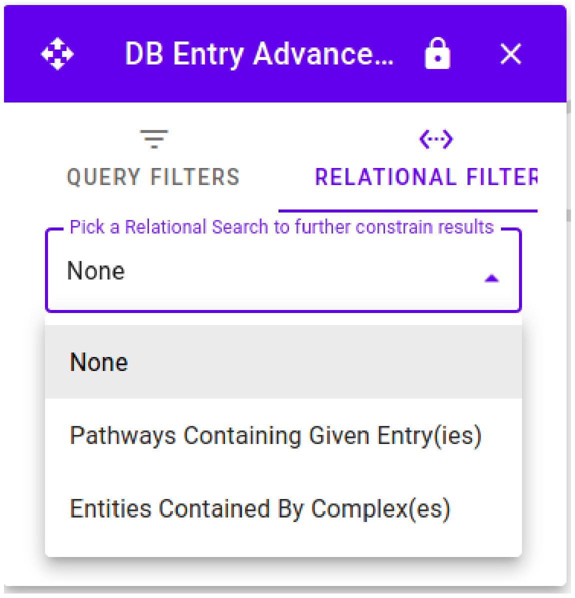

DB Entry Advance...
QUERY FILTERS
RELATIONAL FILTER
Pick a Relational Search to further constrain results
None
None
Pathways Containing Given Entry(ies)
Entities Contained By Complex(es)

Figure 3.11: Ontological relationship filter options

On this tab, there is a drop-down menu for selecting an ontological relationship, shown in Figure 3.11. There is also a radio button set for selecting whether the new set of results have to satisfy the relational search criteria, the original criteria sans relations, or both. The following search logic will be described under the assumption that the both option ("Intersection" option in the actual app's UI) is selected. Upon making appropriate selections to these two fields, a second set of search panes is created (see Figure 3.12). The new intersected search results are displayed in the first simple search pane and must now have a specified ontological relationship with one selected entry from the second simple search pane (see Figure 3.13). It should be noted the because of a entry result size limitation discussed previously, this intersection of results is also similarly incomplete. Furthermore, this second set of search panes may further refine their search results by also using their own "Relational Filters" tab, thereby producing a third set of panes, and etc.

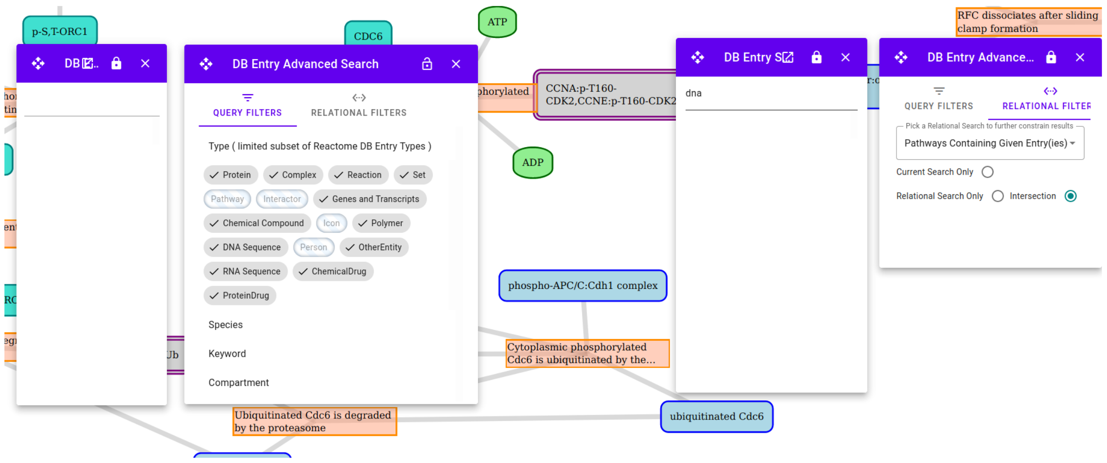


Figure 3.12: A second set of search panes has opened from the chosen relational filter. These search panes have been re-positioned within the diagram for visual clarity. The two panes on the right are the originals, and the two panes of the left have just been created. Notice that in the new advanced search pane, several type facet options have had their selection disabled, while the remainder have been selected by default. Different facet options are disabled according to their compatibility with the chosen relational filter.

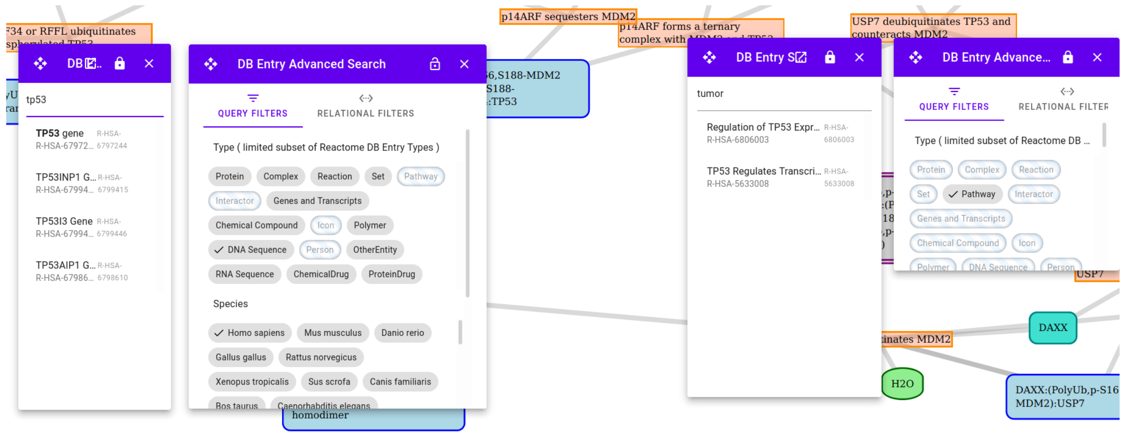


Figure 3.13: An example of the relational search filter in action. A summarization of important search criteria for the displayed search could be "Human pathways containing the search-string "tumor" that contain the selected DNA Sequence "TP53 gene"". The results of this search are displayed in the first (right-side) simple search pane. The first of these two pathway results was selected and then became displayed in the pathway diagram viewer.

This relational searching feature allows users to build a non-branching chain of search queries. Each step in the chain represents finding the set of knowledgebase entries which satisfy a specified ontological relationship with a single selected entry found in the previous step. Previously existing non-relational search functionality (via the "Query Filters" tab), is also specifiable at each step in the chain. As previously mentioned, both search criteria mechanisms operate simultaneously.

In this prototype app, only two ontological relationships are implemented (see Figure 3.11). The filtering mechanism and UI logic is independent of search results. Implementing additional relational filters for the search pane will only require new logic for how the Content Service API can be leveraged to gather all entries that have the desired relationship to a single given entry.

3.4 Data Structures and Algorithms

The only external web requests this app produces are those to Reactome's Content Service API. To make these calls, the `ReactomeAPIQuerier` and `ReactomeRequestProcessor` classes exist. Each of `ReactomeAPIQuerier`'s methods point to a unique end point in the API and transform parameters that are supplied programmatically into parameters formatted to be understood by the API. The `ReactomeRequestProcessor` manages the logic of caching some requests and formatting results in a way that is most usable by other classes concerned with search mechanism logic.

The `ISearchPresManager` class acts as an interface for classes responsible for bridging the gap between UI logic and search mechanism logic. For example, the `PaneComponent` class, responsible for a search pane's window-like features, and the `CustomMDCChipSetComp`, responsible for the facet option selection chips, do not talk directly with implementations of the `BaseSearchComponent` class. This interface has methods for displaying/clearing result data, enabling/disabling the ability to select search criteria including individual facet options, and gathering user-entered search criteria, among other things.

The `BaseSearchComponent` is an abstract base class that outlines the logic of a single search and the general approach for how other searches may narrow and expand its result set. Implementations of `BaseSearchComponent`, such as `PresentingQueryBasedSearchComp`, may instantiate more implementations of `BaseSearchComponent`. When a user selects an ontological relationship in the "Relational Filters" tab, `PresentingQueryBasedSearchComp` must instantiate one new instance of `PresentingQueryBasedSearchComp` for the new search pane. This new instance will potentially instantiate one or more instances of a different implementation of `BaseSearchComponent` for each search result in that new pane. These instances are dubbed “sub-searches” and each will be constructed with a result from the new search pane and find the knowledgebase entries that are related via the specified ontological relationship. One such example of a `BaseSearchComponent` “sub-search” is `NonPresentingPathwaysContainingEntrySearchComp` which finds all pathways that contain a given entry. `BaseSearchComponent` is also responsible for instantiating implementations of `ISearchPresManager` and making calls to them with new search results as they are received and processed in ways described previously.

Chapter 4

Results and Conclusions

The web app developed in this project can serve as a prototype for expanding on the set of features provided by Reactome's tools. The app allows for arbitrarily long chains of relational searches on Reactome's knowledgebase entries to be constructed by the user. Albeit, currently limited to a small selection of ontological relationships. It also implements some of the logic necessary to replicate existing low-level pathway diagrams using only API data. That is, without the need to reverse engineer HTML canvas contents from existing Reactome pathway diagram viewer pages, and then infer the presence of knowledgebase entries and their diagram positioning data.

The app was not hosted on a public web-server and also did not rely on hosting a custom API for data needed by the app. In other words, cached content for queries to Reactome's APIs was client-side and not server-side. Despite this, the user interface is responsive and search retrieval is relatively quick, with no apparent cap on request frequency. This suggests that a more feature complete app would also not necessarily need a custom data API to achieve desirable performance and responsiveness.

No full-featured web frameworks, UI frameworks, or combinations thereof, such as React.js or AngularJS, were used in development. These types of frameworks could have accelerated the development process had they been used, especially for apps with somewhat complex UI state logic like this one. Not using these frameworks served as an educational opportunity for what their benefits may be. Over the course of the development process, and as the complexity of the UI increased, the design patterns that emerged were motivated by the same technical challenges these frameworks specialize in addressing. One such challenge is the exchanging of data and callback functions between hierarchically-arranged UI logic components and furthermore the life-cycle of said components as they compute data, pass data up and down the hierarchy, and make changes to a browser's document object model (DOM) [Mai]. The degree of this challenge was not initially anticipated.

This prototype app also addresses many of the technical challenges that are involved with gathering the data for relational searches and reproducing pathway diagrams. It also demonstrates the responsiveness that a web app can have despite the presence of many diagram elements and rapid sequences of tens of API calls.

In conclusion, this project's prototype app demonstrates that the development of an app satisfying new use cases in the areas of relational searches and customized diagrams of Reactome's knowledgebase is an endeavor that can be realistically approached through a web app based solution without a significant need for a back-end API.

Chapter 5

Limitation and Future Work

5.1 App Usage Instructions

Usage instructions are important for software that reference domain-specific knowledge or other abstractions that are particular to that app's solution to design goals. An example of that in this project's app would be the concept of a relational filter in a search query. Using such software correctly is usually not intuitive, potentially unless the user has read about the motivation behind the features in a corresponding research paper or other resource. In order to remedy this need, the app could include general usage instructions accessible through buttons provided in the app's header section , along with short relevant tooltips (in some cases in the form of an icon) present somewhere on many or most UI components. Update

5.2 General Set Operations Between Search Result Sets Instead of Chains of Intersections

One significant limitation of the app's search logic is the life cycle of the search logic component, `BaseSearchComponent`. In this component, relational searches must be conceptualized as the intersection between the results of the current search's non-relational criteria, and the results of just one other high-level search which may itself involve sub-searches. This is the case because having more than one high-level search to intersect results with would require an algorithm to determine what order search components are evaluated. Furthermore, it is easy for the user lose track of which set of search panes are the originals, and which are the new ones that are implicitly using a sub-search as defined by the first set's choice of relational filter. A mechanism in the UI of differentiating these sets of panes was not implemented.

A better design would not intertwine the displaying of search results and the combination of multiple search criteria in this way. One ambitious design will be roughly outlined. The user could have the ability to construct a directed acyclic graph (DAG) of search criteria. Each node contains a collapsible search pane that determines either some non-relational search criteria and/or some set operation on incoming result set(s). Each edge either simply forwards a result set from the outgoing to incoming nodes, or produces a new result set that represents entries meeting a specified ontological relationship with those entries from the outgoing node. Each edge also specifies whether one chosen or all results from the outgoing node are to be considered. An algorithm could transverse this DAG in such a way to only compute a node's result set when all of its incoming neighbors have already had theirs computed. Finally, for any node in this DAG, but especially those with no outgoing edges, the working search results can be viewed and interacted with. This interaction should still include the ability for entry results that are pathways to be displayed in the diagram viewport.

5.3 Searches Using Selected Diagram Elements

Pathways returned in search results may be displayed in the diagram viewport, but elements in the diagram viewport cannot be interacted with and do not influence searches. Because all elements in the diagram are knowledgebase entries, a set of selected diagram elements could be used by search mechanism components in a similar way as results from a set of search panes. This suggests that the usage of search pane results and selected diagram elements could in some way be interchangeable, if the search mechanism were refactored. Doing so would yield the potential to initiate a search based on selected diagram components.

5.4 A Full-Featured Diagram Editor

As discussed previously in chapter 1, diagram customization may be a desirable feature. A feature in this area on the higher end of development complexity is a diagram editor. A diagram editor could allow the user to move diagram nodes around, add nodes into the diagram from search pane entry results, control the level of informational detail on each node, group nodes together for moving them around in a consolidated fashion, save the diagram state for later usage, or export that state in the same format as that provided by Reactome's API. This set of features would be a minimum in order to be considered a diagram editor for Reactome. Optional features may include customizing the styling of different diagram elements and creating new nodes from new entities with user-provided properties.

5.5 Using a Database in Addition to / Instead of the API

Reactome's Content Service API only allows for querying entries that are related to a given entry for a limited set of ontological relationships. To expand coverage for what kind of ontological relationships can be implemented as relational search filters, either Reactome's relational database or graph database could be leveraged in addition or instead of their Content Service API. This would, however, necessitate the creation of a custom server-side API for retrieving specific queries on a server-side hosted Reactome database.

Reactome's graph database would provide several advantages over their relational database in the areas of query complexity and response times [FKV+18]. Furthermore, the translation of user -constructed relational search criteria into database queries would be more syntactically organic for a graph database than for a relational database, despite the name.

Bibliography

[Bab] Babel documentation. `https://babeljs.io/docs/en/`.

[CMH+14] David Croft, Antonio Fabregat Mundo, Robin Haw, Marija Milacic, Joel Weiser, Guanming Wu, Michael Caudy, Phani Garapati, Marc Gillespie, Maulik R. Kamdar, Bijay Jassal, Steven Jupe, Lisa Matthews, Bruce May, Stanislav Palatnik, Karen Rothfels, Veronica Shamovsky, Heeyeon Song, Mark Williams, Ewan Birney, Henning Hermjakob, Lincoln Stein, and Peter D'Eustachio. The reactome pathway knowledgebase. *Nucleic Acids Research*, 42(D1):D472–D477, 2014.

[D3] D3 home page. `https://d3js.org/`.

[FJM+18] Antonio Fabregat, Steven Jupe, Lisa Matthews, Konstantinos Sidiropoulos, Marc Gillespie, Phani Garapati, Robin Haw, Bijay Jassal, Florian Korninger, Bruce May, Marija Milacic, Corina Duenas Roca, Karen Rothfels, Cristoffer Sevilla, Veronica Shamovsky, Solomon Shorser, Thawfeek Varusai, Guilherme Viteri, Joel Weiser, Guanming Wu, Lincoln Stein, Henning Hermjakob, and Peter D'Eustachio. The reactome pathway knowledgebase. *Nucleic Acids Research*, 46(D1):D649–D655, 2018.

[FKV+18] Antonio Fabregat, Florian Korninger, Guilherme Viteri, Konstantinos Sidiropoulos, Pablo Marin-Garcia, Guanming Wu, Lincoln Stein, and Henning Hermjakob. Reactome graph database: Efficient access to complex pathway data. *PLoS Computational Biology*, 14(1):e1005968, 2018.

[FSG$^+$16] A. Fabregat, K. Sidiropoulos, P. Garapati, M. Gillespie, K. Hausmann, R. Haw, B. Jassal, S. Jupe, F. Korninger, S. McKay, L. Matthews, B. May, M. Milacic, K. Rothfels, V. Shamovsky, M. Webber, J. Weiser, M. Williams, G. Wu, and L. Stein. The reactome pathway knowledgebase. *Nucleic Acids Research*, 44(D1):D481–D487, 2016.

[FSV$^+$18] Antonio Fabregat, Konstantinos Sidiropoulos, Guilherme Viteri, Pablo Marin-Garcia, Peipei Ping, Lincoln Stein, Peter D'Eustachio, and Henning Hermjakob. Reactome diagram viewer: data structures and strategies to boost performance. *Bioinformatics*, 34(7):1208–1214, 2018.

[JMV$^+$20] B. Jassal, L. Matthews, G. Viteri, C. Gong, P. Lorente, A. Fabregat, K. Sidiropoulos, J. Cook, M. Gillespie, R. Haw, F. Loney, B. May, M. Milacic, K. Rothfels, C. Sevilla, V. Shamovsky, S. Shorser, T. Varusai, J. Weiser, and G. Wu. The reactome pathway knowledgebase. *Nucleic acids research*, 48(1):D498–D503, 2020.

[Mai] Hello world. `https://reactjs.org/docs/hello-world.html`. Note that the collection of pages under the "MAIN CONCEPTS" section in the website sidebar are being referenced, and not just this first one.

[Mata] Material components for the web. `https://github.com/material-components/material-components-web`.

[Matb] Foundation. `https://material.io/design/foundation-overview`.

[Par] Parcel documentation. `https://parceljs.org/getting_started.html`.

[Reaa] Content service. `https://reactome.org/ContentService`.

[Reab] Content service. `https://reactome.org/dev/content-service`.

[Reac] Data model. `https://reactome.org/documentation/data-model`.

[Read] Graph database :: Data schema. `https://reactome.org/content/schema`.

[Reae] The pathway browser. `https://reactome.org/dev/content-service`.

[Reaf] Reactome pathway diagrams specifications. https://reactome.org/dev/diagram/pathway-diagram-specs.

[VDS+07] Imre Vastrik, Peter D'Eustachio, Esther Schmidt, Geeta Joshi-Tope, Gopal Gopinath, David Croft, Bernard de Bono, Marc Gillespie, Bijay Jassal, Suzanna Lewis, Lisa Matthews, Guanming Wu, Ewan Birney, and Lincoln Stein. Reactome: a knowledge base of biologic pathways and processes. *Genome Biology*, 8(3):R39, 2007.

[You17] Alex Young. *Node.js in action.* Manning Publications, Shelter Island, NY, second edition. edition, 2017.

Appendices

Appendix A

App Source Code

A.1 Source Code Location

The Node project files for the web app constitute all code written in this project. They can be found at the Github Git repository:

`https://github.com/kicknickr/reactome-search-vis-prototype`

A.2 Instructions on Building the Node Project

It is recommended that one be familiar with Node project structures and package.json files before attempting to build the source code into a functioning a web app.

Building the project will require the Node and NPM applications to be installed on one's machine. Once installed, navigate to the top level directory of the project, and run the default package installation command:

```
npm install
```

Followed by the customized build script:

```
npm run parcel-build --scripts-prepend-node-path=auto
```

This will create a `./build/output` directory containing html, js, and css files that can be opened in a browser.

If alternatively one wishes to run a dev server on the localhost domain that they can point their browser towards, then run the following customized dev server script:

```
npm run parcel-dev --scripts-prepend-node-path=auto
```

and point their browser towards `http://localhost:1234/`.

Please see an important note in the next section about a potential issue with opening the web app in either case.

A.3 Instructions on Opening the Web App

Opening a local web app is usually as simple as opening an index.html or localhost port in a browser of choice. While this is the final step, the browser must first be configured to allow local web apps to make requests to remote servers, such as the Reactome API. In some browsers, a web page making a remote request from a local origin is considered a CORS security risk and is blocked by default. If the app were hosted on an actual web-server and not run locally, this would not be a problem.

A simple way to remove the restrictions imposed by this CORS security violation is to install a browser add-on that disables these restrictions. This technique was used during the development process. It should be noted that tampering with a browser's security settings should be done cautiously. To limit potential risks, one may wish to install/enable and uninstall/disable such an add-on before and after using the desired app.

In the Firefox browser, the "Cross Domain - CORS" add-on works once it is turned on, without the need for additional configuration. To specify a whitelist of request origins for further risk reduction, the "CORS Everywhere" add-on also works. This second add-on was used during the development process while a dev server was running on localhost. The entry placed in add-on's whitelist for the development server was as follows: `/^https?...localhost:1234.+/i`.

Once a browser is configured properly, `./build/output/index.html` may be opened in the browser to open the app developed in this project.