

12/11/2014

CIS 4911 Senior Project

Intelligence Inference Engine
Feasibility Study and Project Plan

Group Members

Jose Acosta
Lazaro Herrera

Mentor

Eric Kobrin

Instructor

Masoud Sadjadi

Intelligence Inference Engine : iie-dev.cs.fiu.edu

[Copyright](#) © [2014] Florida International University. All Rights Reserved. This work is distributed under the W3C[®] Software License [1] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

[1] <http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231>

Abstract

Most web searches currently depend on matching keyword or phrases in order to return results. Matching keyword and phrases works as long as you know what you are searching for and you only want information to that specific thing. In the modern world, when it comes to cyber security new exploits and attacks are constantly coming out. It is not feasible to keep track and know everything about these attacks because there are so many. Fortunately, most new attacks and exploits are usually related to old attacks and exploits. This is where the Semantic Web comes in. Data in Semantic Web is represented by relations, this makes it easy to search for things even without knowing exactly what you are looking for. The only problem with the Semantic Web is that there is a barrier of entry, knowing a specific querying language. This project aims to lower that barrier of entry.

Table of Contents

[Abstract](#)

[Table of Contents](#)

[Introduction](#)

[Problem definition](#)

[Scope of system](#)

[Overall development methodology](#)

[Definitions, acronyms, and abbreviations](#)

[Overview of document](#)

[Feasibility Study](#)

[Description of current system](#)

[Description of alternative solutions considered](#)

[Recommendation](#)

[Project Plan](#)

[Project Organization](#)

[Project Personnel](#)

[Hardware and Software Resources](#)

[Identification of Tasks, Milestones and Deliverables](#)

[Cost of the Project](#)

[System Requirements](#)

[Functional and Nonfunctional Requirements](#)

[Requirements Analysis](#)

[Scenarios](#)

[Use case model](#)

[System Design](#)

[Overview](#)

[Subsystem Decomposition](#)

[Hardware and Software Mapping](#)

[Persistent Data Management](#)

[Security/Privacy](#)

[Detailed Design](#)

[Overview](#)

[Static model](#)

[Dynamic model](#)

[Code Specification](#)

[System Validation](#)

[Subsystem Tests](#)

[System Tests](#)

[Evaluation of Tests](#)

[Glossary](#)

[Appendix](#)

[Appendix A - Project schedule](#)

[Appendix B – All use cases with nonfunctional requirements.](#)

[Appendix C – User Interface designs.](#)

[Appendix D – Analysis models \(static and dynamic\)](#)

[Appendix E – Design models \(static and dynamic\)](#)

[Appendix F – Documented Class interfaces \(code\) and constraints.](#)

[Appendix G – Documented code for test drivers and stubs.](#)

[Appendix H – Diary of meeting and tasks for the entire semester.](#)

[Date: 9/3/2014](#)

[Date: 9/6/2014](#)

[Date: 9/21/2014](#)

[Date: 10/4/2014](#)

[Date: 10/19/2014](#)

[Date: 10-28-2014](#)

[Date: 10-31-2014](#)

[Date:11/1/2014](#)

[Date: 11/17/2014](#)

[Date: 12/1/2014](#)

[References](#)

Introduction

This section of the paper will introduce the problem that this project is attempting to solve. This section will also begin to introduce the solution along with what methods were used to arrive to it. This section will define any necessary terms that are needed to properly explain the problem that is being tackled. Finally this section will introduce the purpose of the document and what to expect in the following sections of the document.

Problem definition

The Semantic Web is about linked data. Linked Data is resource-based linking of information. The Semantic web is built of by large of linked data which is defined by the Resource Description Framework(RDF). Each RDF data point consist of three parts: a subject, a predicate, and an object. In essence, RDF give you little building blocks of data that can be connected to other building blocks of data in both directions. When you build complicated webs of connected information, you end up with really specific detailed structures of conceptual knowledge over which you can answer complex question programmatically.

Right now, if you went to Google and put a search query like “Everything related to exploits on SSL that affects Linux and similar systems ”, the results would mostly be articles that cover SSL, some that may cover Linux, and maybe some that cover exploits on SSL. That search result would be useless because it does not give you what you asked for, instead it just matched the keywords you put in, Google does not know what systems are similar to Linux by keywords alone, it does not know what how to put all that together to find specifically what your are looking for . This means that you have to know what you are looking for before you look for it. With Semantic Web you can use relations between different data points and tries to answer the query using relations.

Scope of system

The deliverables for this project will be given in two week increments which is our current sprint cycle. This will line up with our in-class scheduled presentations. We believe that the features that should be developed should both be shown to the client and our fellow classmates for feedback. The technical limits will go as far as generating valid SPARQL from a web form and using that for both searches and submission of data to our triple-store.

Overall development methodology

Definitions, acronyms, and abbreviations

OSINT - Open-source intelligence

Cyber-attack - Any type of offensive maneuver employed by individuals or whole organizations that targets computer information systems, infrastructures, computer networks, and/or personal computer devices by various means of malicious acts usually originating from an anonymous source that either steals, alters, or destroys a specified target by hacking into a susceptible system [1]

Triple store - A triple-store is a purpose-built database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object. [2]

RDF - a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats. [3]

Semantic Web - collaborative movement led by international standards body the World Wide Web Consortium (W3C). The standard promotes common data formats on the World Wide Web. By encouraging the inclusion of semantic content in web pages, the Semantic Web aims at converting the current web, dominated by unstructured and semi-structured documents into a "web of data". The Semantic Web stack builds on the W3C's Resource Description Framework (RDF). [4]

SPARQL - an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework format. [5]

Overview of document

This document contains the feasibility study of the solution that was created for the problem. In the section for the feasibility study there is a description of the current system. Alternative solutions that were discussed as possible alternative routes to fix the problem are also mentioned in the feasibility study section. Finally, recommendations with an explanation to why the selected solution was chosen rather than alternatives is included.

After the feasibility study section comes the project plan. In the project plan the breakdown of the organization of the solution is described. It also shows what each person was responsible for and what the tasks, milestones, and deliverables were needed for the project. The hardware and software that were used for the solution are talked about and the costs associated with the project.

Next comes the overall system design. In this section the software solution will be broken down into the various subsystems that it is built of. The hardware and the software of the software solution are mapped and the persistent data management is addressed. Finally this section covers the privacy and security concerns of the software solution.

After the overall design comes the detailed design. In the detailed design the various subsystems that were shown in the previous section are dissected. This means that the various models that are used to show the subsystems are explained in detail. This section also shows the specific code that was used to implement the subsystems.

The final section of this document goes over the validation that was used in order to verify that the software solution was working as expected. This section goes over the various system tests and subsystem tests that were run. It shows test cases and explains them along with which use cases they covered. Finally the tests that were used are evaluated to make sure they were sufficient for testing the software solution.

Feasibility Study

Description of current system

Currently, there exist many websites that catalog cyber-attacks, data breaches and vulnerabilities. Although all of these websites are critical to today's security researchers, missing any of them can leave important data behind as well as not knowing the correct term to search for can cause some of the important information to be left behind. Consider the possibility that an attack may be filed as targeting "banks" yet the security researchers might be searching for an attack that is targeting "financial institutions". Similar terms being used but a standard search engine does not take these into account.

Description of alternative solutions considered

An alternative to the proposed solution would be to change the entry and query of data from a web form to an independent client that a user would install. The system could also use a relational database rather than a triple store to store all the data the system will receive.

Recommendation

For our project we will be using a bootstrap-based web form built on Jena. A web form is being used for easy user access from both desktops and mobile devices instead of providing a native client. Jena's triple-store along with its REST interface allows us to perform queries directly from the web form in multiple languages (Sparql/Tql).

Jena was chosen over sqlLite for the inference engine mainly because of its ease of development. Jena is significantly more efficient for what this system needs to do as opposed to using sqlLite. A bootstrap web form was chosen because it is much easier to develop a single web client that anyone with a web browser can access as opposed to native clients for each operating system. Also with a web client there is no installation so its a little easier to use than a web client.

Project Plan

Project Organization

Project Personnel

Lazaro Herrera - Developer / UI Design / Testing

Jose Acosta - Developer / Database / Query Library Development

Eric Kobrin - Project Manager / Client

Hardware and Software Resources

Hardware - Single core CPU / 2 GB Ram / 30 GB hard drive

Software - Apache Jena / Easy Dev PHP / Bootstrap / jQuery / MySQL / Selenium / D3JS

Identification of Tasks, Milestones and Deliverables

The tasks of this project were defined and tracked using Trello.

For our Trello board, we have decided to use color tagging to both identify the assignments of tasks and the completion status of tasks

Colors and Definitions for Priority

Blue - Card created by students

Purple - Card created by client

Colors and Definitions for Completions (only one active at a time)

Green - This task has a completion of 25% or more

Yellow - This task has a completion of 50% or more

Orange - This task has a completion of 75% or more

Red - This task has a completion of 100%

The following epics currently exist on our board.

Some of these are receiving checklists to be converted into stories.

- Setup Development Environment (Jena)
- Setup Development Environment (Web Server + Bootstrap)
- Develop Primary Feature (Data Entry Web Form)
- Test Primary Feature (Data Entry Web Form)

- Develop Primary Feature (Data Retrieval Web Form)
- Test Primary Feature (Data Retrieval Web Form)
- Develop Secondary Feature (Database Explorer)
- Test Secondary Feature (Database Explorer)
- Develop Secondary Feature (Custom Reusable Queries)
- Test Secondary Feature (Custom Reusable Queries)
- Develop Secondary Feature (Confidence Ranking)

There were four major milestones that were followed for this project. The first was getting a basic UI. This milestone included getting all the pages in a mock up stage that were actual html pages. This was important because future updates could be done directly on the mock up eventually turning it into the actual UI. The next milestone was getting basic functionality working. This milestone was important because it was necessary to know whether the basic idea of the solution was a possible course of action. The next milestone was tweaking and upgrading the previous two, the UI and functionality. The final milestone is testing the code. This is very important to make sure that the software solution is in working order and that everything was implemented correctly.

Cost of the Project

Cost Matrix	Weeks Required	Jose - \$30 hour / 12hr week	Lazaro - \$30 hour / 12hr week	Total Cost
Data Entry Form Implementation	2	\$720.00	\$720.00	\$12,672.00
Data Entry Form Testing	1	\$360.00	\$360.00	
Data Retrieval Form Implementation	2	\$720.00	\$720.00	
Data Retrieval Form Testing	1	\$360.00	\$360.00	
Database Explorer Implementation	3	\$1,080.00	\$1,080.00	
Database Explorer Testing	2	\$720.00	\$720.00	
Custom Queries Implementation	1	\$360.00	\$360.00	
Custom Queries Testing	0.6	\$216.00	\$216.00	
Confidence Ranking Implementation	3	\$1,080.00	\$1,080.00	

Confidence Ranking Testing	2	\$720.00	\$720.00	
-------------------------------	---	----------	----------	--

System Requirements

The proposed system is an easy to use forms with easy to utilize UI and a library that allows the generation of SPARQL from simple text inputs. The SPARQL queries are sent to a database where they are run and return a result. The result is displayed to the user in an easy to understand format.

Functional and Nonfunctional Requirements

Functional:

- A user must be able to submit data to be stored through some type of web form
- The data should be query-able directly using Sparql or Datalog
- A user must be able to set up predefined queries which are accessible by other users
- The system must be able to graphically display all the data that it currently stores.
- The system must be able to import data given a file with a specific format.
- The system must be able to export data to a file in a specific format.
- The system must be able to set a confidence interval for the data that is being collected.

NonFunctional:

- The data must be stored in one of the existing semantic web triple- or quad-stores such as Mulgara or Jena

Requirements Analysis

Scenarios

Scenario 1: Data Entry Success

Bob, a security analyst at company X, has gathered some data on attacks that have been happening to Company Y. Bob wants to share this data to see if further connections can be made. Bob goes to the web site that is being hosted at URL Z, and clicks on the button to contribute data. When Bob gets to the web form, he will fill out the various field that are required. Once Bob has filled out the form, he will click on the submit button. The data that Bob entered will then be sent to the Jena triple store and Bob will receive a confirmation telling him that the data has been entered.

Scenario 2: Data Entry Fail : Back End Down

Bob, a security analyst at company X, has gathered some data on attacks that have been happening to Company Y. Bob wants to share this data to see if further connections can be made. Bob goes to the web site that is being hosted at URL Z, and clicks on the button to contribute data. When Bob gets to the web form, he will fill out the various field that are required. Once Bob has filled out the form, he will click on the submit button. The web form will attempt to connect to the back end to enter the information Bob has provided, but it is down. Bob will remain on the current web form and receive an alert informing him that the back end service is currently down and to please try to enter the information again later.

Scenario 3: Data Entry Fail : Invalid Information

Bob, a security analyst at company X, has gathered some data on attacks that have been happening to Company Y. Bob wants to share this data to see if further connections can be made. Bob goes to the web site that is being hosted at URL Z, and clicks on the button to contribute data. When Bob gets to the web form, he will fill out the various field that are required. Once Bob has filled out the form, he will click on the submit button. Once the submit button has been pressed the information on the web form is checked to verify it is valid. It is found that one or more of the fields that Bob has filled out contain invalid information. Bob will remain at the current form. Bob can now fix the invalid fields and retry or he can exit the page and abandon entering data.

Scenario 4 : Data Entry : Import Data

Bob, a security analyst at company X, has gathered a lot of data on some recent attacks. Bob want to share his data but it is too much data to enter manually. Bob would go to the web site that is being hosted at URL Y, and clicks on the button to contribute data. On the page to contribute data, Bob would click on the button that says bulk import. Bob would then choose the file with the data he wants to contribute, and is in the correct format. The data will then be added to the server and the page will show how much data was inserted.

Scenario 5: Data Retrieval Success: Custom Query

Bob, a security analyst at company X, has noticed that recently there have been a lot of attacks in the field his client's company is in. Bob wants too see if his client's company is at risk of an attack. Bob goes to the web site that is being hosted at URL Z, and clicks on the button to query the data that has been collected. In the next form, Bob clicks on the button to enter his own custom sparql query. Bob will enter his query into the text box provided and then click on the submit button. The query that Bob has entered will then be sent to the Jena server and the results will sent back to be displayed on the web client Bob is on. Bob can now view the results of his query.

Scenario 6: Data Retrieval Success: Generated Query

Bob, a security analyst at company X, has noticed that recently there have been a lot of attacks in the field his client's company is in. Bob wants too see if his client's company is at risk of an attack. Bob goes to the web site that is being hosted at URL Z, and clicks on the button to query the data that has been collected. In the next form, Bob will be presented with several fields to fill out in order for the page to automatically generate a query. Once Bob has filled out the required fields, He will then click on the submit button. The web form will generate a query and send it to the back end Jena service. Once the results are retrieved, the page will show him the results of his query.

Scenario 7: Data Retrieval Failure : Back End Is Down

Bob, a security analyst at company X, has noticed that recently there have been a lot of attacks in the field his client's company is in. Bob wants too see if his client's company is at risk of an attack. Bob goes to the web site that is being hosted at URL Z, and clicks on the button to query the data that has been collected. Bob will fill out the necessary information for the type of query he wants to do. Once Bob has filled out the information, he will press submit. Since the back end is currently down, Bob will remain at the current page and he will receive a message

informing him that the back end is currently down and he should retry at a later time. Bob can now wait and retry with his query or abandon his query.

Scenario 8: Data Retrieval Failure : No Results For Desired Query

Bob, a security analyst at company X, has noticed that recently there have been a lot of attacks in the field his client's company is in. Bob wants to see if his client's company is at risk of an attack. Bob goes to the web site that is being hosted at URL Z, and clicks on the button to query the data that has been collected. Bob will fill out the necessary information for the type of query he wants to do. Once Bob has filled out the information, he will press submit. The query is sent to the Jena back end but no results are generated from the query. Bob will remain at the current page and receive a table showing him that his query generated no results. Bob can now change his query and retry or he can abandon his query.

Scenario 9: Custom Query Saving Success

Bob, a security analyst at company X, has created a query that he thinks will retrieve information other people will want. Bob wants to share this query. Bob goes to the website that is being hosted at URL Z, and click on the button to query the data that has been collected. In the next form, Bob will click in the button that says "Custom Query". Once Bob is at the custom query form he will fill out the text box provided. After Bob has filled out the text box provided with the query that he wants to save, he will click on the check box that says "Save Query". Enabling the "Save Query" button will add two additional text boxes to the current form. Bob will fill out the new text boxes with a title for the query and for tags (separated by commas) to describe the query. Once the two new text boxes are filled, Bob will press the submit button. If the query was successful, Bob will be taken to a new page containing the results of the query and a message saying the query was saved along with the id for said query for quick access.

Scenario 10: Custom Query Saving Failure : Back End Services Down

Bob, a security analyst at company X, has created a query that he thinks will retrieve information other people will want. Bob wants to share this query. Bob goes to the website that is being hosted at URL Z, and click on the button to query the data that has been collected. In the next form, Bob will click in the button that says "Custom Query". Once Bob is at the custom query form he will fill out the text box provided. After Bob has filled out the text box provided with the query that he wants to save, he will click on the check box that says "Save Query". Enabling the "Save Query" button will add two additional text boxes to the current form. Bob will fill out the new text boxes with a title for the query and for tags (separated by commas) to describe the query. Once the two new text boxes are filled, Bob will press the submit button. Because the back end services are down the query cannot complete or save. Bob will receive a message informing him that the back end is currently down and the query could not be saved and to wait and retry later. Bob can now wait and retry at a later time or he can abandon saving his query.

Scenario 11: Custom Query Saving Failure : Query already exists

Bob, a security analyst at company X, has created a query that he thinks will retrieve information other people will want. Bob wants to share this query. Bob goes to the website that is being hosted at URL Z, and click on the button to query the data that has been collected. In the next form, Bob will click in the button that says "Custom Query". Once Bob is at the custom

query form he will fill out the text box provided. After Bob has filled out the text box provided with the query that he wants to save, he will click on the check box that says “Save Query”. Enabling the “Save Query” button will add two additional text boxes to the current form. Bob will fill out the new text boxes with a title for the query and for tags (separated by commas) to describe the query. Once the two new text boxes are filled, Bob will press the submit button. The query is attempted to be saved but it is found the query already exists. Bob will be taken to a new page containing the results of his query but will also receive a message informing him that the query already exists and give him the query’s id.

Scenario 12: Looking Up Custom Queries Success

Bob, a security analyst at company X, wants to look up some data, but he is not entirely sure on what he specifically wants to look for. Bob would go to the website that is at URL Z. Once Bob is at the website, he will click on the button to query the data that has been collected. At the query form, Bob will now click on the “Search” button which will take him to a new form. At this new form, Bob can fill out a text box in which he add tags to define what he wants to search for. Bob will click on the submit button when he has put in all the tags he wants to search for. The current saved queries will be compared to the tags Bob has input and relevant queries will be sent back to Bob’s client and be displayed for Bob to look at.

Scenario 13: Looking Up Custom Queries Failure: Back End Services Down

Bob, a security analyst at company X, wants to look up some data, but he is not entirely sure on what he specifically wants to look for. Bob would go to the website that is at URL Z. Once Bob is at the website, he will click on the button to query the data that has been collected. At the query form, Bob will now click on the “Search” button which will take him to a new form. At this new form, Bob can fill out a text box in which he add tags to define what he wants to search for. Bob will click on the submit button when he has put in all the tags he wants to search for. Because the back end services are down the search cannot be completed. Bob will remain at his current page and receive a message informing him that the back end services are current down and to try at a later time. Bob can now wait and retry his search or he can abandon it.

Scenario 14: Looking Up Custom Queries Failure: No Results

Bob, a security analyst at company X, wants to look up some data, but he is not entirely sure on what he specifically wants to look for. Bob would go to the website that is at URL Z. Once Bob is at the website, he will click on the button to query the data that has been collected. At the query form, Bob will now click on the “Search” button which will take him to a new form. At this new form, Bob can fill out a text box in which he add tags to define what he wants to search for. Bob will click on the submit button when he has put in all the tags he wants to search for. The back end services will attempt to find queries that match the parameters given by Bob but none are found. Bob will remain at his current page and receive a message informing him that his search parameters yielded no results. Bob can now modify his search and retry or he can abandon his search.

Scenario 15: Confidence Interval

Bob, a security analyst at company X, wants to look up some data, Bob would go to the website that is at URL Z. Once Bob is at the website, he will click on the button to query the data that has been collected. At the query form, Bob will now click on the “Search” button which will take him to a new form. At this new form, Bob can fill out a text box in which he add tags to define what he wants to search for. Bob will click on the submit button when he has put in all the tags he wants to search for. The current saved queries will be compared to the tags Bob has input and relevant queries will be sent back to Bob’s client and be displayed for Bob to look at. Confidence Interval will give Bob the results sorted by the evidence that was collected as reference when the data was initially input.

Scenario 16: RDF Data Explorer

Bob, a security analyst at company X, is curious about all the data that has been entered into the system but does not want to know any specific thing about the data. Bob would navigate to the RDF Data Explorer page. Once Bob is at the Database explorer page, a graph will be created that shows all the data that is currently in the database. From here Bob can examine the graph.

Scenario 17: RDF Data Explorer Failure: No Data

Bob, a security analyst at company X, is curious about all the data that has been entered into the system but does not want to know any specific thing about the data. Bob would navigate to the RDF Data Explorer page. Once Bob is at the Database explorer page, a graph will be created that shows all the data that is currently in the database. If there is no data then the graph will display two null point and a line between them representing that there is currently no data in the database.

Scenario 18 : Export Data

Bob, a security analyst at company X, has been using the system and has found the data provided to be useful. Bob decides he wants all the data that the database has in order to look over it. Bob will head to the Search Now page and click on the “Bulk Data Export” button. Bob will download a file containing all the data that is currently in the database.

Use case model

There are 7 use cases in this system (refer to Appendix B). The first use case is the Store Data use case. This use case covers the requirement to allow users to be able to store data into the system. Scenarios 1 to 3 are represented in this use case. The next use case is the Import Data use case. This use case covers the requirement to allow users to bulk import data into the database. This use case is an extension of the Store Data use case because it stores data through an additional method. This use case covers scenario 4. The next use case is the Query Data use case. This use case covers the requirement to allow uses to query the database and it covers scenario 6 to 8. The next use case is the Export Data use case. This use case fulfills the requirement to allow users to export data from the database. It is an extension of the Query Data use case as it queries for all the data then exports it. Scenario 18 is covered by this use case. The next use case is Custom Query, this use case also extends the Query Data use case as it also queries the database. This use case satisfies the requirement to allow users to provide their own

queries to be executed and to search other submitted queries and covers scenarios 5,9,12,13, and 14. The following use case is the Saving Custom Query. This use case provides the functionality for the requirement to save custom queries. This use case covers scenarios 10 and 11. The final use case is the Data Visualizer. This meets the requirement of being able to see a visualization of all the data currently in the database. This use case covers scenarios 16 and 17.

System Design

In this section the software solution will be broken down into the various subsystems that it is built of. The hardware and the software of the software solution are mapped and the persistent data management is addressed. Finally this section covers the privacy and security concerns of the software solution.

Overview

The system has a client-server architecture and uses the repository design pattern. The client side consists of the web pages that include the forms and the JavaScript library which generate the queries. The server side consists of several PHP scripts which deal with some minor validation and connect to the two repositories in the system (the SQL database and the RDF triple store). The client-server architecture was used because it was the best way to have the system be scalable. The repository design pattern was crucial for this system. It was crucial because the system depended on being able to store data that could be queried at a later time. The core of this solution is to store and search both queries and data using queries and without the repository design pattern that would not be feasible.

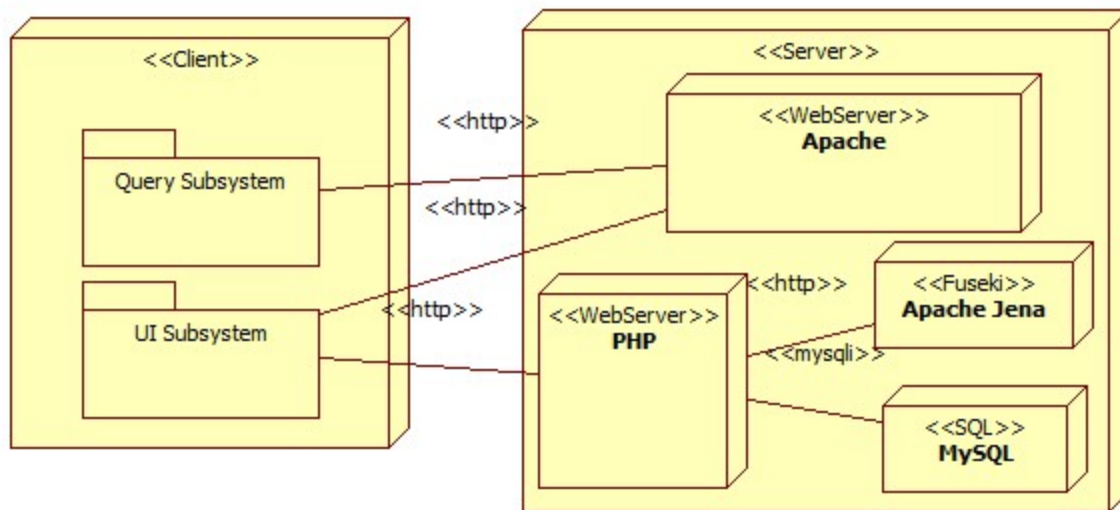
Subsystem Decomposition

The system is composed of two core subsystems that provide all the functionality. They are the user interface subsystem and the query generation subsystem. The user interface subsystem includes web forms for querying data from the database and submitting data from the database. This takes care of parts of the “Store Data” and “Query Data” use cases. The user interface system also involves evaluation of evidence. This takes care of the requirement specified by the user for the system to be able to evaluate the evidence given about submitted data from a user. The next component of the user interface subsystem is the storage and retrieval of custom queries. This component takes care of the “User Custom Query” and “Save Custom Query” use cases. The user interface also provides a visual graph of the data in the database which takes care of most of the “Visualize all data” use case. Finally the user interface implements buttons that allow the import and export of data. These buttons cover part of the “Import Data” and “Export Data” use cases.

The second core subsystem is the Query Generation subsystem. In this subsystem there is a library which creates query generators. These generators use information from the user interface subsystem to create valid SPARQL queries. One of the generators from the library creates select queries. This generator is part of the “Export Data”, “Query Data”, and “Visualize all data” test cases. The second generator from the library creates update queries. This generator provides the main functionality of the “Import Data” and “Store Data” use cases.

Hardware and Software Mapping

Below is a Hardware/Software map of how our software is executed. Apache serves up the QueryJS and UI Subsystems and they interact from the client machine to PHP which can then in turn interact with our database.



Persistent Data Management

For our Jena/Fuseki database, the data is always in the form of Subject - Predicate - Object pairs. Due to the nature of triple-stores this is a given and does not require a Persistent Data Management strategy

For our standard MySQL database, we will be using this storage for custom query storage and retrieval. As such, the following data dictionary is in effect.

Custom Query Data Dictionary			
Name	Length	Type	Rules
query	1024	TEXT	SPARQL code
tags	255	TEXT	CSV data
submission_time	1	TIMESTAMP	default: now()

Security/Privacy

Our main security concern for this system was blocking port 3030. In the end, we could not due to this requiring Windows Firewall to be enabled in order for a firewall rule to be set up. If we enable the Windows Firewall, Windows Desktop Connection would be blocked. This issue would not exist if we were in front of the hypervisor or had VNC capabilities.

Detailed Design

In this section of the document the subsystem that were introduced in the previous section will be further examined. There will be an overview that broadly covers what both of the subsystems do and how they are structured. Next, a detailed description of both subsystems will be given in terms of their static models and dynamic models. Finally, the core pieces of code that make up the two subsystems will explained in terms of that they do, and what conditions allows them to do function properly.

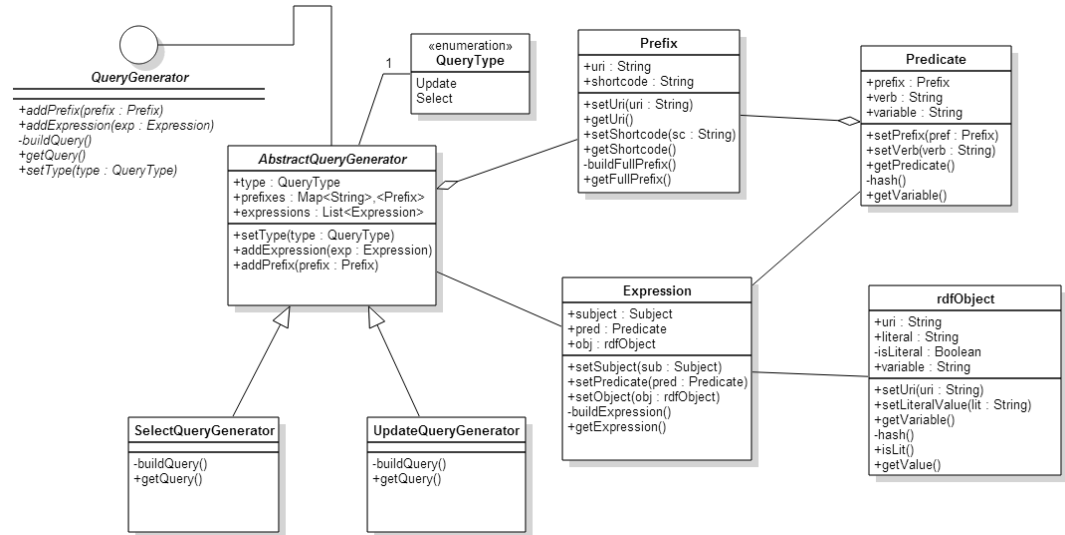
Overview

The QueryJS subsystem behaves as a lazy initialized factory class. Storage of the prefix and expressions occurs ahead of time but the query is only constructed on demand. This both saves resources from recalculating the query each update and allows for no “stale” references to older queries.

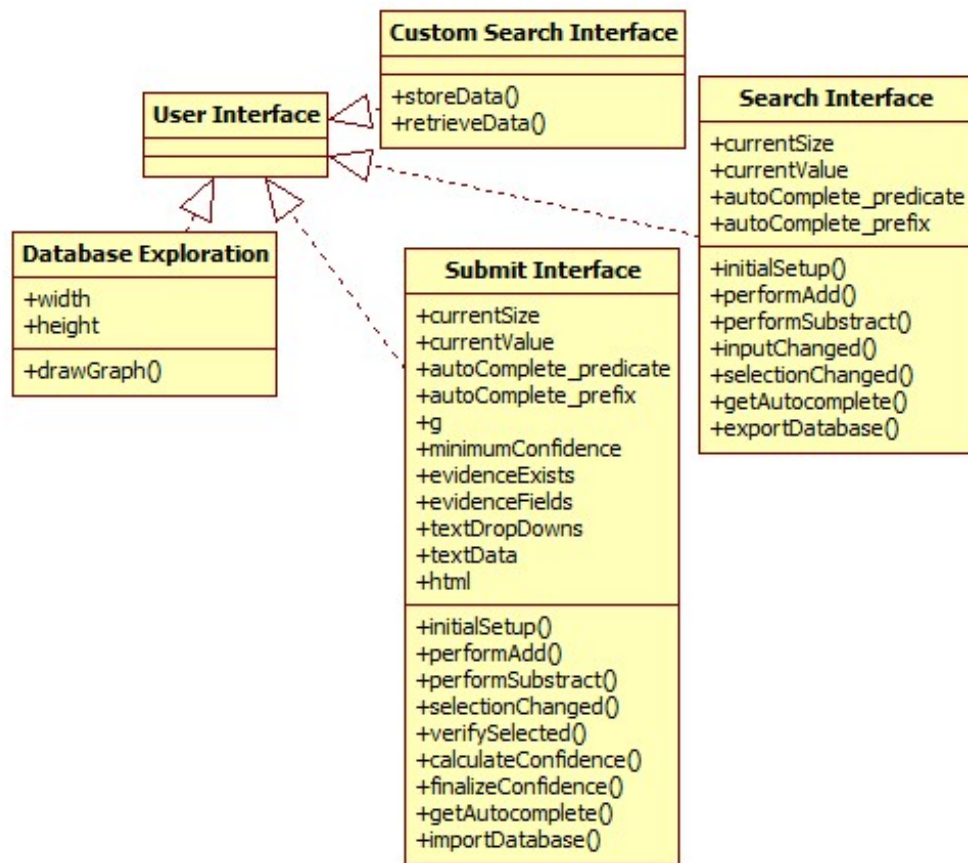
The UI subsystem is designed to create lightweight but powerful UIs on jQuery. Most of the code is shared amongst classes with only a few lines being devoted to the unique features of each UI.

Static model

Static model – detailed description of the structure for each subsystem. May include detailed class diagrams. Place diagrams (e.g., minimal class diagram, detailed class diagram per subsystem) inline. Use at least four (4) design patterns.



For the Query Generation Subsystem, the following design patterns apply (Abstract Factory, Builder and Lazy Initialization) as we delay the construction of the final query until the last possible moment while the factory and builder can be abstracted and controlled by a parameter sent to a method.



For the UI Subsystem, the following design patterns apply (Facade) as we wrap the forward facing capabilities of both QueryJS and Fuseki in a easier to use context. On top of this, we provide full facade capabilities to the custom query storage and retrieval through MySQL.

Dynamic model

Code Specification

In the Query Generation subsystem there are three main classes that control how the library works. The first two are both derieved from the same abstract class so I will describe them first, as they share the same interface.

The classes SelectQueryGenerator and UpdateQueryGenerator both have methods that are vital parts of the library. The first of those methods is the AddPrefix function. This function takes in a valid Prefix object and adds it to the specific query generator. In order for this function to work properly the QueryJS library must be loaded, or else it will not know what a Prefix is. This method adds the prefix given to it to an internal list of prefixes that will later be used for query generation. Another method that is core to the generator objects is the AddExpression method. This method takes in an Expression Object. As with the prefix method QueryJS must be loaded so that it understands what an Expression object is. This method adds the expression given to it to an internal list of expressions that will later be used to generate a query. Another method that is in both the generator classes is the setType function. This function takes in a

QueryType value and will assign it to the generator as its type if it is valid. The most important method of both generator is the BuildQuery function. This function will generate a valid SPARQL query if the prefixes is not null and the expressions is not null or empty. This method is the core of class because it combines the functionality of all the other classes in the library.

Another method that is crucial to the Query Generation subsystem is the BuildGenerator function. While this class is not inside the QueryJS library itself, it is the method which converts what the user interface subsystem gives and converts it into parts that are useable by the QueryJS library. This method has very strict guidelines on what the proper input is, but if the proper input is given and the QueryJS library has been loaded then it functions as a very important method that interfaces the two subsystems.

The UI Subsystem has it's main control interface under the submit and search web pages. This method is the core as it packages all of the UI material and passes it to the QueryJS Library, issues the query that is returned from QueryJS and displays either results or success/failure parameters.

System Validation

For this extremely heavy UI work with javascript, we utilized Selenium testing for regression along with Javascript driver/stubs for verification of regression as the months went on. On the system testing side, every feature was tested with an array of different scenarios as we deployed the new features.

Subsystem Tests

The Selenium test cases and Javascript driver/stubs are attached in Appendix G of this document.

System Tests

Test Case No:	T-001	Purpose:	Store Data
Test Setup:	Database online. UI Functional.	Inputs:	One subject with a predicate connected to another object
Expected Outputs	A subject related to another subject should be a valid test case.		

Test Case No:	T-002	Purpose:	Store Data
Test Setup:	Database online. UI Functional.	Inputs:	One subject with a predicate connected two objects
Expected Outputs	A subject related to multiple objects should be a valid test case.		

Test Case No:	T-003	Purpose:	Store Data
Test Setup:	Database online. UI Functional.	Inputs:	One subject without a predicate connected to an object
Expected Outputs	Subject object relations must have predicates.		

Test Case No:	T-004	Purpose:	Read Data
---------------	-------	----------	-----------

Test Setup:	Database online. UI Functional.	Inputs:	One subject without a predicate
Expected Outputs	All relation-object pairs related to this subject should be returned.		

Test Case No:	T-005	Purpose:	Read Data
Test Setup:	Database online. UI Functional.	Inputs:	One object without a predicate
Expected Outputs	All subject-relation pairs related to this subject should be returned.		

Test Case No:	T-006	Purpose:	Read Data
Test Setup:	Database online. UI Functional.	Inputs:	Subject - Predicate - Object triple that does not exist in the database.
Expected Outputs	An empty table should be returned as there is no relation.		

Test Case No:	T-007	Purpose:	Store Custom Query
Test Setup:	Database online. UI Functional.	Inputs:	Custom Query and a tag for storage
Expected Outputs	The query should be successfully stored		

Test Case No:	T-008	Purpose:	Retrieve Custom Query
Test Setup:	Database online. UI Functional.	Inputs:	Tag for retrieval
Expected Outputs	The query should be successfully recalled		

Test Case No:	T-009	Purpose:	Execute Custom Query
---------------	-------	----------	----------------------

Test Setup:	Database online. UI Functional.	Inputs:	Custom Query
Expected Outputs	Results should be returned to the Custom Query page and displayed.		

Test Case No:	T-010	Purpose:	Database Explorer
Test Setup:	Database online (multiple non-connected records) UI Functional.	Inputs:	None
Expected Outputs	Graph should be displayed with disconnected points.		

Test Case No:	T-011	Purpose:	Database Explorer
Test Setup:	Database online (multiple connected records) UI Functional.	Inputs:	None
Expected Outputs	Graph should be displayed with connected edges		

Test Case No:	T-012	Purpose:	Database Explorer
Test Setup:	Database online (no records) UI Functional.	Inputs:	None
Expected Outputs	Graph should be displayed empty.		

Test Case No:	T-013	Purpose:	Bulk Exporter
Test Setup:	Database online (with records) UI Functional.	Inputs:	None
Expected Outputs	File should contain the record information		

Test Case No:	T-014	Purpose:	Bulk Exporter
Test Setup:	Database offline	Inputs:	None

	UI Functional.		
Expected Outputs	File should contain error information		

Test Case No:	T-015	Purpose:	Bulk Exporter
Test Setup:	Database online (no records) UI Functional.	Inputs:	None
Expected Outputs	File should contain XML but no actual records.		

Test Case No:	T-016	Purpose:	Bulk Importer
Test Setup:	Database online UI Functional.	Inputs:	Import File (no duplicates)
Expected Outputs	Records should import correctly.		

Test Case No:	T-017	Purpose:	Bulk Importer
Test Setup:	Database online UI Functional.	Inputs:	Import File (multiple duplicates)
Expected Outputs	Records should import correctly but duplicates should be removed.		

Test Case No:	T-018	Purpose:	Bulk Importer
Test Setup:	Database online UI Functional.	Inputs:	Non-XML File
Expected Outputs	Nothing should be imported. UI should maintain stability.		

Evaluation of Tests

Test Case Number	Executed	Passed?	Remarks	Solution?
------------------	----------	---------	---------	-----------

T-001	Yes	Passed.	N/A	N/A
T-002	Yes	Passed.	N/A	N/A
T-003	Yes	Passed.	N/A	N/A

Test Case Number	Executed	Passed?	Remarks	Solution?
T-004	Yes	Passed.	N/A	N/A
T-005	Yes	Passed.	N/A	N/A
T-006	Yes	Passed.	N/A	N/A

Test Case Number	Executed	Passed?	Remarks	Solution?
T-007	Yes	Passed.	N/A	N/A
T-008	Yes	Passed.	N/A	N/A
T-009	Yes	Passed.	N/A	N/A

Test Case Number	Executed	Passed?	Remarks	Solution?
T-010	Yes	Passed.	N/A	N/A
T-011	Yes	Passed.	N/A	N/A
T-012	Yes	Passed.	N/A	N/A

Test Case Number	Executed	Passed?	Remarks	Solution?
T-013	Yes	Passed.	N/A	N/A
T-014	Yes	Failed.	Fuseki Error	None.
T-015	Yes	Passed.	N/A	N/A

Test Case Number	Executed	Passed?	Remarks	Solution?
T-016	Yes	Passed.	N/A	N/A
T-017	Yes	Passed.	N/A	N/A
T-018	Yes	Failed.	Chrome Crash.	None.

Glossary

OSINT - Open-source intelligence

Cyber-attack - Any type of offensive maneuver employed by individuals or whole organizations that targets computer information systems, infrastructures, computer networks, and/or personal computer devices by various means of malicious acts usually originating from an anonymous source that either steals, alters, or destroys a specified target by hacking into a susceptible system [1]

Triple store - A triple-store is a purpose-built database for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object. [2]

RDF - a family of World Wide Web Consortium (W3C) specifications originally designed as a metadata data model. It has come to be used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats. [3]

Semantic Web - collaborative movement led by international standards body the World Wide Web Consortium (W3C). The standard promotes common data formats on the World Wide Web. By encouraging the inclusion of semantic content in web pages, the Semantic Web aims at converting the current web, dominated by unstructured and semi-structured documents into a "web of data". The Semantic Web stack builds on the W3C's Resource Description Framework (RDF). [4]

SPARQL - an RDF query language, that is, a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework format. [5]

Appendix

Appendix A - Project schedule

						2014		
Card	Label(s)	Member(s)	Description	Start	End	September	October	November
Backlog								
Develop Primary Feature...	Card created b...	Jose Acosta	duration:14d	09-12-14	11-28-14			
Test Primary Feature (D...	Card created b...	Lazaro Herr...	duration:7d	09-23-14	09-30-14			
Develop Primary Feature...	Card created b...	Jose Acosta	duration:14d	10-07-14	10-21-14			
Test Primary Feature (D...	Card created b...	Lazaro Herr...	duration:7d	10-16-14	10-23-14			
Develop Secondary Feat...	Card created b...	Jose Acosta	duration:21d	09-19-14	10-10-14			
Test Secondary Feature ...	Card created b...	Lazaro Herr...	duration:14d	10-03-14	10-17-14			
Develop Secondary Feat...	Card created b...	Jose Acosta	duration:7d	10-20-14	10-27-14			
Test Secondary Feature ...	Card created b...	Lazaro Herr...	duration:4d	10-25-14	10-29-14			
Develop Secondary Feat...	Card created b...	Jose Acosta	duration: 21d	10-28-14	11-18-14			
Test Secondary Feature ...	Card created b...	Lazaro Herr...	duration:14d	11-14-14	11-28-14			
Ready/Planning								
▸ Setup Development En...	Card created b...	Lazaro Herr...	duration:8d	09-04-14	09-12-14			
▸ Setup Development En...	Card created b...	Jose Acosta	duration:8d	09-04-14	09-12-14			
Development								
Testing/Review								
Acceptance								
Done								
project: Intelligence Inference Engine						✓ Due		
last update:								

Appendix B – All use cases with nonfunctional requirements.

Name:

Store Data

Participants:

User

Events:

1. From the homepage of the website, the user must click on the “Contribute Now” button.
2. The user will be taken to a form which must be filled out.
3. On the form the user may enter his/her name or leave it blank.
4. The user must fill out the prefixes textbox with the prefixes they will be using for their properties.
5. The user must enter the subject of the data he is about to enter in the “Subject of Lead” textbox
6. The user must been enter at least one property of the subject by selecting a property from the properties drop down selecting the appropriate prefix in the prefix drop down and entering the value of the property in the textbox next to it.
7. If the user wants to submit more than one property they may click on the “+” button to add more property segments . If they want to remove any extra property input segment they can click on the “-” button.
8. The user clicks on the “Submit Lead” button.
9. The user will be taken to the Data Entry form and a message will appear on the top of the page notifying the user if their data was accepted or not.

Alternative Events:

- At step 8, if the user clicks the button and any of the data is invalid then an exception will occur that notifies the user there is invalid data.

- At step 1, or step 8, if the user clicks on the button and the system is down an exception will occur which will tell the user the system is currently down.

Entry Conditions:

User has data that they want to submit

User must be at homepage of the system

Exit Condition:

Data has been stored in the triple store and the user receives acknowledgment of it.

Exceptions:

Data that was input was invalid

The system is down.

Name:

Query Data

Participants:

User

Events:

1. From the home page, the user must click on the “Search Now!” button which will take them to the submit query form.
2. In the submit query form the user must enter the prefixes they will be using in the prefix textbox.
3. The user must then fill out one input segment by selecting at least one property from the drop down box along with an appropriate prefix from the prefix drop down and must enter a value in the text box.
4. If the user wants to submit more than one property they may click on the “+” button to add more property segments . If they want to remove any extra property input segment they can click on the “-” button.
5. The user must then click on the “Search Database” button.
6. The user will be taken to a page that has results of the query they submitted, if any exist.

Alternative Events:

- Instead of having the system generate a query for them, a user may submit their own query by selecting the “Custom Search” button, which will take them to the custom search form.
- At step 5, if the user clicks the button and any of the data is invalid then an exception will occur that notifies the user there is invalid data.
- At step 6, if no results are generated from the query that the user has used then an exception will occur which tells the user there are no results.
- At step 1, or step 5, if the system is down and the user goes through one of these steps then an exception will occur which will notify the user that the system is currently down

Entry Conditions:

User must be at homepage of the system

Exit Condition:

User has successfully entered a query and has gotten a list of results.

Exceptions:

User enters invalid data.

There are no results for the query specified by the user.

The system is down.

Name:

Use a custom query

Participants:

User

Events:

1. From the data retrieval page, the user clicks on the “Custom Search” button. This will take the user to the custom search page.
2. At the custom query page, the user will enter a query that they have generated themselves into the textbox provided.
3. The user will click on the “Execute Query” button
4. If the query is valid and produces results then the user will be taken to a results page.

Alternative Events:

- At step 2, If the user wants to use a query that another user has saved instead of one they generated themselves, then the user will enter the appropriate tags for that query in the tags textbox then click on “Find Queries”.
- At step 3, if the user has entered an invalid query then an exception will occur notifying the user that their query is malformed.
- At step 1, or 3, if the user follows these steps and the system is down an exception will occur notifying the user that the system is currently down.
- At step 4, if the user’s query provides no results then an exception will occur and the user will be notified that their query produced no results.

Entry Conditions:

The user must be at the data retrieval page

The user must have a query or the user must know the tags of a query that has been saved

Exit Condition:

User has entered a valid query and has gotten a list of results the query generated

Exceptions:

User entered an invalid query.

The query has returned no results.

The system is down.

Name:

Saving a custom query

Participants:

User

Events:

1. From the data retrieval page, the user clicks on the “Custom Search” button. This will take the user to the custom search page.
2. At the custom search page the user will enter the query that they want to store and enter the tags that will be used to identify that query in the tags textbox.
3. The user will click on the “Add Query” button.
4. The user will receive a message telling them whether the query they submitted was saved.

Alternative Events:

- At step 1, or step 3, if the system is down then an exception will occur and the user will be notified that the system is currently down.

Entry Conditions:

The user must be at the data retrieval page

The user must have a query and the tags he wants to save that query under

Exit Condition:

The query has been saved into the system.

Exceptions:

The system is down.

Name:

Import Data

Participants:

User

Events:

1. From the data entry page, the user will click on the “Bulk Import Data” button.
2. When the button is clicked, a new window will open asking the user to select the file they want to import.
3. The user selects the file that they want to import.
4. The file will be processed entry by entry and valid entries will be added to the database.

Alternative Events:

- At step 3, if the user selects a file that is invalid the web page will refuse the file and do nothing.
- At step 4, if there is an invalid entry then the page will tell the user that an entry has failed to be inserted.

Entry Conditions:

The user must be at the data retrieval page.

The user must have a file that they want to import in the correct xml format.

Exit Condition:

The data has been imported into the system.

Exceptions:

The system is down.

File in unsupported format.

Name:

Bulk Data Export

Participants:

User

Events:

1. At the data retrieval page, the user must click on the “Bulk Export Database” button.
2. Once the button has been pressed the user will download a file containing all the data that is currently in the database.

Alternative Events:

- N/A

Entry Conditions:

The user must be at the data retrieval page.

The user must be able to save a file to their local machine.

Exit Condition:

The user has downloaded a file with all the data in the database.

Exceptions:

The system is down.

Name:

Database Visualizer

Participants:

User

Events:

1. From the homepage, the user will click on the “Explore our database” tab at the top of the page.
2. Once the user has pressed the tab, then the user will be moved to the visualizer page.
3. At the visualizer page, a graph will be constructed using all the data that is currently in the database and the page will display the graph.

Alternative Events:

- At step 3, if there is no data in the database then a graph showing two null points and a line connecting them will be displayed.

Entry Conditions:

The user must be on the homepage of the system.

Exit Condition:

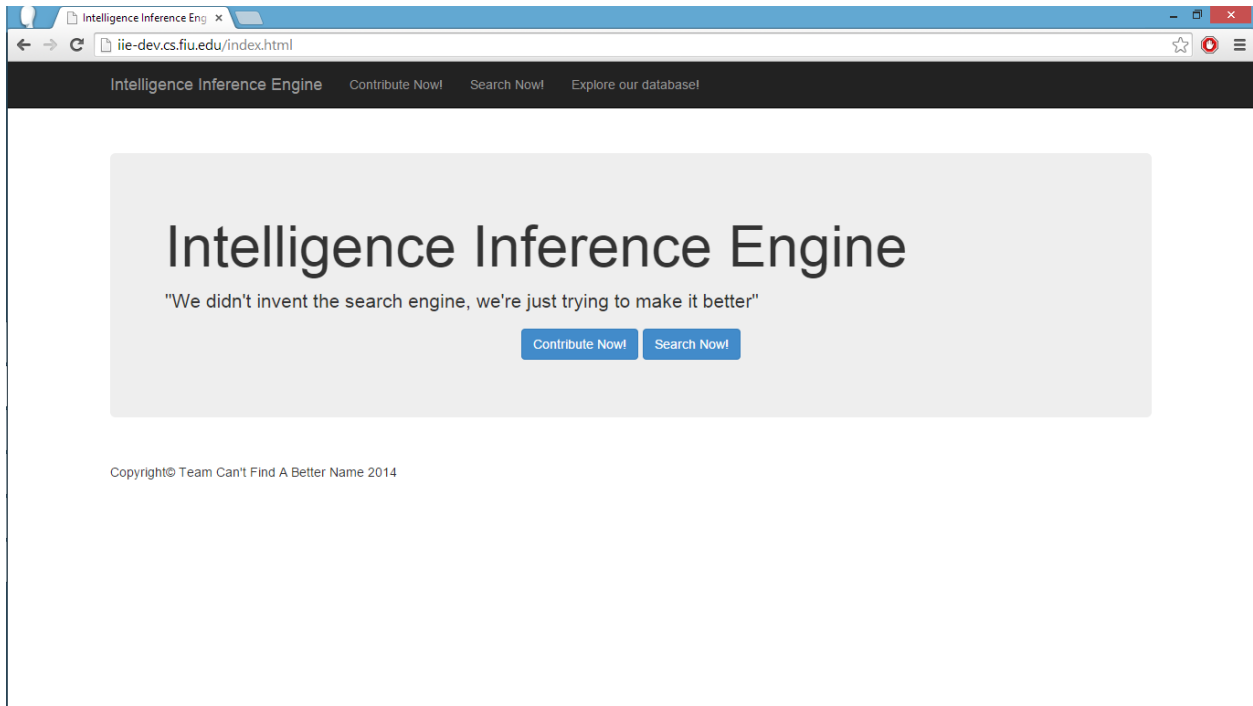
The user will be able to see a graph with all the data in the database.

Exceptions:

The system is down.

Appendix C – User Interface designs.

Home Page

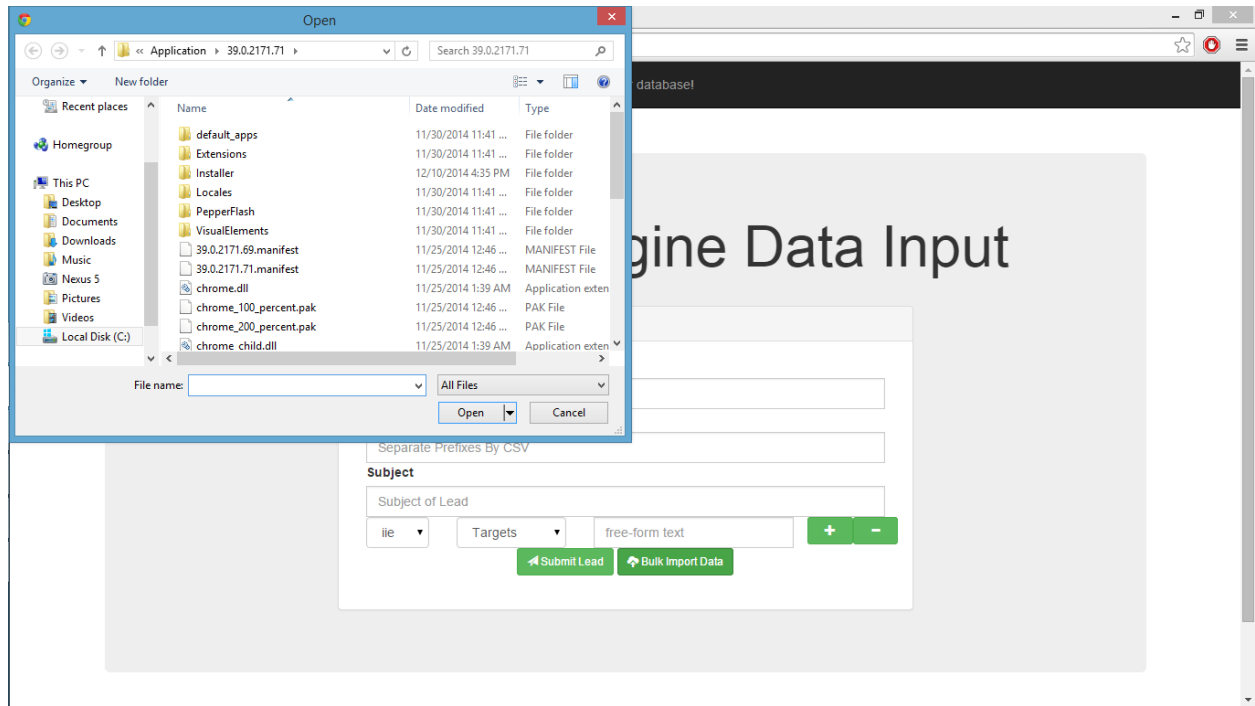


Data Input Page

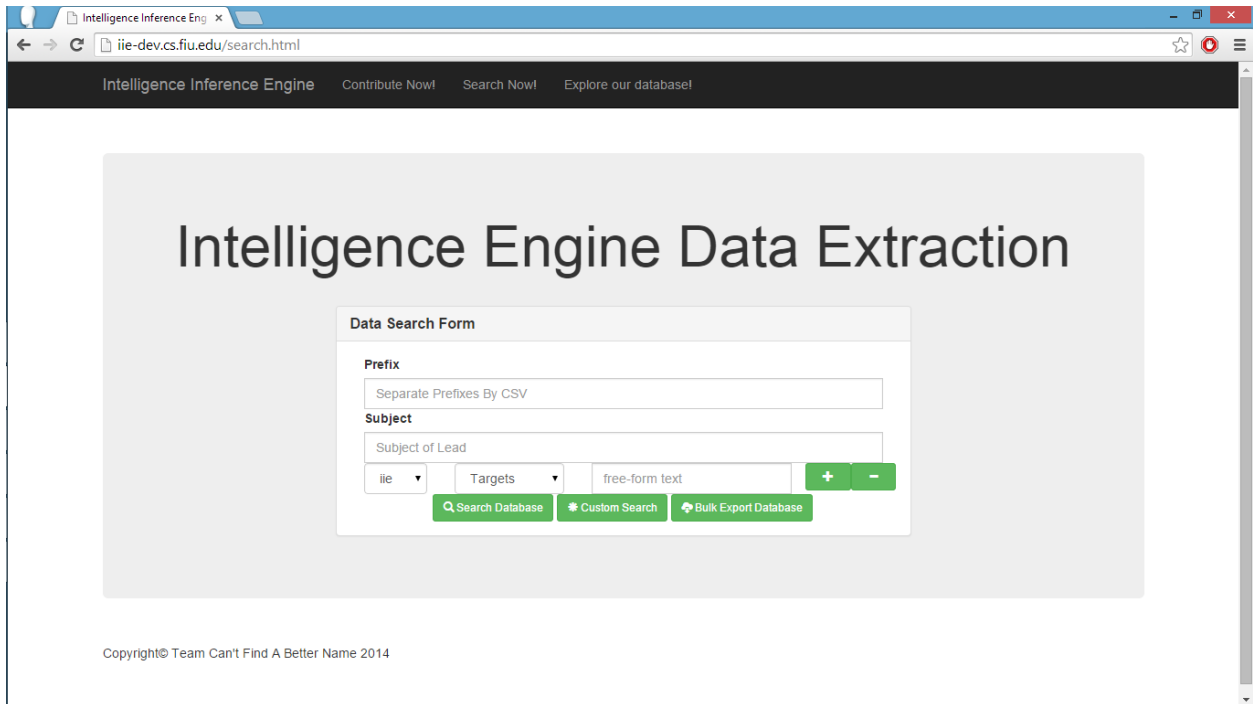
A screenshot of a web browser displaying the "Intelligence Engine Data Input" page. The browser's address bar shows the URL "iie-dev.cs.fiu.edu/submit.html". The page features a dark navigation bar with the text "Intelligence Inference Engine" and links for "Contribute Now!", "Search Now!", and "Explore our database!". The main content area has a light gray background with the title "Intelligence Engine Data Input" in a large, bold font. Below the title is a "Data Entry Form" with the following fields and controls:

- Submitter**: A text input field with the placeholder "Your Name/Institution Name".
- Prefix**: A text input field with the placeholder "Separate Prefixes By CSV".
- Subject**: A text input field with the placeholder "Subject of Lead".
- Below the Subject field, there are three dropdown menus: "iie", "Targets", and "free-form text".
- To the right of the "free-form text" dropdown are two green buttons: "+" and "-".
- At the bottom of the form are two green buttons: "Submit Lead" and "Bulk Import Data".

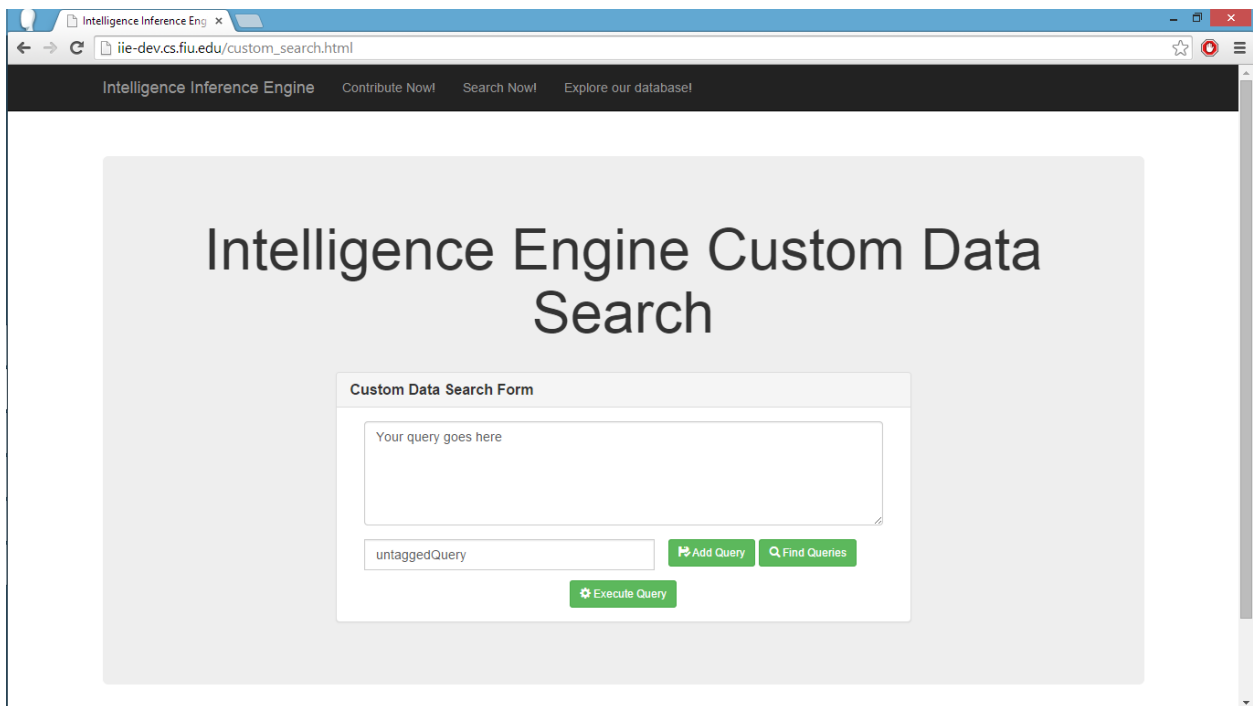
Data Import Dialog



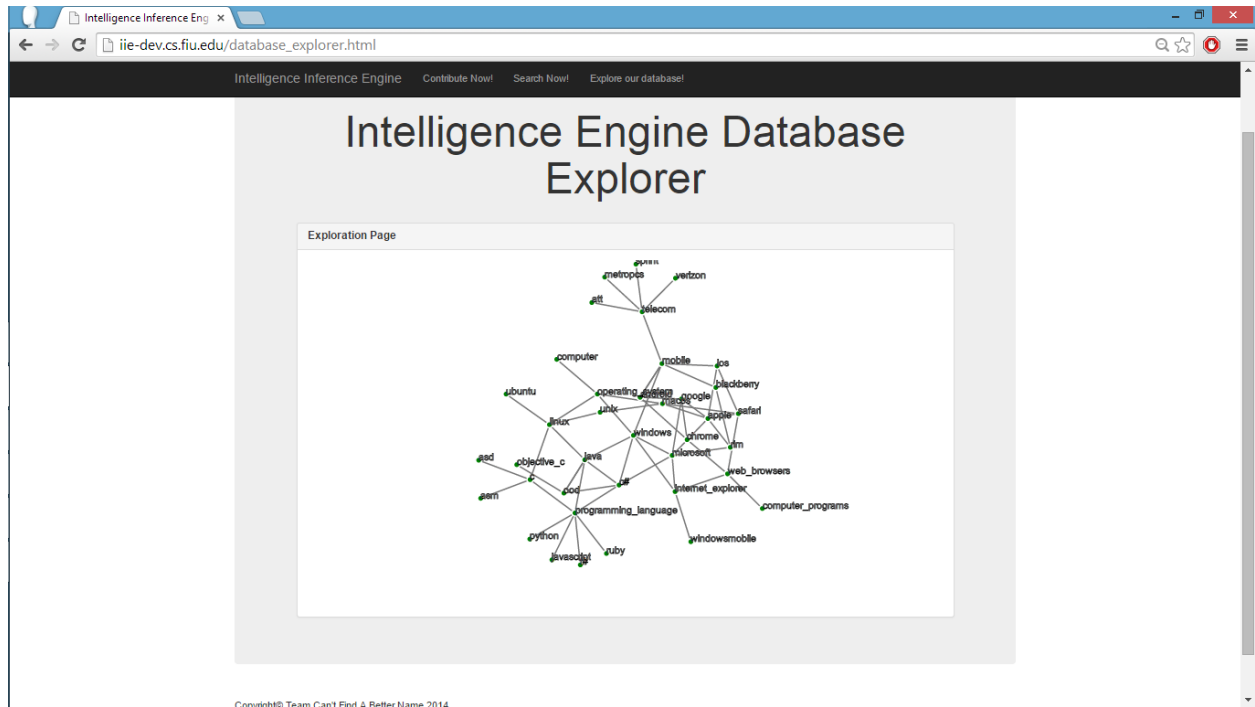
Data Search Page



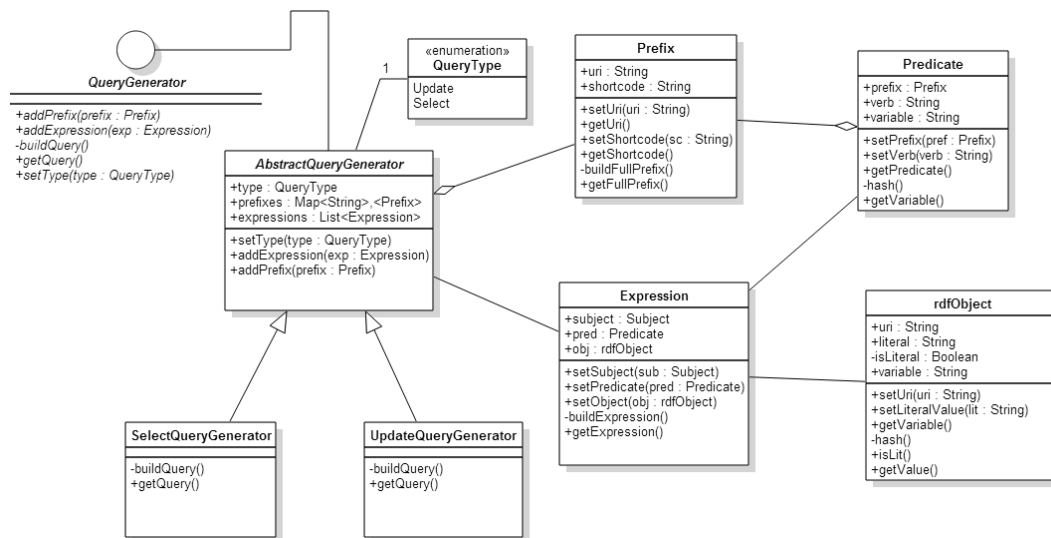
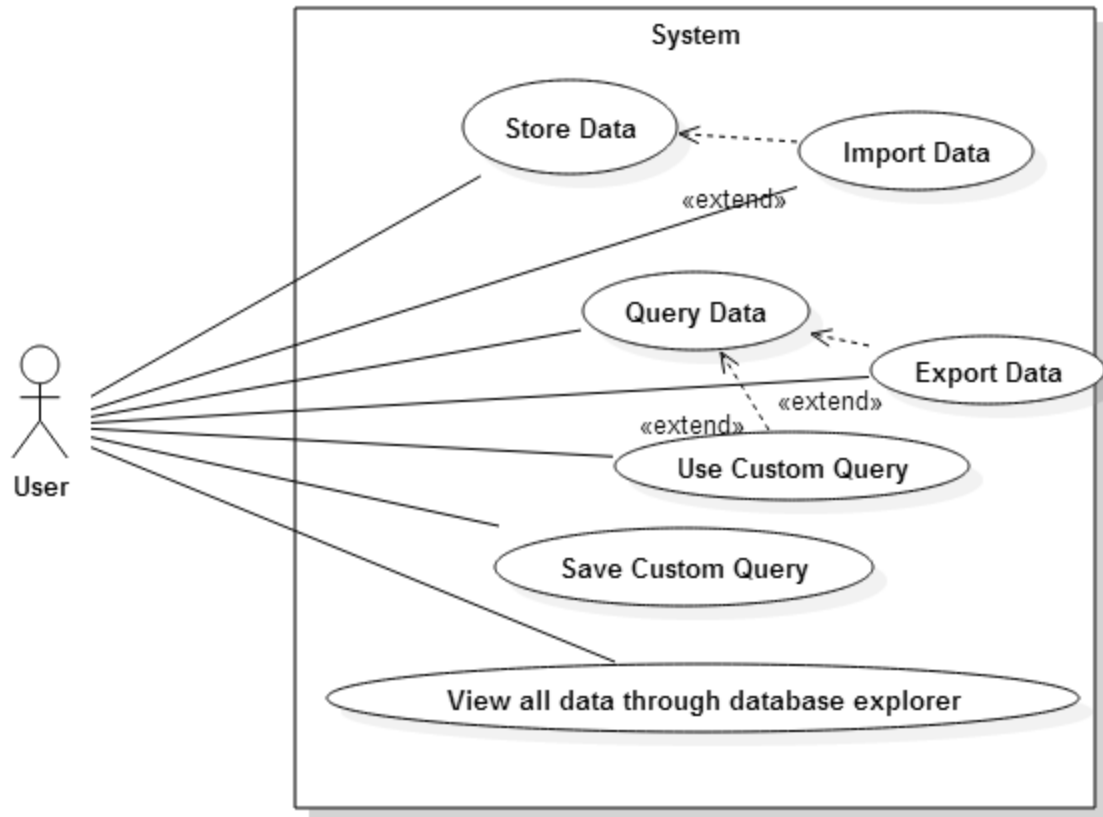
Custom Search Page

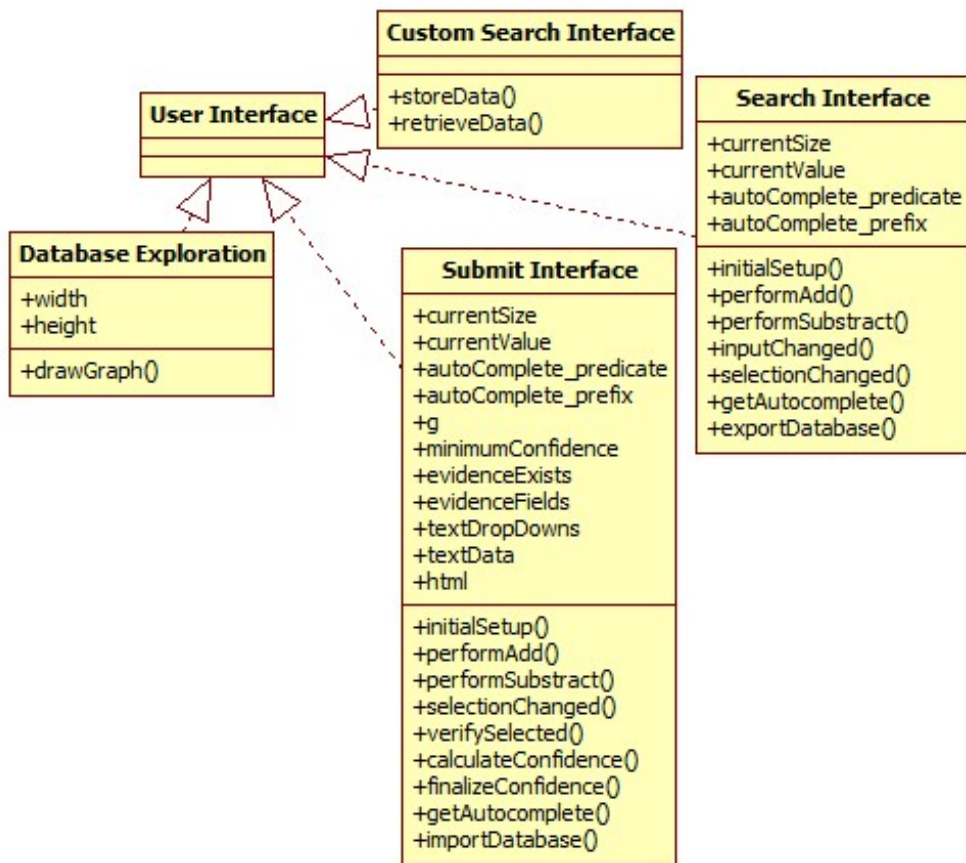


Data Exploration Page

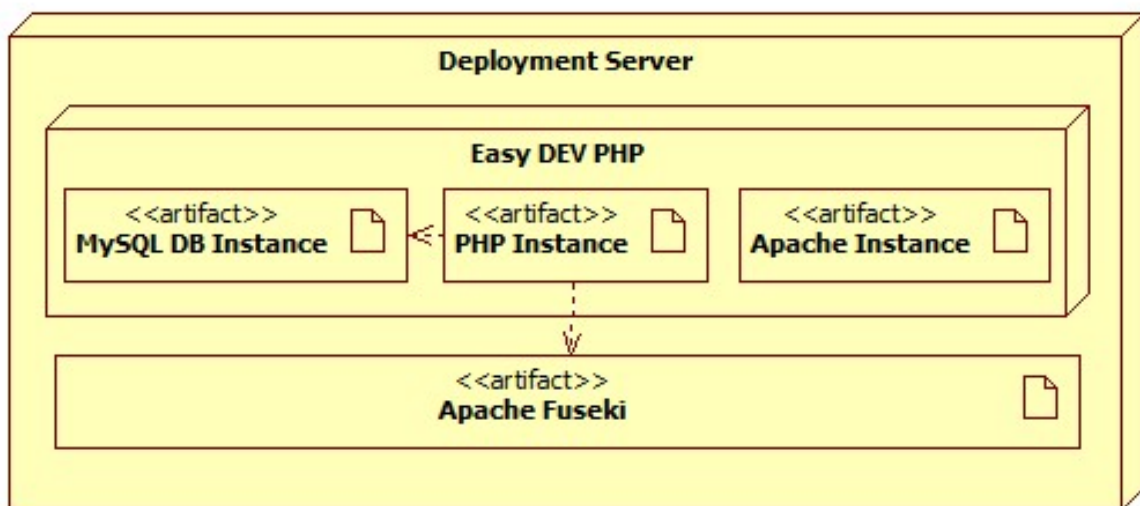


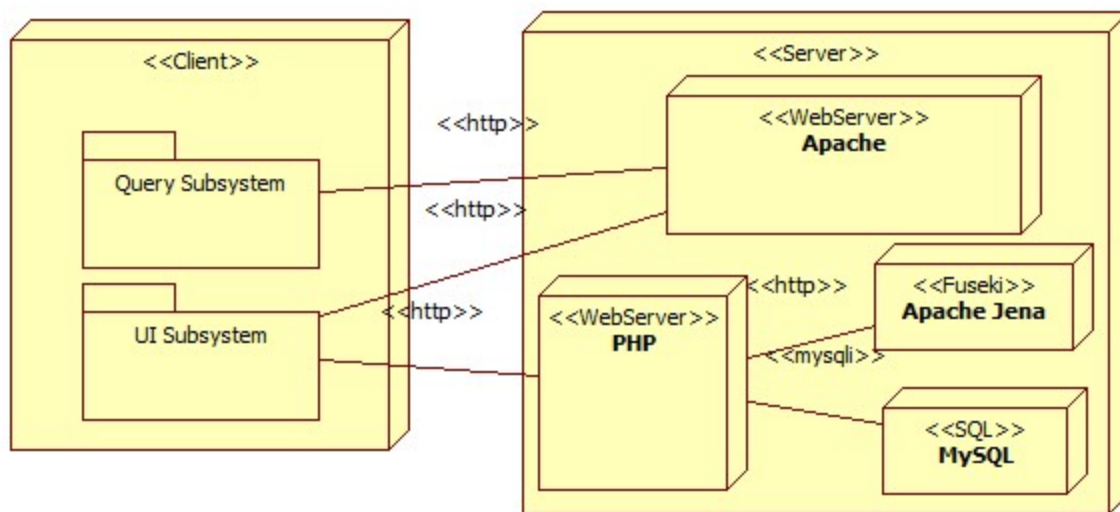
Appendix D – Analysis models (static and dynamic)



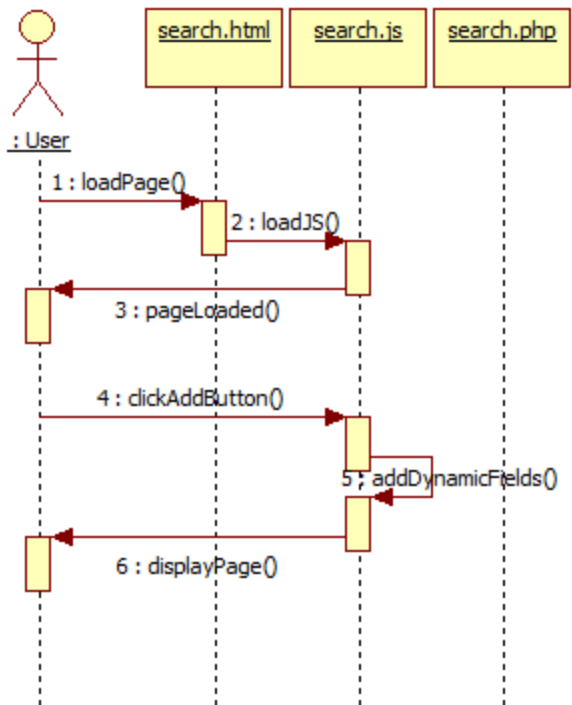


Appendix E – Design models (static and dynamic)

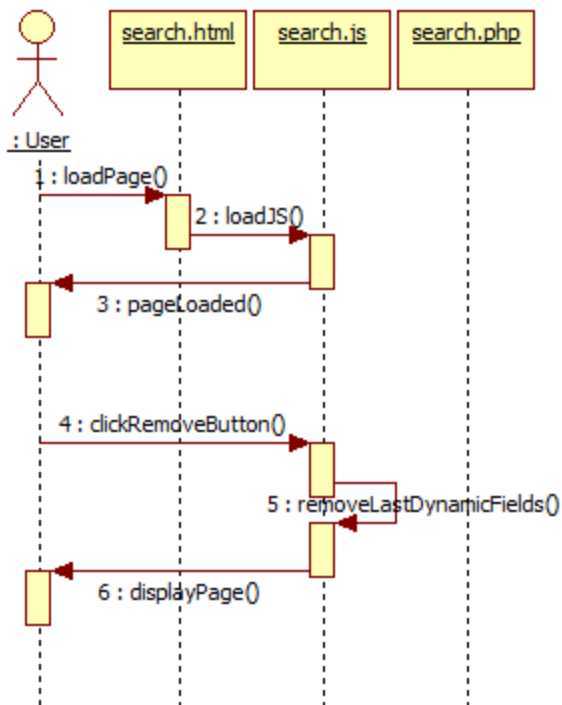


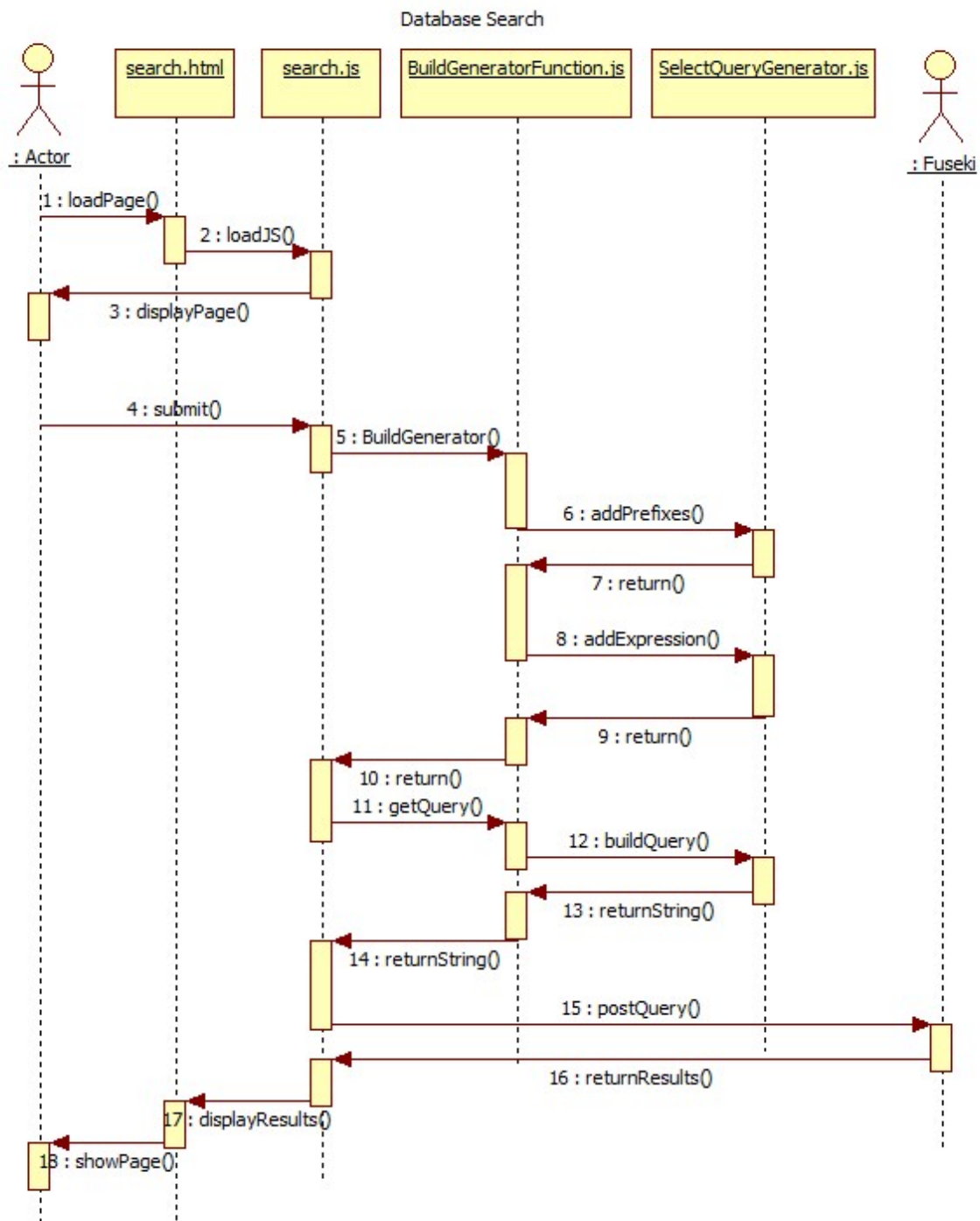


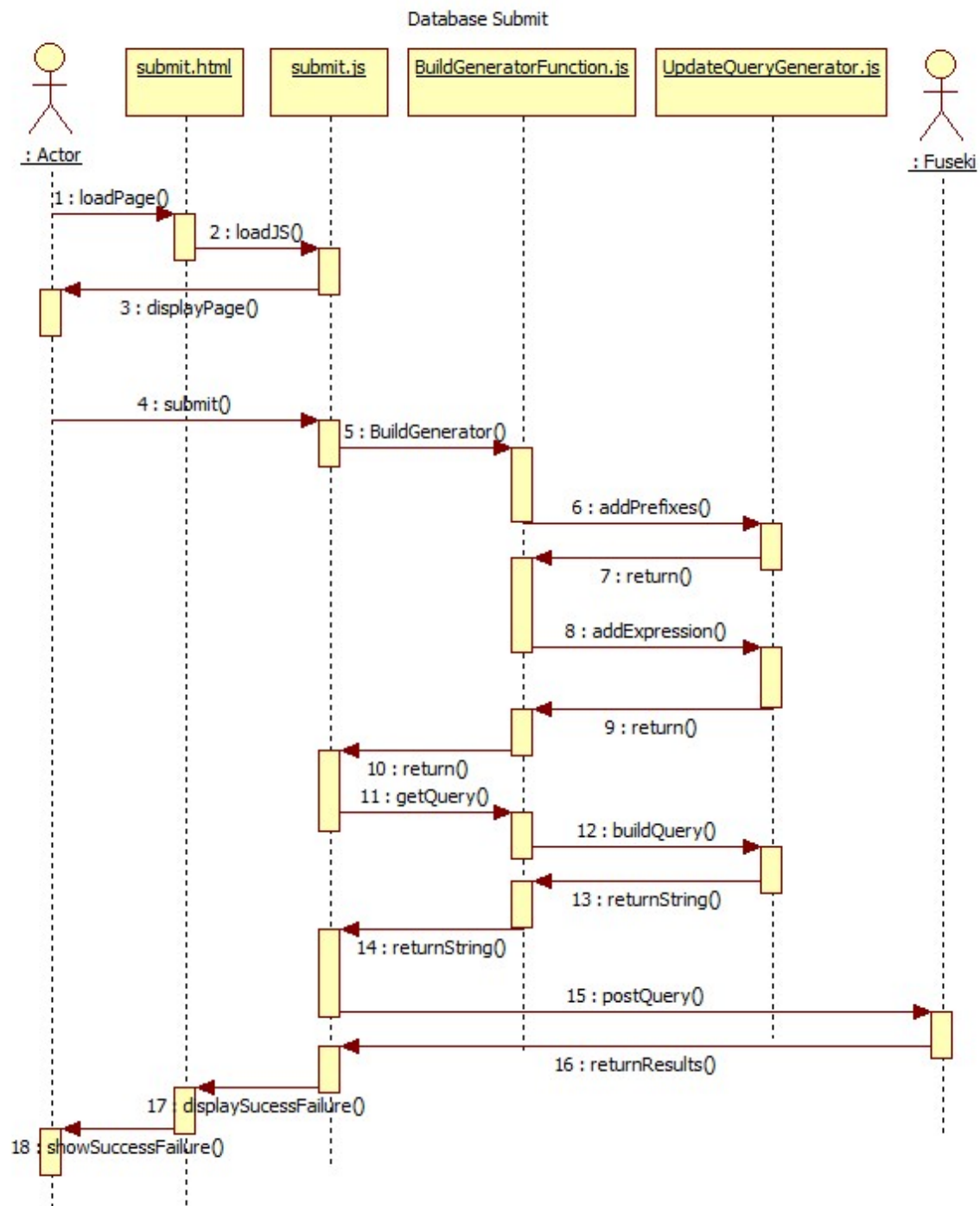
Adding Modular Fields



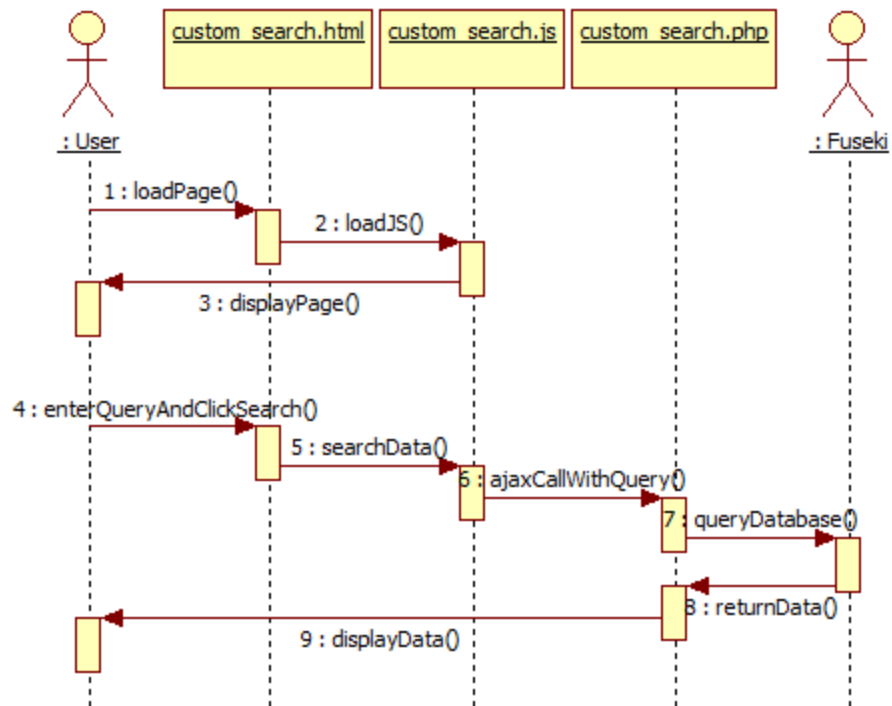
Removing Modular Fields



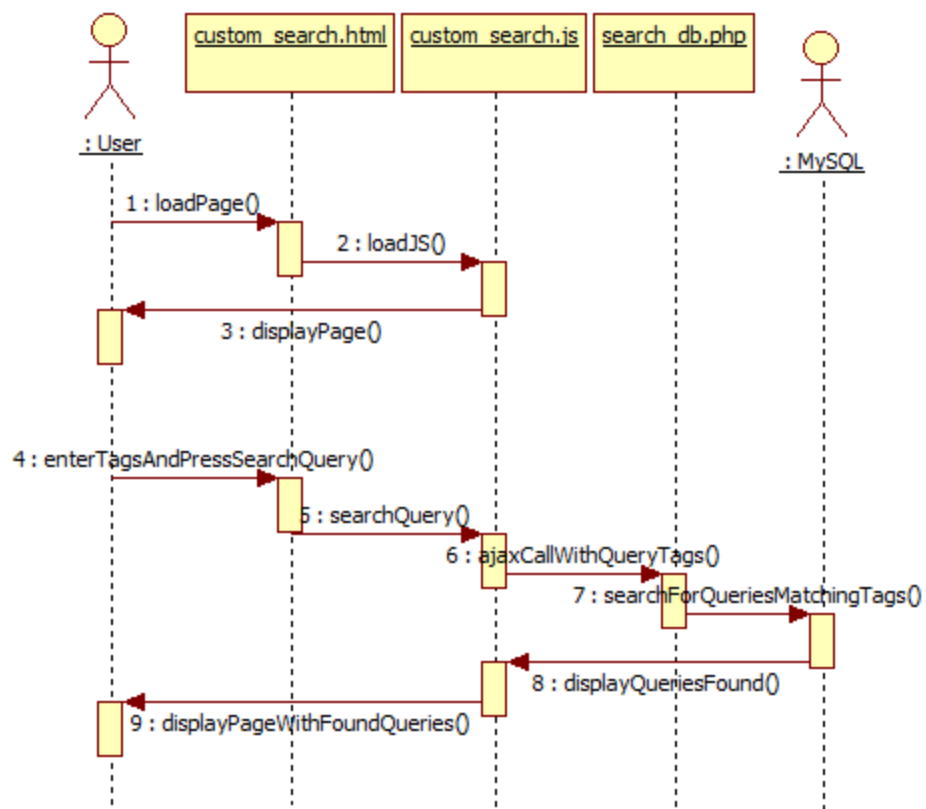




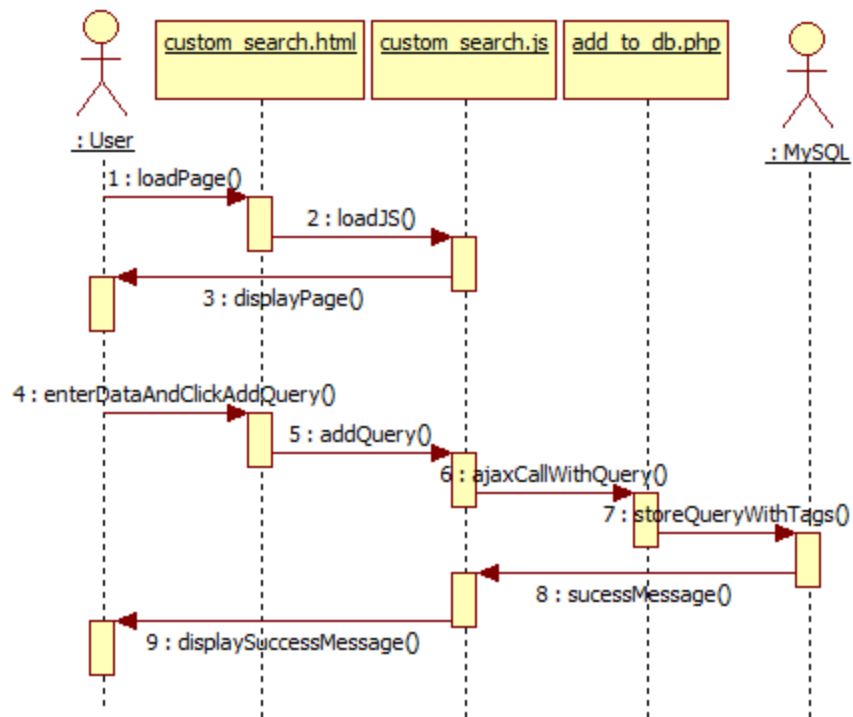
Search Custom Query

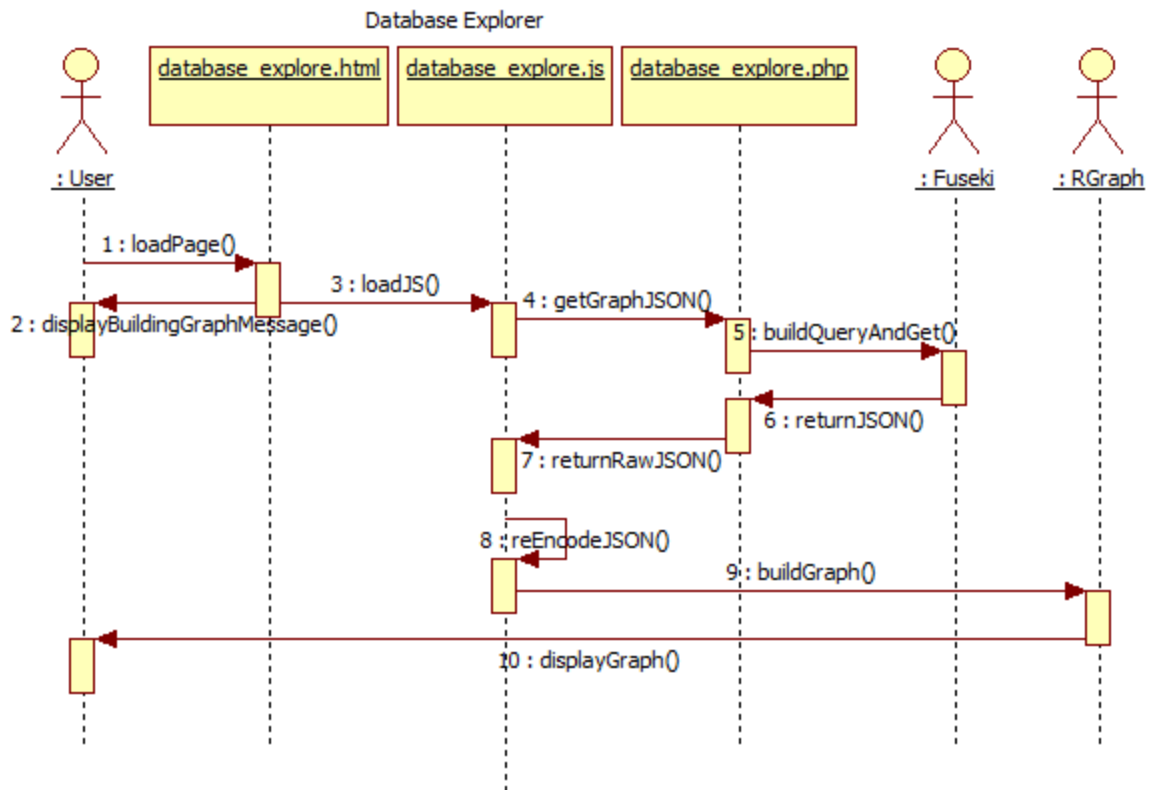


Search For A Custom Query



Storing Custom Query





Appendix F – Documented Class interfaces (code) and constraints.

UpdateQueryGenerator.js

```

var type;
var prefixes;
var expressions;

var UpdateQueryGenerator = function UpdateQueryGenerator()
{
  //Description : sets the type of the generator
  //Input : QueryType enum value
  //Output : N/A
  //Pre-Condition : QueryJS library must be loaded
  //Post-Condition : The type of the generator is set to the type supplied
  UpdateQueryGenerator.prototype.setType = function(type)

```



```
{}
```

```
//Description : add an expression object to the list of expressions of the generator
```

```
//Input : an Expression object
```

```
//Output : N/A
```

```
//Pre-Condition : QueryJS library must be loaded
```

```
//Post-Condition : expression is added to the generator
```

```
UpdateQueryGenerator.prototype.addExpression = function(exp)
```

```
{}
```

```
//Description : add a Prefix object to the list of prefixes of the generator
```

```
//Input : a Prefix object
```

```
//Output : N/A
```

```
//Pre-Condition : QueryJS library must be loaded
```

```
//Post-Condition : prefix is added to the generator
```

```
UpdateQueryGenerator.prototype.addPrefix = function(prefix) //make sure it is the same prefix  
object as the one the expression is using
```

```
{}
```

```
//Description : creates a valid update SPARQL query
```

```
//Input : n/a
```

```
//Output : a string
```

```
//Pre-Condition : prefixes and expressions are not null or expressions is not empty
```

```
//Post-Condition : the string returned is a valid SPARQL update query
```

```
UpdateQueryGenerator.prototype.buildQuery = function()
```

```
{}
```

```
//Description : returns a valid SPARQL update query
```

```
//Input : N/A
```

```
//Output : a string that is a SPARQL query
```

```
//Pre-Condition : buildQuery has been implemented
```

```
//Post-Condition : a valid SPARQL update query is returned
```

```
UpdateQueryGenerator.prototype.getQuery = function()
```

```
{}
```

SelectQueryGenertor.js

```

-----
var type; //the type of the generator, is QueryType enum value
var prefixes;// an array of Prefix objects
var expressions;// an array of Expression objects

var SelectQueryGenerator = function SelectQueryGenerator()
{
//Description : sets the type of the generator
//Input : QueryType enum value
//Output : N/A
//Pre-Condition : QueryJS library must be loaded
//Post-Condition : The type of the generator is set to the type supplied
SelectQueryGenerator.prototype.setType = function(type)
{
//Description : add an expression object to the list of expressions of the generator
//Input : an Expression object
//Output : N/A
//Pre-Condition : QueryJS library must be loaded
//Post-Condition : expression is added to the generator
SelectQueryGenerator.prototype.addExpression = function(exp)
{
//Description : add a Prefix object to the list of prefixes of the generator
//Input : a Prefix object
//Output : N/A
//Pre-Condition : QueryJS library must be loaded
//Post-Condition : prefix is added to the generator
SelectQueryGenerator.prototype.addPrefix = function(prefix) //make sure it is the same prefix
object as the one the expression is using
{
//Description : creates a valid select SPARQL query
//Input : n/a
//Output : a string

```

//Pre-Condition : prefixes and expressions are not null or expressions is not empty

//Post-Condition : the string returned is a valid SPARQL select query

SelectQueryGenerator.prototype.buildQuery = function()

{}

//Description : returns a valid SPARQL query

//Input : N/A

//Output : a string that is a SPARQL query

//Pre-Condition : buildQuery has been implemented

//Post-Condition : a valid SPARQL query is returned

SelectQueryGenerator.prototype.getQuery = function()

{}

BuildGeneratorFunction.js

//Description : A function that takes in the parts of a SPARQL query and creates a generator from the parts

//Input : data : an array whose elements are an array of strings and length 4 with the elements being subject, prefix shortcode, prefix definition, and object in that order

// pref : an array whose elements are an array of strings and length 2 with the elements being prefix shortcode, prefix definition

// type : a string or QueryType enum value that determines the type of generator being created

//Output : A query generator (either SelectQueryGenerator or UpdateQueryGenerator depending on the value of type given)

//Pre-Condition : The QueryJS.js library has been loaded

//Post-Condition : The function returns a Query generator

var BuildGenerator = function(data, pref, type)

{}

submit.html inner js

//Global variables for autocompletion and the evidence graph.

var currentValue = 2;

```
var currentSize = 1;
var autoComplete_prefix = null;
var autoComplete_predicate = null;
var g;

//Global variables for handling confidence intervals.
var minimumConfidence = 100;
var evidenceExists = false;
var evidenceFields = "";
var textConcat = "";
var textDropDowns = "";
var textData = "";
var html;
```

```
//Class definition for an empty userInterface;
function userInterface() {}
```

```
/*
```

Description: Initialize the size value on the page and the gauge variable.

Pre-Condition: The currentSize, the input element with id "size" exists and the div element with id "gauge" exists.

Post-Condition: The input element with id "size" has the initial UI size and the "gauge" id element has basic settings.

```
*/
```

```
userInterface.prototype.initialSetup = function()
{
}
```

```
/*
```

Description: Add a modular element to the page for an extra entry.

Pre-Condition: The div element with id "holder" exists.

Post-Condition: The input element with id "size" has the current UI size and the div element with id "holder" has a new element.

*/

```
userInterface.prototype.performAdd = function()
```

```
{}
```

/*

Description: Remove the last modular element from the page (assuming there is more than one)

Pre-Condition: The elements with id "xtra-clear, xtra-prefix, x-tra-select,x-tra-text-input, and size" exist.

Post-Condition: The elements with id "xtra-clear, xtra-prefix, x-tra-select,x-tra-text-input" no longer exist.

The input element with "size" contains the current UI element count.

*/

```
userInterface.prototype.performSubtract = function()
```

```
{}
```

/*

Description: Handler for change events for select fields.

Pre-Condition: Page is loaded.

Post-Condition: UI changes ("drop!"/ "short-code:prefix"/ "prefix-split") have occurred.

Post-Condition: Confidence evaluation has occurred.

*/

```
userInterface.prototype.selectionChanged = function()
```

```
{}
```

/*

Description: Verifies if evidence is existing. If it is, recalculates the evidence interval.

Pre-Condition: Page is loaded.

Post-Condition: Evidence data is extracted into variables.

Post-Condition: UI changes ("text") has occurred.

*/

```
userInterface.prototype.verifySelected = function( index )
```

```
{};
```

```
/*
```

Description: Extracts page info for confidence intervals.

Pre-Condition: Confidence has been found in the page.

Post-Condition: Data from page extracted.

```
*/
```

```
userInterface.prototype.calculateConfidence = function()
```

```
{}
```

```
/*
```

Description: Calculates confidence ranking.

Pre-Condition: Data from page extracted into variables.

Post-Condition: Number from 0 to 100 assigned for confidence.

```
*/
```

```
userInterface.prototype.finalizeConfidence = function()
```

```
{}
```

```
/*
```

Description: Extractor for auto-complete data.

Pre-Condition: Page is loaded.

Post-Condition: autoComplete_prefix and autoComplete_predicate have autocomplete items from the RDF database.

```
*/
```

```
userInterface.prototype.getAutocomplete = function()
```

```
{}
```

```
/*
```

Description: Imports database from XML.

Pre-Condition: Page is loaded and database

Post-Condition: autoComplete_prefix and autoComplete_predicate have autocomplete items from the RDF database.

```
*/
```

```
userInterface.prototype.importDatabase = function()
{
}
/*
```

Description: Input Sanitizing

Pre-Condition: String is not null.

Post-Condition: Sanitized string for query creation.

```
*/
```

```
userInterface.prototype.sanitizeInput = function(str)
{
}
```

```
/*
```

Description: Submission method.

Pre-Condition: Page loaded.

Post-Condition: Table created with search results.

```
*/
```

```
userInterface.prototype.submit = function()
{
});
}
```

```
$(document).ready(
    function()
    {
        var UI = new userInterface();
        UI.initialSetup();

        //Deal with adds.
        $( "#plus" ).click(UI.performAdd);

        //Deal with removals.
        $( "#minus" ).click(UI.performSubtract);

        $("#submit").click(UI.submit);
    }
);
```

```

//Deals with bulk import disabling if it's not supported in other browsers.
if (typeof window.FileReader !== 'function') {
    $("#bulk-import").attr("disabled", "true");
}
else
{
    $("#bulk-import").click(UI.importDatabase);
}

//Get all selected values to be printed.
$("#div").on('change', 'select.form-control', UI.selectionChanged);
$("#div").on('change', 'input.form-control', UI.selectionChanged);

//Attach auto-complete handlers to each input field on click.
UI.getAutocomplete();
$( "div" ).on( "click", "input.form-control" , function() {
    if($(this).attr("name").indexOf("prefix") > -1 &&
autoComplete_prefix.length > 0)
        $(this).autocomplete({
            source: autoComplete_prefix,
            select: function(event, ui) {
                setTimeout(function() {
                    UI.inputChanged();
                }, 500);
            }
        });
    if($(this).attr("name").indexOf("data") > -1 &&
autoComplete_predicate.length > 0)
        $(this).autocomplete({
            source: autoComplete_predicate,
            select: function(event, ui) {
                setTimeout(function() {

```



```

                                UI.inputChanged();
                                }, 500);
                                }
                                });
                                });
                                }
                                );

```

submit.html inner js

```

//Global variables for autocompletion and the evidence graph.

```

```

var currentValue = 2;
var currentSize = 1;
var autoComplete_prefix = null;
var autoComplete_predicate = null;
var g;

```

```

//Global variables for handling confidence intervals.

```

```

var minimumConfidence = 100;
var evidenceExists = false;
var evidenceFields = "";
var textConcat = "";
var textDropDowns = "";
var textData = "";
var html;

```

```

//Class definition for an empty userInterface;

```

```

function userInterface() {}

```

```

/*

```

Description: Initialize the size value on the page and the gauge variable.

Pre-Condition: The currentSize, the input element with id "size" exists and the div element with id "gauge" exists.

Post-Condition: The input element with id "size" has the initial UI size and the "gauge" id element has basic settings.

*/

```
userInterface.prototype.initialSetup = function()
```

```
{}
```

/*

Description: Add a modular element to the page for an extra entry.

Pre-Condition: The div element with id "holder" exists.

Post-Condition: The input element with id "size" has the current UI size and the div element with id "holder" has a new element.

*/

```
userInterface.prototype.performAdd = function()
```

```
{}
```

/*

Description: Remove the last modular element from the page (assuming there is more than one)

Pre-Condition: The elements with id "xtra-clear, xtra-prefix, x-tra-select,x-tra-text-input, and size" exist.

Post-Condition: The elements with id "xtra-clear, xtra-prefix, x-tra-select,x-tra-text-input" no longer exist.

The input element with "size" contains the current UI element count.

*/

```
userInterface.prototype.performSubstract = function()
```

```
{}
```

/*

Description: Handler for change events for select fields.

Pre-Condition: Page is loaded.

Post-Condition: UI changes ("drop!"/ "short-code:prefix"/ "prefix-split") have occurred.

Post-Condition: Confidence evaluation has occurred.

*/

```
userInterface.prototype.selectionChanged = function()
```

```
{}
```

/*

Description: Verifies if evidence is existing. If it is, recalculates the evidence interval.

Pre-Condition: Page is loaded.

Post-Condition: Evidence data is extracted into variables.

Post-Condition: UI changes ("text") has occurred.

*/

```
userInterface.prototype.verifySelected = function( index )
```

```
{};
```

/*

Description: Extracts page info for confidence intervals.

Pre-Condition: Confidence has been found in the page.

Post-Condition: Data from page extracted.

*/

```
userInterface.prototype.calculateConfidence = function()
```

```
{}
```

/*

Description: Calculates confidence ranking.

Pre-Condition: Data from page extracted into variables.

Post-Condition: Number from 0 to 100 assigned for confidence.

*/

```
userInterface.prototype.finalizeConfidence = function()
```

```
{}
```

/*

Description: Extractor for auto-complete data.

Pre-Condition: Page is loaded.

Post-Condition: autoComplete_prefix and autoComplete_predicate have autocomplete items from the RDF database.

*/

```
userInterface.prototype.getAutocomplete = function()
```

```
{}
```

/*

Description: Imports database from XML.

Pre-Condition: Page is loaded and database

Post-Condition: autoComplete_prefix and autoComplete_predicate have autocomplete items from the RDF database.

*/

```
userInterface.prototype.importDatabase = function()
```

```
{}
```

/*

Description: Input Sanitizing

Pre-Condition: String is not null.

Post-Condition: Sanitized string for query creation.

*/

```
userInterface.prototype.sanitizeInput = function(str)
```

```
{}
```

/*

Description: Submission method.

Pre-Condition: Page loaded.

Post-Condition: Table created with search results.

*/

```
userInterface.prototype.submit = function()
```

```
{}
```

```
$(document).ready(
```

```
    function()
```

```
{
```

```

var UI = new userInterface();
UI.initialSetup();

//Deal with adds.
$("#plus").click(UI.performAdd);

//Deal with removals.
$("#minus").click(UI.performSubtract);

$("#submit").click(UI.submit);

//Deals with bulk import disabling if it's not supported in other browsers.
if (typeof window.FileReader !== 'function') {
    $("#bulk-import").attr("disabled", "true");
}
else
{
    $("#bulk-import").click(UI.importDatabase);
}

//Get all selected values to be printed.
$("#div").on('change', 'select.form-control', UI.selectionChanged);
$("#div").on('change', 'input.form-control', UI.selectionChanged);

//Attach auto-complete handlers to each input field on click.
UI.getAutocomplete();
$("#div").on("click", "input.form-control", function() {
    if($("#this").attr("name").indexOf("prefix") > -1 &&
autoComplete_prefix.length > 0)
        $("#this").autocomplete({
            source: autoComplete_prefix,
            select: function(event, ui) {

```

```

                                setTimeout(function() {
                                    UI.inputChanged();
                                }, 500);
                            }
                        });
                    if($(this).attr("name").indexOf("data") > -1 &&
autoComplete_predicate.length > 0)
                        $(this).autocomplete({
                            source: autoComplete_predicate,
                            select: function(event, ui) {
                                setTimeout(function() {
                                    UI.inputChanged();
                                }, 500);
                            }
                        });
                });
            }
        );

```

database_explorer.html inner js

```

//Class definition for an empty userInterface;

```

```

function userInterface() {}

```

```

/*

```

Description: Draws the entire database graph.

Pre-Condition: Page is loaded.

Post-Condition: A graph is created on the page.

```

*/

```

```

userInterface.prototype.drawGraph = function(value)

```

```

{

```

```

$(document).ready(

```

```

//Get the entire database.
$.ajax({
    type : "GET",
    url : "http://iie-dev.cs.fiu.edu/scripts/bulk_export.php",
    success: function(data)
    {
        //Parse the XML into CSV format.
        var value = "";
        $(data).find("result").each(function()
        {
            $(this).find("binding").each(function()
            {
                var targs = $(this).first().text().split("/");
                value +=
targs[targs.length-1].replace("<", "").replace(">", "").trim() + "," ;
            });
        });

        //Remove extra commas and draw the graph.
        value = value.substring(0, value.length - 1);
        var UI = new userInterface();
        UI.drawGraph(value);
    }
});

```

```

custom_search.html inner js
<script type="text/javascript">
$(document).ready(
    //Basic Functionality.
    function()

```

```

{
    $("#tags").val("untaggedQuery");
    $( "#add" ).click(

        //Store the data.
        function()
        {
            var tagData = $("#tags").val();
            var queryData = $("#query").val();

            //Ajax call to adding-query to DB.
            $.ajax({
                type : "POST",
                data : {query : queryData, tags : tagData},
                url : "http://iie-dev.cs.fiu.edu/scripts/add-to-db.php",
                success: function(data){
                    if(data.indexOf('unsuccessful') < 0)
                    {
                        console.log(data);
                        $("#query-area").append('<div class="alert
alert-success" align="center" role="alert">Query Successfully Stored</div>');
                        setTimeout(function () {

                            $("#query-area").children().remove('div');

                        }, 5000);
                    }
                }
            });
        }
    );

    $( "#search" ).click(

```



```

//Retrieve the data.
function()
{
    var tagData = $("#tags").val();

    //Ajax call to retrieve query from DB.
    $.ajax({
        type : "POST",
        data : {tags : tagData},
        url : "http://iie-dev.cs.fiu.edu/scripts/search-db.php",
        success: function(data){
            $("#query").val(data);
        }
    });
}

);
}

```

Appendix G – Documented code for test drivers and stubs.

CustomSearchTests.java

```

-----
import java.net.URL;
import java.util.UUID;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

```

```

import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class CustomSearchTests
{
    String baseURL = "http://iie-dev.cs.fiu.edu/";
    String testURL = baseURL + "custom_search.html";
    private WebDriver driver;

    @BeforeMethod
    public void setUp() throws Exception {

        DesiredCapabilities capabilities = DesiredCapabilities.chrome();
        capabilities.setCapability("version", "35");
        capabilities.setCapability("platform", Platform.XP);
        // Create the connection to Sauce Labs to run the tests
        driver = new RemoteWebDriver(
            new
URL("http://lazherrera:a46f025c-0e5c-495a-a403-da422d5c60b0@ondemand.saucelabs.com:80/wd/hu
b"),
        capabilities);
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    }

    @Test
    public void verifyBasicUILayout() throws Exception {
        driver.get(testURL);

        //Confirm layout.
        Thread.sleep(200);
        verifyExistence(driver, driver.findElement(By.className("panel-heading")));
        verifyExistence(driver, driver.findElement(By.className("panel-body")));

        //Confirm input fields.
        verifyExistence(driver, driver.findElement(By.id("query")));
        verifyExistence(driver, driver.findElement(By.id("tags")));

        //Confirm UI/Buttons.
        verifyExistence(driver, driver.findElement(By.id("add")));
    }
}

```

```

verifyExistence(driver, driver.findElement(By.id("search")));
verifyExistence(driver, driver.findElement(By.id("submit")));

    }

    @Test
    public void verifyDataStorage() throws Exception {
        driver.get(testURL);

        //Generate a random UUID for this test.
        Thread.sleep(200);
        String target = UUID.randomUUID().toString();

        //Add a real query to the query location, tag and store it.
        driver.findElement(By.id("query")).clear();
        driver.findElement(By.id("query")).sendKeys("SELECT ?x ?y ?z WHERE { ?x ?y ?z");
    });

        driver.findElement(By.id("tags")).clear();
        driver.findElement(By.id("tags")).sendKeys(target);
        driver.findElement(By.id("add")).click();
        Thread.sleep(5000);

        //Recall the query.
        driver.get(testURL);

        Thread.sleep(200);
        driver.findElement(By.id("tags")).clear();
        driver.findElement(By.id("tags")).sendKeys(target);
        driver.findElement(By.id("search")).click();
        Thread.sleep(5000);

        //Verify the query's storage.
        if(driver.findElement(By.id("query")).getText().length() == 0)
            throw new Exception("Storage failure for custom queries.");

    }

    @AfterMethod
    public void tearDown() throws Exception {
        driver.quit();
    }

```

```

        public void verifyExistence(WebDriver driver, WebElement element) throws
InterruptedException {
            Thread.sleep(1000);
            for (int i = 0; i < 2; i++) {
                JavascriptExecutor js = (JavascriptExecutor) driver;
                js.executeScript("arguments[0].setAttribute('style', arguments[1]);",
                    element, "color: yellow; border: 2px solid yellow;");
            }
        }
    }
}

```

DatabaseExplorerTests.java

```

-----
import java.net.URL;
import java.util.List;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class DatabaseExplorerTests
{
    String baseURL = "http://iie-dev.cs.fiu.edu/";
    String testURL = baseURL + "database_explorer.html";

    private WebDriver driver;

    @BeforeMethod
    public void setUp() throws Exception {

        DesiredCapabilities capabilities = DesiredCapabilities.chrome();
        capabilities.setCapability("version", "35");
    }
}

```

```

        capabilities.setCapability("platform", Platform.XP);
        // Create the connection to Sauce Labs to run the tests
        driver = new RemoteWebDriver(
            new
URL("http://lazherrera:a46f025c-0e5c-495a-a403-da422d5c60b0@ondemand.saucelabs.com:80/wd/hu
b"),
        capabilities);
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    }

```

```

@Test
public void verifyRendering() throws Exception {
    driver.get(testURL);

    //Get current data.
    Thread.sleep(2000);
    List<WebElement> links = driver.findElements(By.className("link"));
    List<WebElement> nodes = driver.findElements(By.className("node"));

    //Verify attributes for links.
    for (WebElement current : links)
    {
        if(
            current.getAttribute("x1") == "0" || current.getAttribute("x2") == "0" ||
            current.getAttribute("y1") == "0" || current.getAttribute("y2") == "0")
        {
            throw new Exception ("Lines being incorrectly rendered.");
        }
    }

    //Verify attributes for nodes.
    for (WebElement current : nodes)
    {
        if(
            current.getAttribute("transform").equals(""))
        {
            throw new Exception ("Nodes being incorrectly rendered.");
        }
    }
}

@AfterMethod
public void tearDown() throws Exception {
    driver.quit();
}

```

```
}
```

SearchTests.java

```
-----
import org.testng.annotations.AfterMethod;
import org.testng.annotations.Test;
import org.testng.annotations.BeforeMethod;

import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;

public class SearchTests {
    String baseURL = "http://iie-dev.cs.fiu.edu/";
    String testURL = baseURL + "search.html";

    private WebDriver driver;

    @BeforeMethod
    public void setUp() throws Exception {

        DesiredCapabilities capabilities = DesiredCapabilities.chrome();
        capabilities.setCapability("version", "35");
        capabilities.setCapability("platform", Platform.XP);
        // Create the connection to Sauce Labs to run the tests
        driver = new RemoteWebDriver(
            new URL(

"http://lzherrera:a46f025c-0e5c-495a-a403-da422d5c60b0@ondemand.saucelabs.com:80/wd/hub"),
            capabilities);
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    }
}
```

```

@Test
public void verifyBasicUILayout() throws Exception {
    driver.get(testURL);

    // Confirm layout.
    Thread.sleep(200);
    verifyExistence(driver,
        driver.findElement(By.className("panel-heading")));
    verifyExistence(driver, driver.findElement(By.className("panel-body")));

    // Confirm input fields.
    verifyExistence(driver, driver.findElement(By.name("prefixName")));
    verifyExistence(driver, driver.findElement(By.name("subject")));
    verifyExistence(driver, driver.findElement(By.name("data-prefix1")));
    verifyExistence(driver, driver.findElement(By.name("data-pred1")));
    verifyExistence(driver, driver.findElement(By.name("data-1")));

    // Confirm UI/Buttons.
    verifyExistence(driver, driver.findElement(By.id("plus")));
    verifyExistence(driver, driver.findElement(By.id("minus")));
    verifyExistence(driver, driver.findElement(By.id("submit")));
    verifyExistence(driver, driver.findElement(By.id("custom-search")));
    verifyExistence(driver, driver.findElement(By.id("bulk-export")));

}

```

```

@Test
public void verifyModularUI() throws Exception {
    driver.get(testURL);

    // Verify initial state.
    Thread.sleep(200);
    verifyExistence(driver, driver.findElement(By.id("plus")));
    verifyExistence(driver, driver.findElement(By.id("minus")));

    // Add a new element set.
    driver.findElement(By.id("plus")).click();
    Thread.sleep(5000);

    // Verify the original set exists.
    verifyExistence(driver, driver.findElement(By.name("data-prefix1")));
    verifyExistence(driver, driver.findElement(By.name("data-pred1")));
    verifyExistence(driver, driver.findElement(By.name("data-1")));
}

```

```

// Verify the new element set.
verifyExistence(driver, driver.findElement(By.name("data-prefix2")));
verifyExistence(driver, driver.findElement(By.name("data-pred2")));
verifyExistence(driver, driver.findElement(By.name("data-2")));

// Add a new element set.
driver.findElement(By.id("minus")).click();
Thread.sleep(5000);

// Verify the original set exists.
verifyExistence(driver, driver.findElement(By.name("data-prefix1")));
verifyExistence(driver, driver.findElement(By.name("data-pred1")));
verifyExistence(driver, driver.findElement(By.name("data-1")));

try {
    if (driver.findElement(By.name("data-prefix2")) != null
        || driver.findElement(By.name("data-pred1")) != null
        || driver.findElement(By.name("data-1")) != null) {
        throw new Exception("A dynamic UI element was not removed.");
    }
} catch (Exception e) {
    // A Successful test ends with us in this section.
    // Ending in the try section is a test fail.
}

// Add a new element set.
driver.findElement(By.id("plus")).click();
Thread.sleep(2000);

// Verify the new element set is correctly named.
verifyExistence(driver, driver.findElement(By.name("data-prefix2")));
verifyExistence(driver, driver.findElement(By.name("data-pred2")));
verifyExistence(driver, driver.findElement(By.name("data-2")));

}

// TODO - Find Solution
@Test
public void verifyAutocompleteCapabilities() throws Exception {
    // Verify data-level autocomplete.
    driver.get(testURL);

```



```

Thread.sleep(200);
driver.findElement(By.name("data-1")).sendKeys("a");
Thread.sleep(200);
driver.findElement(By.className("ui-menu-item"));

// Verify prefix-level autocomplete.
driver.get(testURL);
Thread.sleep(200);
driver.findElement(By.name("prefixName")).sendKeys("a");
Thread.sleep(200);
driver.findElement(By.className("ui-menu-item"));

```

```

}

```

```

@Test

```

```

public void verifyBulkExporting() throws Exception {

```

```

    // This test is wholly useless but it will at least serve as a warning
    // if the button disappears all of a sudden.
    driver.get(testURL);
    Thread.sleep(200);
    driver.findElement(By.id("bulk-export")).click();
    Thread.sleep(2000);

```

```

}

```

```

@AfterMethod

```

```

public void tearDown() throws Exception {
    driver.quit();
}

```

```

public void verifyExistence(WebDriver driver, WebElement element)
    throws InterruptedException {

```

```

    for (int i = 0; i < 2; i++) {
        JavascriptExecutor js = (JavascriptExecutor) driver;
        js.executeScript(
            "arguments[0].setAttribute('style', arguments[1]);",
            element, "color: yellow; border: 2px solid yellow;");
    }

```

```

}

```

```

}

```

SubmitTests.java

```
-----
import java.net.URL;
import java.util.concurrent.TimeUnit;

import org.openqa.selenium.JavascriptExecutor;
import org.openqa.selenium.Platform;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.remote.DesiredCapabilities;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.openqa.selenium.support.ui.Select;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class SubmitTests {
    String baseURL = "http://iie-dev.cs.fiu.edu/";
    String testURL = baseURL + "submit.html";

    private WebDriver driver;

    @BeforeMethod
    public void setUp() throws Exception {

        DesiredCapabilities capabilities = DesiredCapabilities.chrome();
        capabilities.setCapability("version", "35");
        capabilities.setCapability("platform", Platform.XP);
        // Create the connection to Sauce Labs to run the tests
        driver = new RemoteWebDriver(
            new URL(

"http://lzherrera:a46f025c-0e5c-495a-a403-da422d5c60b0@ondemand.saucelabs.com:80/wd/hub"),
            capabilities);
        driver.manage().window().maximize();
        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
    }

    @Test
```

```

public void verifyBasicUILayout() throws Exception {
    driver.get(testURL);

    // Confirm layout.
    Thread.sleep(200);
    verifyExistence(driver,
        driver.findElement(By.className("panel-heading")));
    verifyExistence(driver, driver.findElement(By.className("panel-body")));

    // Confirm input fields.
    verifyExistence(driver, driver.findElement(By.name("reporterName")));
    verifyExistence(driver, driver.findElement(By.name("prefixName")));
    verifyExistence(driver, driver.findElement(By.name("subject")));
    verifyExistence(driver, driver.findElement(By.name("data-prefix1")));
    verifyExistence(driver, driver.findElement(By.name("data-pred1")));
    verifyExistence(driver, driver.findElement(By.name("data-1")));

    // Confirm UI/Buttons.
    verifyExistence(driver, driver.findElement(By.id("plus")));
    verifyExistence(driver, driver.findElement(By.id("minus")));
    verifyExistence(driver, driver.findElement(By.id("submit")));
    verifyExistence(driver, driver.findElement(By.id("bulk-import")));

}

```

```

@Test
public void verifyModularUI() throws Exception {
    driver.get(testURL);

    // Verify initial state.
    Thread.sleep(200);
    verifyExistence(driver, driver.findElement(By.id("plus")));
    verifyExistence(driver, driver.findElement(By.id("minus")));

    // Add a new element set.
    driver.findElement(By.id("plus")).click();
    Thread.sleep(5000);

    // Verify the original set exists.
    verifyExistence(driver, driver.findElement(By.name("data-prefix1")));
    verifyExistence(driver, driver.findElement(By.name("data-pred1")));
    verifyExistence(driver, driver.findElement(By.name("data-1")));
}

```

```

// Verify the new element set.
verifyExistence(driver, driver.findElement(By.name("data-prefix2")));
verifyExistence(driver, driver.findElement(By.name("data-pred2")));
verifyExistence(driver, driver.findElement(By.name("data-2")));

// Subtract a new element set.
driver.findElement(By.id("minus")).click();
Thread.sleep(5000);

// Verify the original set exists.
verifyExistence(driver, driver.findElement(By.name("data-prefix1")));
verifyExistence(driver, driver.findElement(By.name("data-pred1")));
verifyExistence(driver, driver.findElement(By.name("data-1")));

try {
    if (driver.findElement(By.name("data-prefix2")) != null
        || driver.findElement(By.name("data-pred1")) != null
        || driver.findElement(By.name("data-1")) != null) {
        throw new Exception("A dynamic UI element was not removed.");
    }
} catch (Exception e) {
    // A Successful test ends with us in this section.
    // Ending in the try section is a test fail.
}

// Add a new element set.
driver.findElement(By.id("plus")).click();
Thread.sleep(5000);

// Verify the new element set is correctly named.
verifyExistence(driver, driver.findElement(By.name("data-prefix2")));
verifyExistence(driver, driver.findElement(By.name("data-pred2")));
verifyExistence(driver, driver.findElement(By.name("data-2")));

}

@Test
public void verifyEvidenceEvaluation() throws Exception {
    driver.get(testURL);

    // Add an element.
    Thread.sleep(200);
    driver.findElement(By.name("data-1")).sendKeys("Google");

```

```

        driver.findElement(By.id("plus")).click();
        Thread.sleep(2000);

        // Add the evidence element.
        driver.findElement(By.name("data-2")).sendKeys("http://www.google.com");
        Select sel = new Select(driver.findElement(By.name("data-pred2")));
        sel.selectByValue("evidence");
        Thread.sleep(2000);

        // Verify gauge existence.
        verifyExistence(driver, driver.findElement(By.id("wordsMatched")));
        verifyExistence(driver, driver.findElement(By.id("wordsText")));
        verifyExistence(driver, driver.findElement(By.id("wordsNotMatched")));
        verifyExistence(driver, driver.findElement(By.id("confidence")));

    }

    // TODO - Find solution.
    @Test
    public void verifyAutocompleteCapabilities() throws Exception {
        // Verify data-level autocomplete.
        driver.get(testURL);
        Thread.sleep(200);
        driver.findElement(By.name("data-1")).sendKeys("a");
        Thread.sleep(200);
        driver.findElement(By.className("ui-menu-item"));

        // Verify prefix-level autocomplete.
        driver.get(testURL);
        Thread.sleep(200);
        driver.findElement(By.name("prefixName")).sendKeys("a");
        Thread.sleep(200);
        driver.findElement(By.className("ui-menu-item"));

    }

    @Test
    public void verifyBulkImporting() throws Exception {

        // This test is wholly useless but it will at least serve as a warning
        // if the button disappears all of a sudden.
        driver.get(testURL);
        driver.findElement(By.id("bulk-import")).click();
    }

```

```

    }

    @AfterMethod
    public void tearDown() throws Exception {
        driver.quit();
    }

    public void verifyExistence(WebDriver driver, WebElement element)
        throws InterruptedException {
        Thread.sleep(1000);
        for (int i = 0; i < 2; i++) {
            JavascriptExecutor js = (JavascriptExecutor) driver;
            js.executeScript(
                "arguments[0].setAttribute('style', arguments[1]);",
                element, "color: yellow; border: 2px solid yellow;");
        }
    }
}

```

SelectGeneratorDriver.js

```

-----
//Driver to test the SelectQueryGenerator and all classes included in it
var p = new Prefix("iie", "</fake/iie/types/>")
var sub = new rdfObject()
var pred = new Predicate(p, "related")
var obj = new rdfObject("android", true)
var e = new Expression(sub, pred, obj)
var abs = new SelectQueryGenerator()
abs.addPrefix(p)
abs.addExpression(e)
abs.getQuery()

```

UpdateGeneratorDriver.js

```

-----
//Driver to test the UpdateQueryGenerator and all classes included in it
var p = new Prefix("iie", "</fake/iie/types/>")
var sub = new rdfObject("<asd>")
var pred = new Predicate(p, "related")
var obj = new rdfObject("android", true)
var e = new Expression(sub, pred, obj)
var abs = new UpdateQueryGenerator()
abs.addPrefix(p)
abs.addExpression(e)

```

abs.getQuery()

Appendix H – Diary of meeting and tasks for the entire semester.

Date: 9/3/2014

Meeting Begins: 6:03PM

Meeting Ends: 6:45PM

Medium of Communication: Skype

Present Members: Eric, Jose, Lazaro

6:03PM - 6:15PM Eric (Formal Introductions, Intro to Inference Engine)

6:15PM - 6:30PM Use Case Elicitation

6:30PM - 6:33PM Architecture concerns and discussion

6:33PM - 6:33PM Meeting with client formally disbands

6:33PM - 6:45PM Jose and Lazaro discuss required documentation, recap about requirements.

Assignments:

Lazaro and Jose will install Mulgara and attempt to execute basic queries, we'll also be looking at bootstrap-based approach to generating simple queries from a web form.

Lazaro and Jose will also begin writing the required initial drafts for our Feasibility Study, Requirements Document and Project Plan.

Date: 9/6/2014

Meeting Begins: 11:30AM

Meeting Ends: 6:00PM

Medium of Communication: Physical Meeting

Present Members: Jose, Lazaro

11:30AM - 1:30PM: Trello Board Upgrades and Discussion

1:30PM - 4:30PM: Feasibility Study Documentation

4:30PM - 5:30PM: Requirements Documentation

Date: 9/21/2014

Meeting Begin: 10:00 AM

Meeting Ends: 3:00 PM

Medium: Physical Meeting

Present members: Jose, Lazaro

In this meeting Lazaro started work on implementing the UI of the website. The initial UIs were done. He also installed Mulgara and testing it to make sure it worked. Jose begin looking for a library to do the REST calls that the UI needed to make in order to make queries. He found a library and started testing it to make sure that it had everything that was needed for the project

Date: 10/4/2014

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

In this meeting Lazaro worked mainly on getting the confidence ranking to show up on the forms when evidence was added. Apart from that, he also included a lot of bug fixes for code that was previously submitted. Jose worked on the PHP scripts that are going to generate SPARQL queries based on what users entered in the forms. He also worked on some bug fixes for the scripts.

Date: 10/19/2014

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

In this meeting Lazaro began working on the custom query subsystems of the system. He also continued to work on the previous forms. Jose continued testing and developing the PHP scripts so that they generated valid SPARQL queries and correctly connected to the server.

Date: 10-28-2014

Meeting Begins: 4:30PM

Meeting Ends: 5:15PM

Medium of Communication: Skype

Present Members: Jose, Eric

4:30 - 4:40 Discussed what parts of the system I was showing today

4:40 - 4:50 Talked about the data entry form and how predicates worked

4:50 - 4:55 Discussed how predicates currently worked and changes to be made

4:55 - 5:05 Talked about the data search form and some specific queries

5:10 - 5:15 Talked about some good ontologies to use

Date: 10-31-2014

Meeting Begins: 11:30

Meeting Ends: 1:30

Medium of Communication: Physical Meeting

Present Members: Jose, Eric, Lazaro

11:30 - 11:50 Tour of Eric's workplace

11:50 - 12:20 Lazaro showed Eric some of the features of the system and asked for comments

12:20 - 12:30 Eric talked to us about how he would like to be implemented

12:30 - 12:50 We discussed about some good ontologies to use and which default ontologies we should use

12:50 - 12:55 Jose asked about the web crawler and if Eric has any recommendations

12:55 - 1:10 Eric explained what he wanted the crawler to do as we thought it had a different function

1:10 - 1:30 We got some minor features that Eric wanted us to implement and also got comments on the current system

Date: 11/1/2014

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

In this meeting, we discussed the need to change our core technology. We decided it was necessary to replace Mulgara with something that supported everything that was necessary for the project. Lazaro continued work on the front end forms. Jose continued work on the scripts and modified them for the new back end triple store that was going to be used.

Date: 11/17/2014

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

Jose modified the PHP scripts so that they returned XML and prepared the PHP scripts to be retired. He also began work on the javascript library that were going to replace the PHP script for query generation. Lazaro worked on getting auto-complete for the different text fields working. He also added the base code import and export.

Date: 12/1/2014

Begin : 11:00 AM

End : 4:00 PM

Medium : Physical Meeting

Present Members : Jose, Lazaro

Lazaro began the testing of the front end part of the system. Jose added some of the javascript files for query generation.

Date: 12/6/2014

Begin : 11:00 AM

End : 6:00 PM

Medium : Physical Meeting

Both Jose and Lazaro finalized their code for their respective parts and began integrating the parts together. Both also fixed any bugs that came up from integration in their respective parts. Both continued to test their parts and tested the integration.

References

[1] - S. Karnouskos: *Stuxnet Worm Impact on Industrial Cyber-Physical System Security*. In: *37th Annual Conference of the IEEE Industrial Electronics Society (IECON 2011)*, Melbourne, Australia, 7-10 Nov 2011. Retrieved 20 Apr 2014.

- [2] - Jack Rusher, [Semantic Web Advanced Development for Europe](#) (SWAD-Europe), Workshop on Semantic Web Storage and Retrieval - Position Papers
- [3] - <http://www.w3.org/TR/PR-rdf-syntax/> "Resource Description Framework (RDF) Model and Syntax Specification
- [4] - Berners-Lee, Tim; James Hendler; Ora Lassila (May 17, 2001). "[The Semantic Web](#)". *Scientific American Magazine*
- [5] - Hebel, John; Fisher, Matthew; Blace, Ryan; Perez-Lopez, Andrew (2009). *Semantic Web Programming*. Indianapolis, Indiana: [John Wiley & Sons](#). p. 406. ISBN 978-0-470-41801-7