

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Table of Contents

- I. [Table of Contents](#)
- II. [Preface](#)
- III. [Overview of Product](#)
- IV. [Technical Specifications](#)
- V. [Installation Steps](#)
- VI. [Development](#)
- VII. [Features](#)
- VIII. [Troubleshooting Steps](#)
- IX. [Frequently Asked Questions](#)
- X. [Testing](#)

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Preface

Team:

Nishant Chandrashekar | UCD 2018 | nchandrandrashekar@ucdavis.edu
Front-End Developer and System Admin

Wonhee Park | UCD 2018 | wonpark@ucdavis.edu
Front-End Developer and Data Scientist

Sailesh Patnala | UCD 2018 | sgpatnala@ucdavis.edu
Full Stack Developer

Mithun Vijayasekar | UCD 2018 | mvijayasekar@ucdavis.edu
Back-End Developer

Client:

Dr. Carl Stahmer | cstahmer@ucdavis.edu
Director of Data and Digital Scholarship Department — UCD Shields Library

Mentors:

Dr. Xin Liu | xinliu@ucdavis.edu
Computer Science Professor, Faculty Advisor for ECS 193

Albara Ramli | arramli@ucdavis.edu
Computer Science Graduate Student, Teaching Assistant for ECS 193

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Overview of Product

Intended Audience:

This user guide is meant for developers and members of technical staff from the DSI Library department who intend to reproduce and/or build on top of this project. The user guide will cover development environment requirements, technologies used, and codebase logic that are the cornerstone of the Semantic Web application.

Background on Semantic Web:

Semantic web--an extension of the World Wide Web--was the original vision of having a structured open web. The semantic web provides a common framework that allows datasets to be shared and reused across different applications. The format used is RDF (Resource Description Framework). Starting ~6 years ago machine users outnumbered human users (e.g. web crawlers), but the data can't be exploited to its full potential because the open web is not formatted properly.

Motivation:

UC Davis is involved in a pilot program with Cornell University and Harvard College to be the first academic libraries in the world to change their library catalog to a linked-data universe by the end of the year.

Product Overview:

The web application is a proof of concept that shows how information from different linked-data sources are related to one another. We visualize this relationship using a D3.js graph. Users can use this web application to see the relationships between data found in the UC Davis Library catalog with other data found in external databases that have a SPARQL endpoints (ex: Wikipedia, Library of Congress, Online Computer Library Center).

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Technical Specifications

Programming Languages

Java - Primary backend language. Used to create and call query requests, parse data from the endpoints, and return the results in a unified triple format for the client-side to handle easily.

Python - Used for cleaning the JSON-LD UCD library catalog files in order to upload the library catalog data into Apache Jena Fuseki server.

D3.js version 4 - Data visualization framework used on client-side to help create the dynamically growing graph animation from aggregated data.

Javascript - Frontend coding language, used with D3 to allow client-side aggregation of the data that is being returned from the backend.

HTML, CSS - Frontend web development languages in order to set up the landing page as well as the page displaying the graph.

Databases

Apache Jena Fuseki - Apache Jena Fuseki is a SPARQL server with a triple-store database. It provides the SPARQL protocols for RDF query and storage systems. It provides a robust, transactional persistent storage layer. Jena stores linked-data and allows user to query, add, remove, manipulate, and publish data. Jena has a number of major subsystems with clearly defined interfaces between them. We use Jena to store the UC Davis library catalog data, which comes in the form of .jsonld files.

Reconciler - The Reconciliation API is based on the [MediaWiki](#) API. The Reconciler provides an authority record of URIs from other endpoints that are associated to the given URI. The Reconciler API is used to accumulate all possible endpoints the web app needs to hit in order to grow the D3.graph with more information.

Web Technologies

Apache Maven - Apache Maven is a software project management and comprehension tool. Maven provides an environment to add necessary libraries

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

to the Java code and allows the code to run as a standalone war file and has functions to deploy war file, to run Java project on a Tomcat server and to compile the Java package.

Jersey Framework - Jersey is an open-source framework that enables the development of RESTful Web Services that supports exposing data in a variety of representation media types.

D3.js - D3.js is a framework for the JavaScript Library that helps create dynamic and interactive data visualizations in a web browser.

Installation Steps

Below are the installation guidelines and dev environment required to run our project.

Development Environment Requirements:

1. **Java 8** - Need to install and make sure that you are running the project under Java 8 version. Older or newer Java versions will give warnings and break mvn.

Install a specific java version:

```
brew tap caskroom/versions
brew cask install java8
```

Switch b/w Java 8 and Java 9 if you have multiple versions:

```
cd ~/
open .bash_profile on terminal and add these lines
alias j9="export JAVA_HOME=`/usr/libexec/java_home -v 9`; java -version"
alias j8="export JAVA_HOME=`/usr/libexec/java_home -v 1.8`; java -version"
alias j7="export JAVA_HOME=`/usr/libexec/java_home -v 1.7`; java -version"
Source .bash_profile
Type j8 on terminal and it'll show that you switched to java 8
```

2. **Python 3** - UC Davis library catalog data files need to be cleaned before being uploaded into Apache Jena Fuseki. In order to run the python script that cleans the files, you need python 3. Make sure you have `brew` installed already.

Python install:

```
brew install python
```

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

3. **Apache Jena Fuseki** - SPARQL server we use to store the UCD library catalog dataset. We upload .jsonld files and use their sparql query API.

Installation Steps:

Go to <https://jena.apache.org/download/> and download
apache-jena-fuseki-3.7.0.zip

4. **Apache Maven** - The installation of Apache Maven is a simple process of extracting the archive and adding the bin folder with the mvn command to the PATH.

Installation Steps:

1. Ensure JAVA_HOME environment variable is set and points to your JDK installation.
2. Extract distribution archive in any directory
unzip apache-maven-3.5.3-bin.zip
or
tar xzvf apache-maven-3.5.3-bin.tar.gz

A [Project Object Model](#) or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. It contains default values for most projects. If any modifications are done to the pom.xml file, the dependencies need to be installed again by using the following command: `mvn install`

5. **Apache Tomcat 6** - Apache Tomcat 6 is required to run the project. The Apache Maven installation should included a tomcat plugin.
 6. **Unix OS** - Our project was developed and run on MacOS. We did not test whether this project runs on Windows.
 7. **Chrome or Firefox browser** - We tested our web link on Chrome and Firefox browsers.
-

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Development

To Clean UCD Library Catalog Files:

1. Create a folder and place all library `<filename>.jsonld` files that you want to sanitize and the `clean_data.py` script in the folder. The `clean_data.py` script can be found under `./swq_viz/data/` path of the project repo.
2. Run the command `python3 clean_data.py`. The script will go through all `.jsonld` files in directory where script is saved and output a new file `<filename>_fixed.jsonld` with the cleaned data.

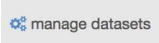
Information about existing cleaned UCD library files:

- strict_query file has 27,452 triples | 4.5MB
- loose_query file has 396,723 triples | 53.0MB

[Link to the fixed UCD library files](#)

To Start and Upload files to Apache Jena Fuseki:

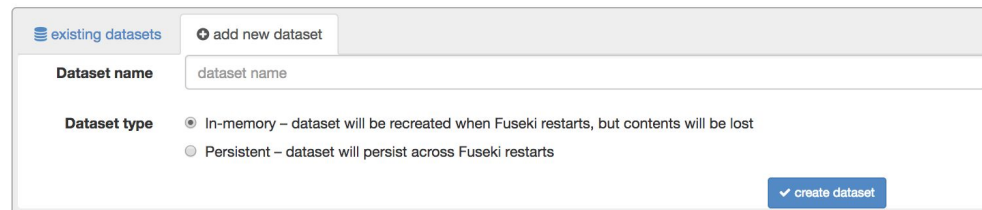
1. Open up terminal and go to the directory where you downloaded `apache-jena-fuseki-3.7.0.zip`.
2. Run the below commands to start Apache Jena Fuseki server:

```
Cd apache-jena-fuseki-3.7.0.zip
./fuseki-server
```
3. To upload files into Jena Fuseki server, do the following steps:
 - a. Go to browser and type `localhost:3030/` to use the Jena UI
 - b. Click on this tab  to create a new dataset and upload cleaned library files
 - c. Click on add new dataset. Make you dataset name ds this is the dataset that our project has hardcoded. So if you want a new dataset name, make sure to change it as well in the project repo. Set the dataset type to be persistent so that you don't have to keep re-uploading the files after

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

ending your fuseki session.



- d. Then click on upload data for the ds dataset and upload your `<filename>_fixed.jsonld` files. If there is an error, check the troubleshooting steps below in the user guide.

To Run Web App Project:

Please look at the [README.md](#) file in our GitHub repo to see how to run the project in more detail.

Features

Search - User can search for a specific term from a list given by VIAF database's autocomplete feature. When the search button is clicked, the page passes the VIAF ID of the term selected and finds the appropriate ID to use to query the UCD library dataset.

Start/Stop - User can click the start or stop button in order to start/stop the graph while it is dynamically growing. When the stop button is clicked, the graph will complete the current query and will not continue requerying until the start button is clicked.

Zoom - In order to zoom in or out of the window to see more or less of the d3 graph, use 2 fingers in a "in" or "out" fashion to zoom in and zoom out respectively.

Drag Graph - User can move the graph statically if it grows past the boundaries of the window. Click anywhere on the screen where the graph isn't taking up space.

Base graph triple form - If user wants json data of the UCD library triples from the inputted search term. Go to

`http://localhost:8080/TripleDataProcessor/webapi/myresource`

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Troubleshooting Steps

Apache Jena Fuseki

GC overhead limit exceeded

```
[2018-06-02 19:37:40] Fuseki WARN [5] RC = 500 : GC overhead limit exceeded  
java.lang.OutOfMemoryError: GC overhead limit exceeded
```

The Java runtime environment utilizes a built-in [Garbage Collection](#) process to help clear the available memory. However, you can get a GC overhead limit exceeded if your application uses up all the available memory and GC is not able to continuously clear it. A quick fix is open up the `./fuseki-server` script in a code editor and modify the line `JVM_ARGS=${JVM_ARGS:--Xmx1200M}` with `JVM_ARGS=${JVM_ARGS:--Xmx2048M}`. This increases the amount of space allocated to the application. If you still receive the same error, your file might be too large and you should try to load it into a TDB and use the [TDB database with Fuseki](#).

Console Logs

Both the server-side and client-side architectures create logs on the console.

The Java backend outputs the POST request IDs, responses from the endpoints as well as output of the parsers to the console. Any errors that occur with the Java back-end will be reflected here and this is a good way if you're trying to debug issues with new endpoints and new parsers. (More information about how to test endpoints is in the Testing section)

Similarly on the client-side, we have added `console.log` statement that will show how the nodes and links are being appended in order to create the graph. It also shows the requests that the client is making so that we can ensure the re-querying is working properly.

Uploading .jsonld files

If uploading `.jsonld` files to Apache Jena Fuseki is unsuccessful. There will be an error message stating line # where the error is happening, typically it is syntax related.

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Frequently Asked Questions

Q: What terms can I search?

A: Terms that exist in the VIAF database. An example is “Baklanov, Grigorij Âkovlevič 1923-2009”

Q: Why does the graph hang?

A: There are requeries happening on the backend, and large sets of triples are being aggregated and rendered on D3, so you may experience some latency. The status of the graph rendering is displayed on the web page so you can see if it is still in progress or if the graph rendering is done. API calls to external endpoints, like Reconciler’s DB, also takes a while.

Q: I installed the Apache Jena Fuseki server but its not working?

A: Please make sure you are running Apache Jena Fuseki as a server (./fuseki-server), if running this project locally on your machine. If you are planning to host this project on a proprietary server, there are other options to run Fuseki as a OS service or as a web service. If other questions or problems persist, please check out [Apache Fuseki documentation page](#).

Q: I created a dataset on Fuseki and uploaded files successfully, but the web app is not able to recognize the dataset.

A: Make sure your dataset name is **ds**, since this is what is hardcoded in the project. If you’d like to change the dataset name, make sure the new dataset name is reflected in the code as well.

Testing

Client-Side:

- Used browser’s developer tools console for debugging. We added console.log statements in JavaScript code.
- We tested functionality of the JavaScript aggregator.
- We tested that the correct endpoints were being re-queried

DSI Semantic Web App Visualization: User Guide

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

- We tested that nodes and links were being appended properly to construct the D3 graph.

Server Side:

- Debug with `system.out` or `console.log` statements to console
 - We also used Postman, which is an application that can be used to test server endpoints
 - Compile project (`mvn clean package`) and run server (`mvn tomcat:run`)
 - open up the Postman application and type in the URL for the server endpoint that you're trying to test
 - Specify whether you're testing a GET or POST request and make sure that the body of the POST request is matching that of the Consumes attribute of the Jersey Web Service endpoint.
-

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Appendix

Table of Contents

- I. [Table of Contents](#)
- II. [Introduction](#)
- III. [Glossary Terms](#)
- IV. [Choosing a Triple-Store Graph Database](#)
 - A. [Technology Survey](#)
 - B. [Option #1: Apache Jena Fuseki](#)
 - C. [Option #2: Neo4J](#)
 - D. [Option 3: Stardog \(Community\)](#)
 - E. [Option 4: AllegroGraph](#)
 - F. [Option 5: Ontotext-GraphDB](#)
 - G. [Technology Survey Results](#)
- V. [System Architecture Overview](#)
 - A. [Software Architecture](#)
 - B. [Web Application - FrontEnd UI](#)
- VI. [Requirements](#)
 - A. [Use Cases / User Stories](#)
- VII. [System Models](#)
 - A. [UML](#)
 - B. [Sequence Diagram - Java Backend](#)
 - C. [Data Flow Diagram](#)
- VIII. [Technology Employed](#)
 - A. [Programming Languages](#)
 - B. [Python](#)
 - C. [Databases](#)
 - D. [Web Application](#)
 - E. [Tools](#)
- IX. [Citation](#)
- X. [Revision History](#)
- XI. [Final Product](#)
 - A. [Homepage:](#)
<http://discover.library.ucdavis.edu/TripleDataProcessor/webapi/search>
 - B. [Temporarily Disabled URL:](#) <http://sparql.library.ucdavis.edu>
- XII. [GitHub Repository](#)
 - A. [Private Repo:](#) https://github.com/saileshpatnala/swq_viz

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Introduction

Background

Semantic web--an extension of the World Wide Web--was the original vision of having a structured open web. The semantic web provides a common framework that allows datasets to be shared and reused across different applications. The format used is RDF (Resource Description Framework). Starting ~6 years ago machine users outnumbered human users (e.g. web crawlers), but the data can't be exploited to its full potential because the open web is not formatted properly.

Currently, companies like Google and Apple (Siri) make use of the structured web (called knowledge cards) in their information architecture and use it to improve machine learning efforts. We are hoping to structuralize UC Davis library catalog into RDF format and make it available on the semantic web for other applications of the open web to use.

Motivation

UC Davis is involved in a pilot program with Cornell University and Harvard College to be the first academic libraries in the world to change their library catalog to a linked-data universe by end of the year.

Objective

- 1) Implement an n-triple store graph database using Apache Jena Fuseki to house library catalog data that comes in JSON-LD format.
- 2) Clean and structure the library catalog data into triples.
- 3) Build a web application for the user to interact with the data from library catalog and other n-triple store databases that have a SPARQL endpoint. We will be using D3.js framework to visualize the data similar to [BigDiva](#) and [RelFinder](#) .

End Product

Web application that visualizes relationships between input information and information using library dataset and the semantic web. Leveraging the UCD library dataset will not only give user ability to see the meta information regarding a topic but also query other sources on the web to return more substantive information.

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Glossary Terms

Term	Definition
semantic web	set of standards and best practices for sharing data and the semantics of that data over the web for use by application
semantics	meaning of data (ex: meaning of website URL shows where source is from and what it links to)
N-triple	<p>long list of semantic triplets</p> <p>The following N-Triples file consists of three RDF statements:</p> <pre><http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://purl.org/dc/elements/1.1/creator> "Dave Beckett" . <http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://purl.org/dc/elements/1.1/creator> "Art Barstow" . <http://www.w3.org/2001/sw/RDFCore/ntriples/> <http://purl.org/dc/elements/1.1/publisher> <http://www.w3.org/> .</pre> <p>which represents the following RDF/XML:</p> <pre><rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/"> <rdf:Description rdf:about="http://www.w3.org/2001/sw/RDFCore/ntriples/"> <dc:creator>Art Barstow</dc:creator> <dc:creator>Dave Beckett</dc:creator> <dc:publisher rdf:resource="http://www.w3.org/" /> </rdf:Description> </rdf:RDF></pre> <p>Triples are stored in a subject-predicate-object format. The Subject is what we are looking at, the predicate is the relationship between the subject and object, and the object is what we are looking for. Look at rdf:Description in the second picture, that is what is summarised in the triples in the first picture. The Subject may be a URI or a blank node, the predicate must be an URI, and the object can be a blank node, literal, or URI.</p>
SPARQL	Sparse query language = designed specifically to query linked data stores
SPARQL endpoint	Sparse query language = designed specifically to query linked data stores
RDF	enables users (human or other) to query a knowledge base via the SPARQL language. Results are typically returned in one or more machine-processable formats.
URI	Uniform Resource Identifier (URI) is a compact sequence of characters that identifies an abstract or physical resource.

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

OWL	a.k.a W3C Web Ontology Language. lets us store valuable bits of meaning so that we can get more out of our data.
ontology	relationship b/w entities (ex: A hasBoughtFrom B = B hasSoldto A)
Apache Jena Fuseki	Apache Jena Fuseki is a SPARQL server. It provides the SPARQL protocols for RDF query and storage systems.
D3js	JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS.
Java Servlet	Java program that extends capabilities of server, used for web application backend.
Semantic triplets	Subject-predicate-object data
literal	string
RelFinder.php	semantic web app that utilizes multiple SPARQL endpoints in the world. Ex: If you search "Kill bill", it shows as a string literal but grabs its URI during query process
BigDiva.org	similar to RelFinder. Uses product called D3 to show better graph visualization of relationships. Currently, has a small universe.

Choosing a Triple-Store Graph Database

Requirements:

- Has to have a SPARQL endpoint to query dataset
- Must allow for persistent data
- Must be scalable
- Supports RDF schema
- Can ingest data in RDF or JSON-LD form
- Allowed to be deployed as as a standalone server or web service

Research Citations:

- <https://db-engines.com/en/system/Jena:Neo4j>
- <https://programminghistorian.org/lessons/graph-databases-and-SPARQL>
- http://neo4j-org-dev.herokuapp.com/develop/linked_data
- <https://medium.com/@eisenzopf/graph-databases-linked-data-rdf-and-the-seman-tic-web-wasteland-69e9f4347a5b>
- <https://franz.com/agraph/allegrograph/>

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

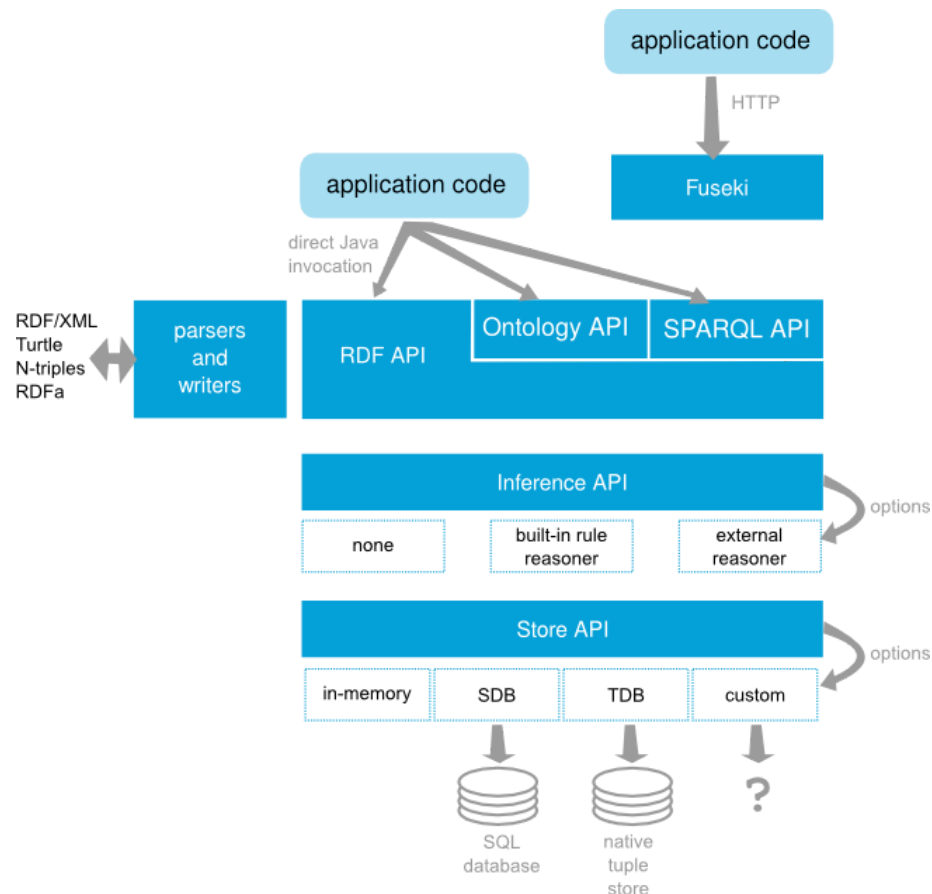
Technology Survey

Option #1: Apache Jena Fuseki

- **Background:** Apache Jena Fuseki is a SPARQL server. It provides a robust, transactional persistent storage layer.
- **Advantages:**
 - Can run as a Java web application (WAR file) as well as a standalone server
 - Easy to set up and monitor through its UI.
 - Secured with Apache Shiro
 - Provides a protocol engine for RDF query and storage systems.
 - Open Source project
 - Expose your triples as a SPARQL endpoint accessible over HTTP.
 - Fuseki provides REST-style interaction with your RDF data.
- **Architecture:**
 - At its core, Jena stores information as RDF triples in directed graphs, and allows your code to add, remove, manipulate, store and publish that information. Jena has a number of major subsystems with clearly defined interfaces between them

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala



-
- **Plugins:**
 - ARQ - is a query engine for Jena that supports the SPARQL RDF Query language. SPARQL is the query language developed by the W3C RDF Data Access Working Group.
 - Apache Maven - use manage all dependencies
 - Eclipse for java applications
- **APIs:**
 - RDF API - Interact with the core API to create and read RDF graphs. Serialise your triples using formats such as RDF/XML or Turtle
 - Inference API - Reason over your data to expand and check the content of your triple store.
 - Ontology API - Work with models, RDFS and the Web Ontology Language (OWL) to add extra semantics to your RDF data.
- **Scalability:** Very robust

Option #2: Neo4J

- **Background:** Graph database management system, #1 platform for connected data

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

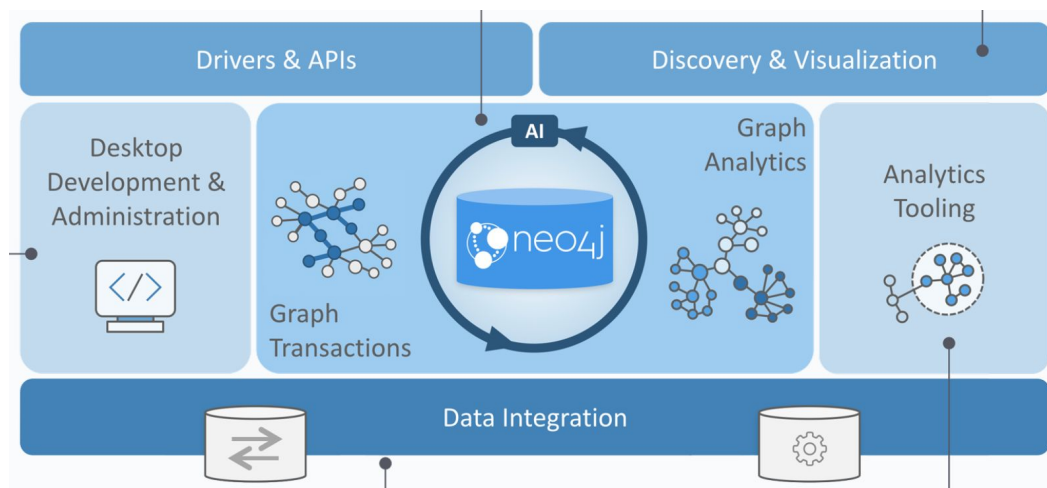
- **Advantages:**

- Graph database model can easily be adapted for linked data RDF model. This also adds support for RDF and SPARQL endpoints
- Supports data integration from Apache Spark, Hadoop.
 - Neo4J ETL can find relationships in relational databases and import into graph database
 - Supports fast data ingestion
 - Good for scaling since more data can be easily on-boarded to the local triple store
- Cypher can read from CSV files

- **Disadvantages:**

- Graph datastore inappropriate for transactional information like accounts
- Harder to do summing and maxing queries but counting queries are easier
- . Need to learn Cypher query language

- **Architecture:**



- **Plugins:**

- We're going to need the LinkedData plugin to support RDF and SparQL endpoints

- **APIs:**

- C#
- Java
- JavaScript
- Python

- **Scalability:**

- Offers a HIGH AVAILABILITY scalability package that offers
 - Increased read/write loads
 - Increased data size

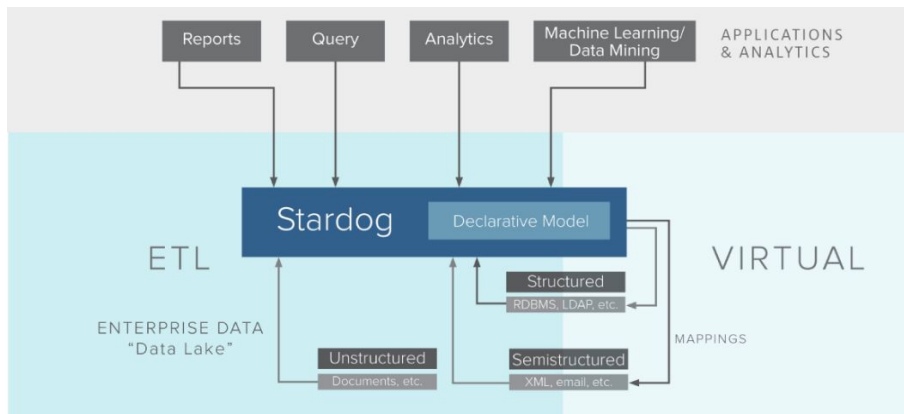
DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

- Load balancer
- Full data redundancy
- Service fault tolerance
- Can be easily scaled to use distributed server architecture

Option 3: Stardog (Community)

- **Background:** Knowledge Graph Platform. Need to email for license: <https://www.stardog.com/versions>
- **Advantages:**
 - Unifies all the data
 - speed to production
 - Reusable data model
 - Eliminates complexity when fusing machine learning, reasoning, and rules for contextualized insight of all the data
- **Disadvantages:**
 - 25M Nodes & Edges
 - 10K Axioms
 - 4 Users for Enterprise Graph Security & Authorization
- **Architecture:**



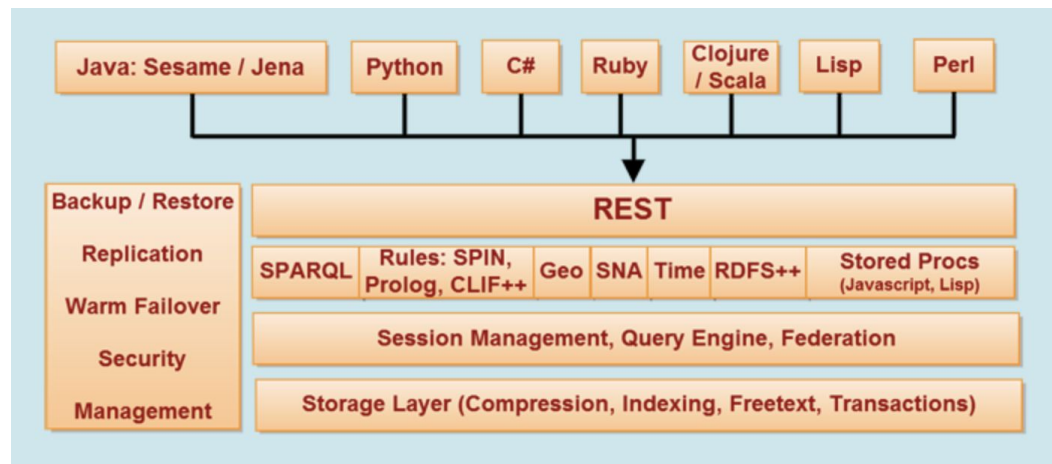
- **APIs:** https://www.stardog.com/docs/#_api_deprecation
 - Java
 - JavaScript
 - Clojure
 - .Net
 - Groovy
- **Scalability:** Can scale up to 50 Billion triples on a \$10k server, otherwise 25M for community

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Option 4: AllegroGraph

- **Background:** industry leading graph database, semantic graph technologies. Download from: <https://franz.com/agraph/downloads/?ui=new>
- **Advantages:**
 - Semantic Data Lake platform allows for rapid integration of new data
 - Supports deployment on Amazon AWS EC2
 - Gruff platform for visualization, navigation and querying graph datastore
 - Lot's of 3rd party integrations
 - Cloudera
 - MongoDB
 - Apache Solr
 - Anaconda
- **Disadvantages:**
 - Proprietary graph visualization. Don't think we will be able to connect it to D3.js
- **Architecture:**



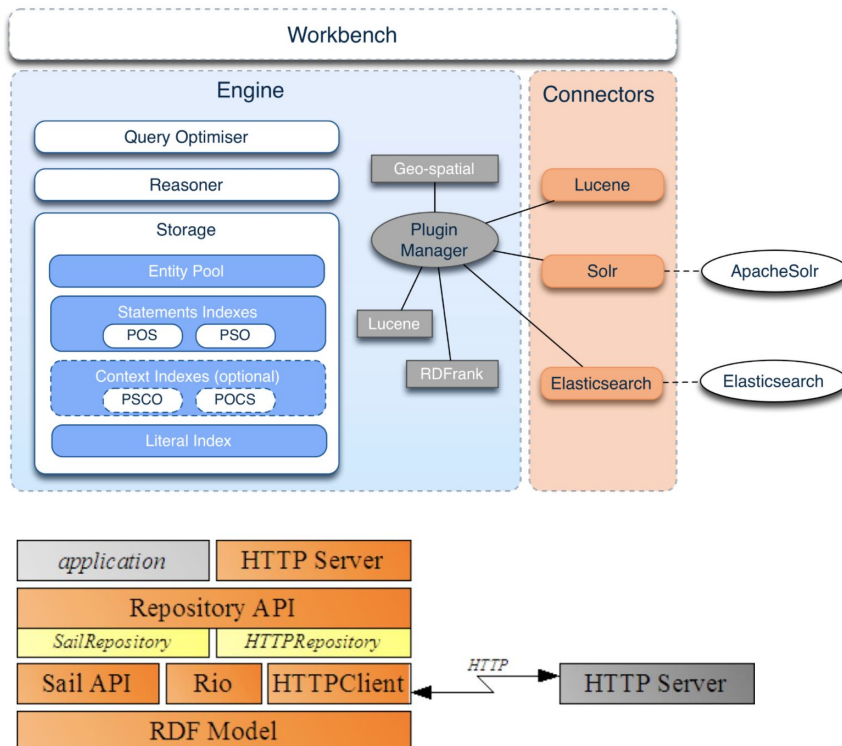
- **Plugins:**
 - Amazon AWS EC2
 - MongoDB data integration
 - Apache Solr integration
- **APIs:**
 - REST layer API offers support for Java, Python, Lisp, Scala, Ruby, Perl. C#
- **Scalability:**
 - Can connect to AWS so definitely a scalable option
 - Set new record in 2011 - 1 trillion

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Option 5: Ontotext-GraphDB

- **Background:** GraphDB is packaged as a Storage and Inference Layer (SAIL) for RDF4J and makes extensive use of the features and infrastructure of RDF4J, especially the RDF model, RDF parsers and query engines. Download: <https://ontotext.com/>
- **Advantages:**
 - No constraints on volume of loaded data
 - Good starting point for smart data proof-of-concepts
 - Supports all RDF serialisation formats
 - Community help support
 - Compatible with Jena with a built-in adapter
 - Import/export of RDF syntaxes through RDF4J: XML, N3, N-Triples, N-Quads, Turtle, TriG, TriX;
- **Disadvantages:**
 - Moderate query loads (no more than 2 queries in parallel) for free version.
- **Architecture:** Implements RDF4J framework interface. One of few triple stores that can perform semantic inferencing/ Free version: scales to 10s of billions of RDF statements on single server w/ limit of 2 concurrent queries. Workbench is the GraphDB web-based administration tool. Performs query and reasoning operations using file-based indice



DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

- **APIs:**
 - SAIL API = used to be integrated with rest of RDF4J framework (ex: query engine and web UI)
 - RDF4J framework = framework for storing, querying, and reasoning with RDF data.
 - Can be used as either a standalone server or embedded in an application as a java library.
 - Supports SPARQL query language and most popular RDF file formats
- **Scalability:**
 - Free version used for proof of concept as of now, but can manage 10's of billions of RDF statements on a single server

Technology Survey Results

We researched 5 n-triple store graph databases and looked at the following metrics: API integration, plugins, scalability, cost, and deployment options. We decided on Apache Jena Fuseki for the following reasons:

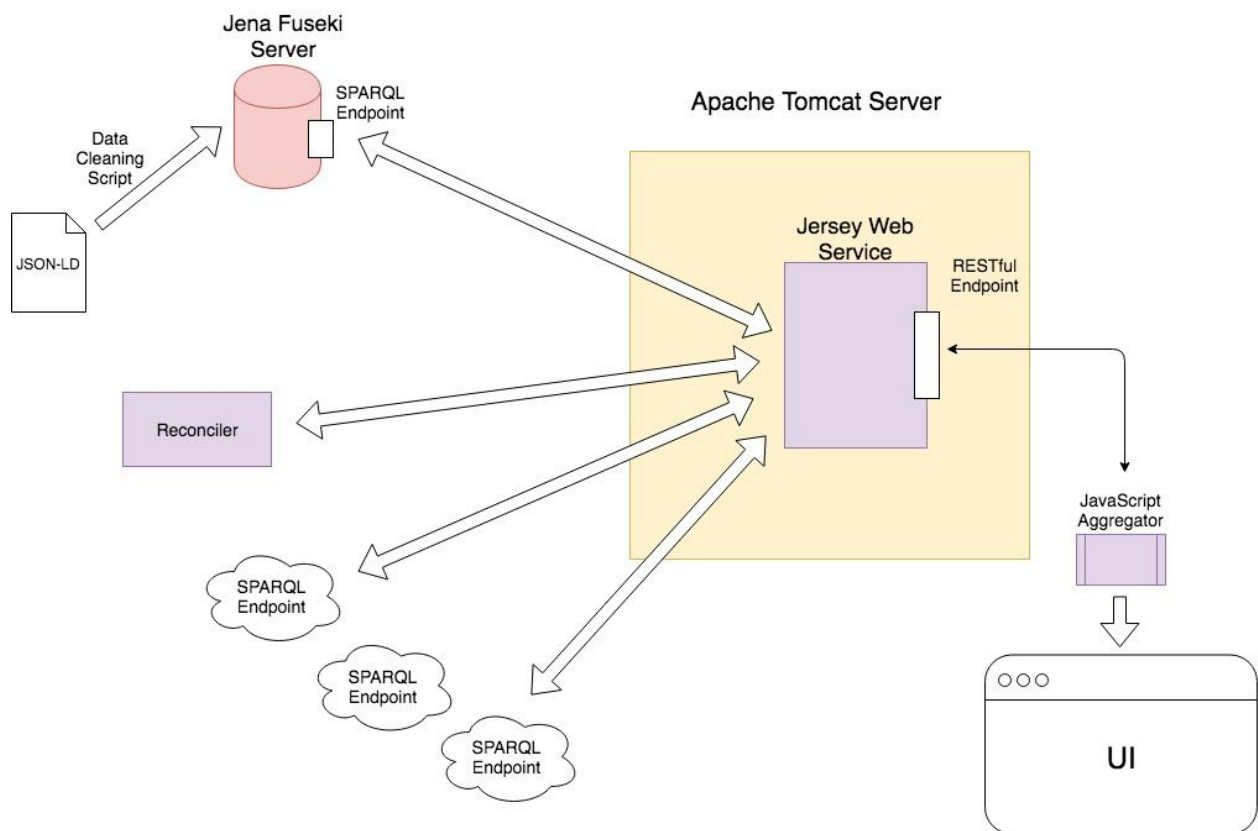
- 1) easy to use interface and setup
- 2) easy integration with its built-in API in java that is compatible with our system architecture
- 3) Scalable and free
- 4) Ability to operate as a standalone server or a web service
- 5) Persistent dataset memory

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

System Architecture Overview

Software Architecture



There are 3 main components to our system architecture:

1. Jena Fuseki Server

The Jena Fuseki server is the library data store that holds the entire library catalog in a triple format. The client gave us the library catalog in a linked data format known as JSON-LD which was cleaned using a script and then uploaded into the Fuseki triple store. The Apache Jena Fuseki server offers a SPARQL endpoint which we use to query the data in the triple data store. The Jena Fuseki server is hosted on the library server with persistent library catalog data, and thus able to configure, access, and upload data onto the production Jena Fuseki server.

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

2. **Jersey Web Service**

The Jersey web service serves as the backend for our web application. It performs several functions including: rendering the HTML pages, performing queries on the library triple data-store as well as the third-party triple stores using SPARQL endpoints, as well as maintain proper data flow between the client. The web server first pings the library catalog for the searched data and further pings the other SPARQL endpoints to substantiate on the information in order to build the knowledge graph. When working in a world of URIs, we need the Reconciler which is a proprietary product created by the company OCLC in collaboration with UC Davis and some other organizations researching the semantic web. The reconciler offers us other matching URIs that are being used by other data stores to refer to the same term. We use the reconciler to get the additional URIs and the Jersey web service queries the list of URIs given by the reconciler to get more triples before sending the data to the client-side via the JavaScript aggregator. The client requested Tomcat server to host our Jersey web server. Therefore, the Jersey web service is running on Tomcat alongside the Jena Fuseki server in the Library server.

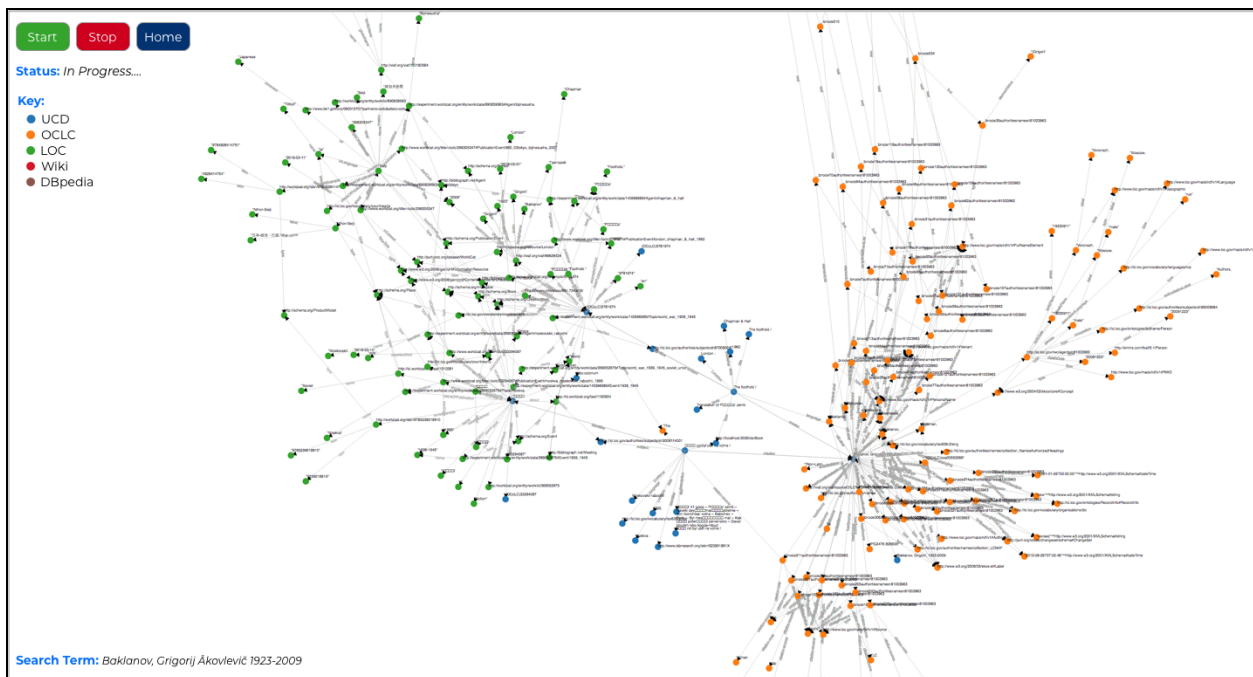
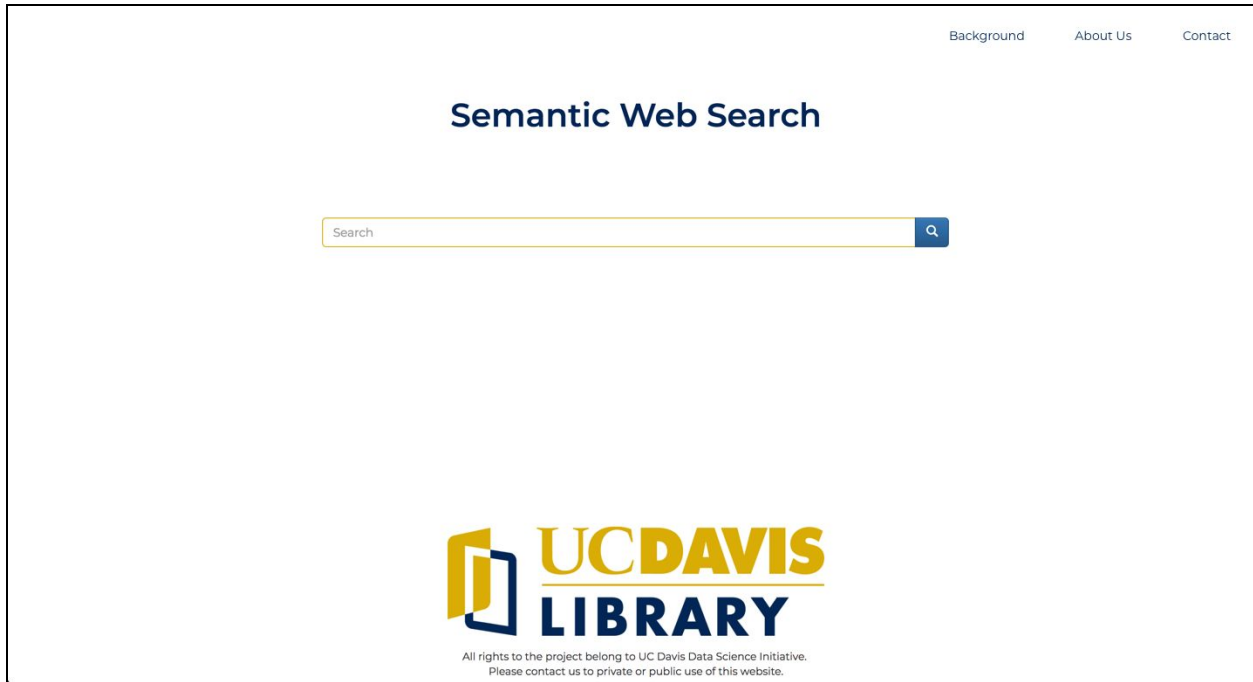
3. **JavaScript Aggregator**

Since we are trying to emulate a [RelFinder](#) like front-end, we decided to have client side aggregation for the data returned from the Jersey web service. The JSON that is returned from the web server is parsed in JavaScript, duplicates are removed and converted into a format that the D3.js can render. In addition to this, we're filtering the returned JSON for URIs and sending them back to the server to re-query until we find the string literal instead of a URI. Besides the aggregator, the client-side code also contains implementations of the VIAF autocomplete feature so the user can pick a term that exists in the VIAF database based on their continuing input. We then use GET and POST requests to render the D3 graph based on the search input and the options selected, if any.

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Web Application - FrontEnd UI



DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Requirements

Use Cases / User Stories

Use Case	Description	Metric - Acceptance Test
#1	Search for a topic input and see related information graph grow/evolve coming from UC Davis Library catalog.	<i>INPUT</i> feature: The user will be able to see a dynamic graph with new subject nodes and their relationships to the main subject term coming from the UC Davis Library Catalog dataset.
#2	Search for a topic input and see related information graph grow/evolve coming from other SPARQL endpoints like wikidata.	<i>INPUT</i> feature: The user will be able to see a dynamic graph with new subject nodes and their relationships to the main subject term coming from external databases such as WikiData.
#3	User can select which endpoints to query in the building of the information graph.	<i>SOURCE FILTER</i> feature: The user will be able to specify one or more of the available sources the web app is using to query information from.
#4	User can checkbox option to return data in raw JSON data form or in D3.js graph form.	<i>DISPLAY OPTIONS</i> feature: The user can have the results of their search return in either a graph (default) or as JSON text.
#5	Checkbox option to download data into CSV file.	<i>DOWNLOAD CSV</i> feature: The user can choose to download all the triples returned from the search result into a CSV file for external use.
#6	Stop building of the evolving graph.	<i>STOP</i> feature: The user can stop the ongoing build of the graph from the search result if they want.
#7	Pause/Continue building of the evolving graph.	<i>PAUSE/CONTINUE</i> feature: The user can pause the ongoing build of the graph from the search result, and can later click continue if they want the

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

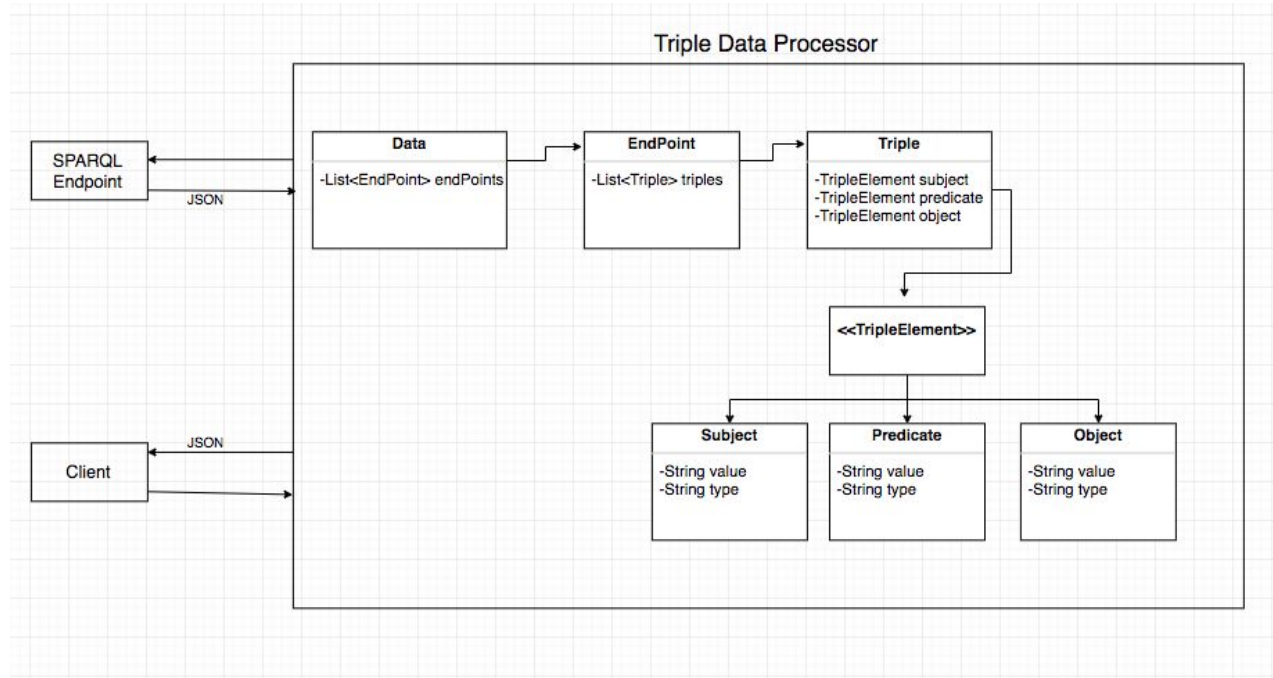
		graph to continue growing from where they last paused.
#8	Checkbox option to display the VIAF ID of chosen search term.	<i>SHOW VIAF ID</i> feature: The user can have the VIAF ID of the chosen search term displayed on the rendered D3 graph page.
#9	Checkbox option to select the growth depth of D3.js graph from search term. Either show all related information of input search term or information that only directly contains input search term.	<i>SEARCH DEPTH</i> feature: The user can choose to have their rendered D3 graph grow continuously and display all related subject nodes and their relationship to main subject term (MAX) or have their rendered D3 graph display subject nodes that only contain the input search term (MIN).
#10	The home button allows user to return to the search screen and stops build of any active graphs.	<i>HOME</i> feature: The user will be able to see the web page return the home screen where they can see the main search fields empty again with search options.

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

System Models

UML

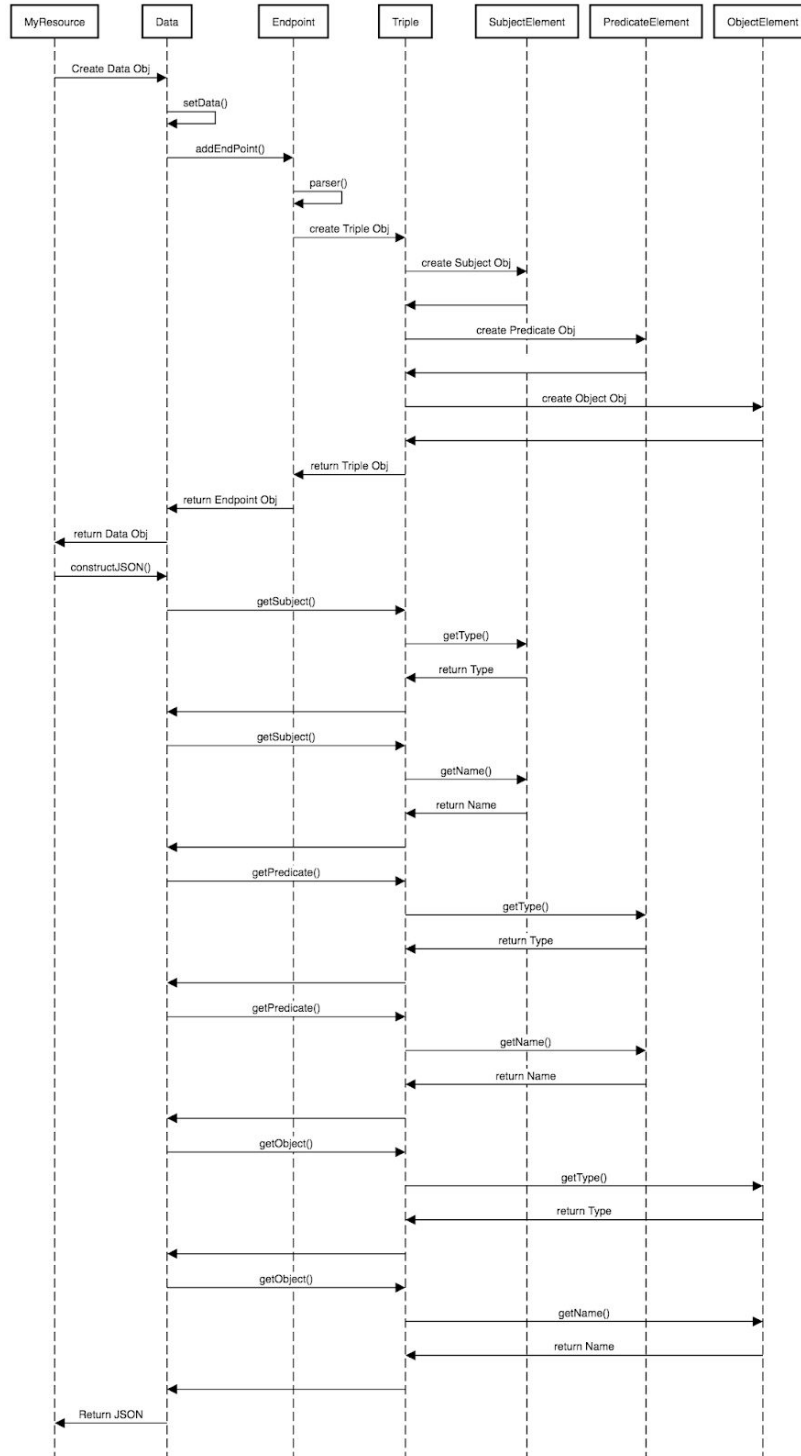


In the backend, there included an unique java class for each endpoint query. There is also a Helper.java class that generated the query url based on the id provided. After the endpoints are queried, they are parsed and loaded to the Data class (outlined above). Finally, the the Data class will construct a formatted JSON that will be returned back to the client side.

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

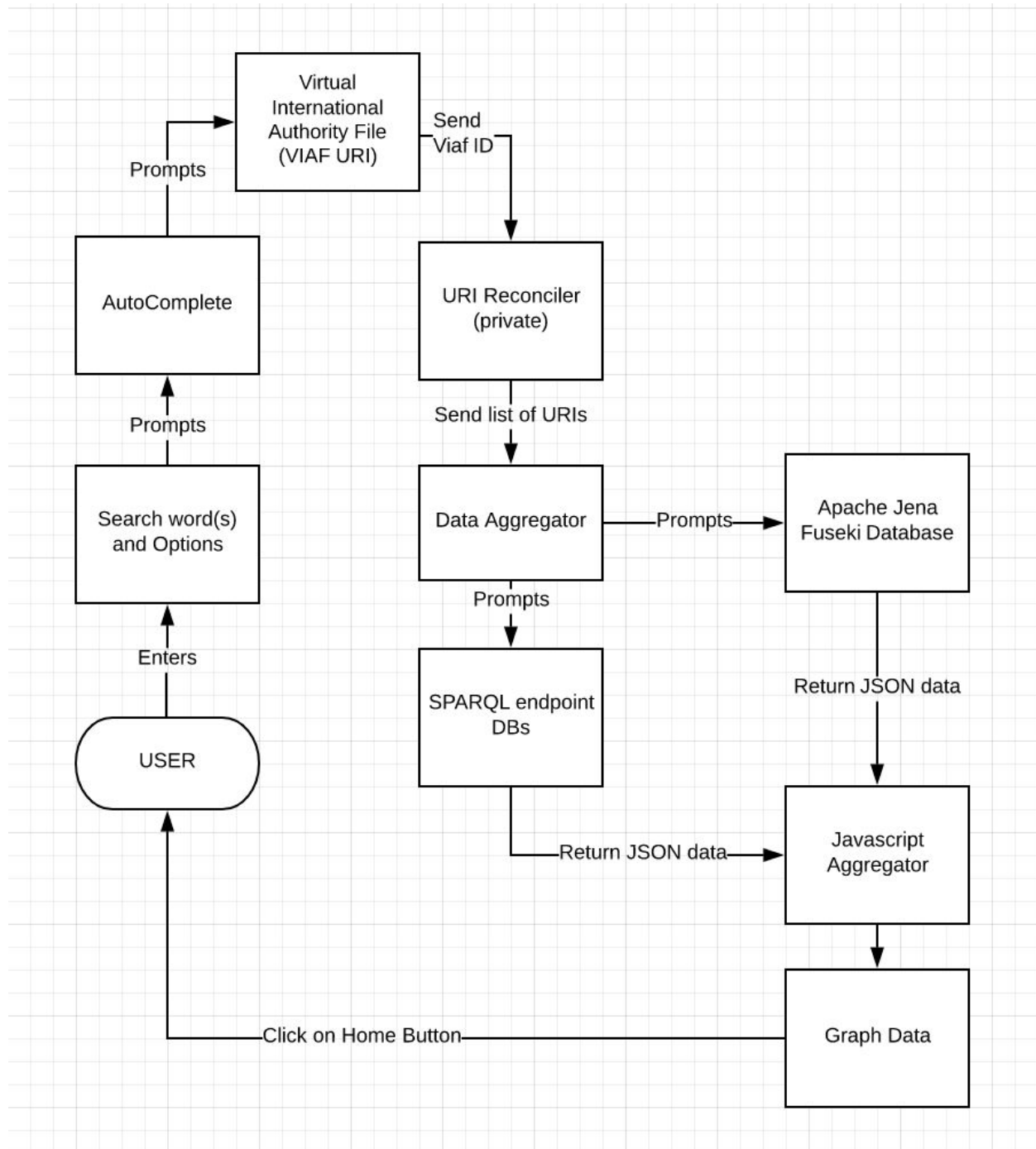
Sequence Diagram - Java Backend



DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Data Flow Diagram



DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Technology Employed

Programming Languages

- **Java**
Java programs are compiled independently of the platform because it utilizes bytecode language. This allows Java programs to run on any machine that has the Java Virtual Machine installed. Java also makes including necessary external libraries for our project and communicating HTTP requests easy because of its ability to work with web applications. This also works well with our choice of using Apache Maven to build and compile code quickly and deploy it to the Apache Tomcat Server.
- **JavaScript**
JavaScript is one of the three core languages/technologies used in creating an interactive website and avoids the necessity of the java plug-in being installed to view the page. Since JavaScript was initially created as a complementary scripting language to go with Java and is widely used to build websites, especially with the new languages of NodeJS, ReactJS, AngularJS, and other frameworks, we decided to choose JavaScript as our other primary programming languages.
- **Python**
Python script used to clean JSON-LD file of library catalog data to be ingested into Apache Jena Fuseki server. Run the script locally and it will output a new fixed file <original_name>_fixed.jsonld that you can upload manually to Apache Jena Fuseki.

Databases

- **Apache Jena Fuseki**
Information about Apache Jena Fuseki as well as our reasoning for choosing it can be seen in section [IV.B](#).
- **Reconciler**
The Reconciliation API is based on the [MediaWiki](#) API and the index of Passage entities. The Reconciler provides alternative URIs associated with other endpoints for a given URI. This API is used to accumulate all possible triple data relation for the searched literal.

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Web Application

- **Apache Tomcat Server**

The Apache Tomcat software is an open-source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. The client had a pre-existing tomcat server where he was hosting other projects and since we decided to choose a Java server implementation this was an obvious choice for us. The client provided us with access to the server with authentication using our kerberos username and password and we will be installing the necessary software on there and hosting our application from there.

- **Jersey Framework**

Jersey is an open-source framework that enables the development of RESTful Web Services that supports exposing data in a variety of representation media types. It abstracts away the low-level details of the client-server communication. In order to simplify development of RESTful Web services and their clients in Java, a standard and portable [JAX-RS API](#) has been designed. We decided to work with the Jersey framework over Java Servlet since Jersey is one abstraction away since it is built on top of Java Servlet and offers a simpler and more intuitive API to work with. There are a lot of things that are offered out of the box when accessing POJOs including serialization and deserialization of JSON objects. In addition, the framework offers easy consumption of data by JavaScript clients especially since we're going to be using jQuery.

- **D3js**

D3.js is a framework for the JavaScript Library that helps create dynamic and interactive data visualizations in a web browser. We decided on D3.js because it is one of the most popular libraries for visualization and we wanted to maintain consistency in language and usability. Our goal is to implement the functionality of Relfinder with the aesthetics of BigDiva using D3.js

Tools

- **Asana**

Asana is an application that helps teams break down and assign tasks to team members. It is a simplified version of Atlassian Jira.

- **Slack**

Communication channel with client and team members. Integrate google drive and github plugins.

- **GitHub**

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Used for revision tracking and source code sharing amongst team and client

Citation

VIAF Autocomplete:

- <https://github.com/oclc-developer-house/jquery-viaf-autocomplete>

Apache Jena Fuseki:

- <https://jena.apache.org/documentation/fuseki2/>

D3.js

- <https://github.com/d3/d3/wiki/Gallery>

Reconcilor

- http://18.218.102.193/wiki/Main_Page
- http://www.projectpassage.org/wiki/Main_Page

D3.js Zoom Feature

- <https://stackoverflow.com/q/41482473>

D3.js Label Text Wrap

- <https://stackoverflow.com/q/24784302>
- <https://bl.ocks.org/mbostock/7555321>

Revision History

Revision #	Description	Initials	Date
1.0	Initial Requirements Document	authors	2/10/18
2.0	Final Requirements Document	authors	4/7/18
3.0	Updated Document after Beta Release	authors	6/2/18

DSI Semantic Web App Visualization: Requirement Document

Nishant Chandrashekar | Wonhee Park | Mithun Vijayasekar | Sailesh Patnala

Final Product

The final product will be available on localhost after setting up project. The project should run on port 8080 by default. The homepage will be located on: <http://localhost:8080/TripleDataProcessor/webapi/search>

Homepage: ** Holds older version of product

<http://discover.library.ucdavis.edu/TripleDataProcessor/webapi/search>

Temporarily Disabled URL: <http://sparql.library.ucdavis.edu>

**Due to querying problems through tomcat server, url is currently rerouted to homepage

GitHub Repository

Private Repo: https://github.com/saileshpatnala/swq_viz

** Please email sgpatnala@ucdavis.edu to gain access to repository. Proprietary to UC Davis Library DSI director: Carl Stahmer.

Software:

- Apache Jena Fuseki 3.6.0 (Database)
- Apache Tomcat (Server)
- Pulse Secure ([VPN Access to UC Davis web server](#))
- SPARQL Querying Language
- Programming Language: Java, Javascript, D3.js, HTML, CSS, and Python