Visual Query System (VQS)

System Documentation

Contents

[1. Introduction 4](#_Toc504881064)

[2. VQS Code 4](#_Toc504881065)

[3. Project Setup 4](#_Toc504881066)

[3.1. Prerequisites 4](#_Toc504881067)

[3.2. Import Project 5](#_Toc504881068)

[4. Building and Running Application 5](#_Toc504881069)

[5. System Maintenance 6](#_Toc504881070)

[5.1. Server and Rest Services 6](#_Toc504881071)

[5.2. Backend Services 7](#_Toc504881072)

[5.3. POM – Dependencies 7](#_Toc504881073)

[6. Packaging and Deploying 7](#_Toc504881074)

[7. Appendix 9](#_Toc504881075)

[7.1. Command Line Install 9](#_Toc504881076)

[7.2. Eclipse Import 10](#_Toc504881077)

[7.3. Packaging and deploying 14](#_Toc504881078)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Name** | **Description** |
| 1 | 01/24/18 | Shahila Retnadhas | Initial Document |

# Introduction

This project is a standalone JAVA application that serves the Visual Query System (VQS) and also publish a REST web service, using Jersey and Embedded Jetty server. JAX-RS (JAVA API for REST Web Services) is used for the VQS Interface. The entire application is packed using Maven as a single JAR executable file that can be started from any machine with Java environment.

VQS is a visual query formulation tool for expressing information needs in terms of queries over ontologies. VQS is composed of an interface and a navigation graph extracted from the underlying ontologies. The interface components are populated and driven according to the information in the navigation graph.

# VQS Code

All code for the VQS is found in the Github repository located at:

<https://github.uio.no/ernestoj/OptiqueVQS>

Github UiO account is accessed with the UiO username and password. Request the owner of the code to get access to the repository.

# Project Setup

# Prerequisites

Install JDK 8 and Maven 3.5. If you plan to work with Eclipse, install Maven and Github plugins.

Installing Maven for Windows

* Download Maven from [Apache Website](https://maven.apache.org/download.cgi) and configure the necessary environment variables.
* Unzip the file to the folder of your choice.
* Add the bin directory (e.g.: *C:\Program Files\apache-maven-3.5.2\bin*) to the PATH variable.
* Add environment variables MAVEN\_HOME, M2\_HOME and point to the maven directory.

Installing Maven Eclipse Plugin

* Check if Maven Plugin M2Eclipse is already installed.
* If not, install the plugin, *Help-> Install New Software*.
* Search for M2Eclipse and click *OK*.
* Plugin will be installed.

# Import Project

Import the project in Eclipse or command line to run and debug the application.

Command Line

* Download the code with git:

git clone git://github.uio.no/ ernestoj/OptiqueVQS

Eclipse Import

* Open Eclipse with the workspace you want to use.
* *File -> Import*.
* Select *Git->Projects* from Git and Click *Next*.
* Select *CloneURI*.
* Enter the following details and Click *Next*.

URI : <https://github.uio.no/shahilar/OptiqueVQS>

User: UioUsername

Password:UioPassword

* Select Master Branch and Click *Next*.
* Select the local directory destination and Click *Next.*
* Select “Import existing Eclipse Project” and Click *Next and Finish.*

Screenshots can be found [here](#_Eclipse_Import).

# Building and Running Application

Command Line

* Build and package the web application

cd OptiqueVQS mvn clean install

* Start the web application

mvn exec:java@VQSStart -Dexec.args="5080 5443 5090"

* The server is up and the application is running in the given ports. Browse the page using the URL, The HTTP URL

<http://localhost:5080>

* The HTTPS URL

<https://localhost:5443>

* Stopped by executing the maven command in another terminal.

mvn exec:java@VQSStop -Dexec.args="5090"

* If no port is given in any of the command above, the application will be running in the default ports: 8080-http port; 8443-https port; 8090-stop port;

Screenshots can be found [here](#_Command_Line_Install).

Eclipse Build

* Build the project and start the “eu.optiquevqs.server.JettyStart” class to start the application.
* The application is now running and can be accessed in the localhost.

# System Maintenance

# Server and Rest Services

The VQS application is deployed with the Jetty Webserver. The Jetty Webserver is embedded within the application. The paths within the application and the methods for the rest services are handled by using the JAX-RS provider classes.

Below is the explanation of the different java packages and classes

* **eu.optiquevqs.server**

The server and their configurations is defined in this package/

* + *JettyStart***:** The jetty server is instantiated and runs the jetty server in the given port. The default http port and https ports are used if it is not passed as a parameter. A basic level authentication is established that can be used to control the access of the different resources. The VQS application is configured to run in the root context.
  + *Monitor*: This class listens for the stop command and causes the jetty server to stop the server instance.
  + *Stop*: This issues the stop command so that the jetty server is stopped.
* **eu.optiquevqs.api.rest.resources**

The REST Services are established in the provider/Resource class used by JAX-RS.

* + *QFOntologyAccess*: This is one of the JAX-RS provider class that handles the path “REST/JSON/QFOntologyAccess” and the different methods required for accessing the Ontology by VQS and establishes the REST service at the path.
  + *QFQueryCatalogAccess*: This JAX-RS provider class handles the path “REST/JSON/QFQueryCatalogAccess” and the different methods related to query processing VQS and establishes the REST service at the path.
  + *SparqlEndpoint*: This JAX-RS provider class establishes the REST service for executing SPARQL query in the given SPARQL Endpoint.
* **eu.optiquevqs.api.rest.resources.impl**

The implementation of the rest resource methods defined in the previous package are done in this package.

# Backend Services

The Backend logic related to the ontology reasoning is defined in the “Ontology-service-toolkit” maven project. The toolkit is added as a dependency to this project and the methods are accessed in the implementation methods of the rest resources. Documentation for the same can be accessed here.

# POM – Dependencies

The necessary dependencies are added to the Pom.xml file. The dependencies added are Jersey, Jetty, owlapi, json, jaxrs api, ontology service toolkit and more. In case, we need to add a 3rd party jar file as a dependency, it can be done by executing the following maven command;

mvn install:install-file -Dfile=<path-to-file> -DgroupId=<group-id>

-DartifactId=<artifact-id> -Dversion=<version> -Dpackaging=<packaging>

For example, to add RDFox lib to local maven repository, the command used is

mvn install:install-file -Dfile= E:\UiO\RDFox\JRDFox.jar -DgroupId=uk.oxford

-DartifactId=JRDFox -Dversion=2776 -Dpackaging=jar

# Packaging and Deploying

The application can be packaged either as a single fat executable jar file with all the dependencies or simple jar file with dependencies in separate folder. It can then be executed in any java machines. The Maven POM file handles building the JAR file by making use of the maven jar plugin.

CommandLine:

* Package the application by running the following command

mvn clean install

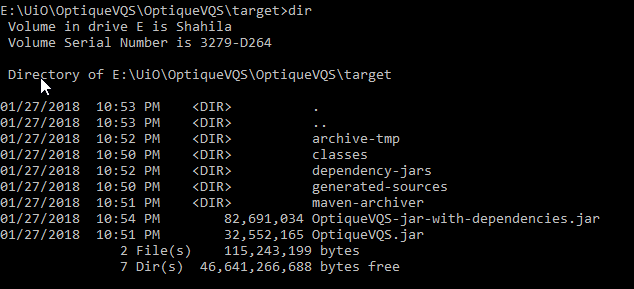
Eclipse:

* Run the Pom.xml file as Maven clean and then Maven Install.

Lean Jar File execution:

The lean jar file execution is same as Fat jar file execution. The only difference is that it needs to be imported along with their dependency folder. This is a better way than creating the fat file as there is a possibility of some files being overwritten.

Fat Jar File execution:

The POM file is configured such that two JAR files will be created in the target directory. One with all the dependencies included and the other with path to the dependency folder. 

To deploy the application in a new system, execute the jar with all dependencies;

java -jar OptiqueVQS-jar-with-dependencies.jar

The application will be now running and can be accessed in the URLs http:localhost:8080 or https:localhost:8443. To change the port number, pass the port numbers as parameters;

java -jar OptiqueVQS-jar-with-dependencies.jar 5080 5445 5090

To stop the application execute the following command in another terminal.

java -cp OptiqueVQS-jar-with-dependencies.jar eu.optiquevqs.server.Stop

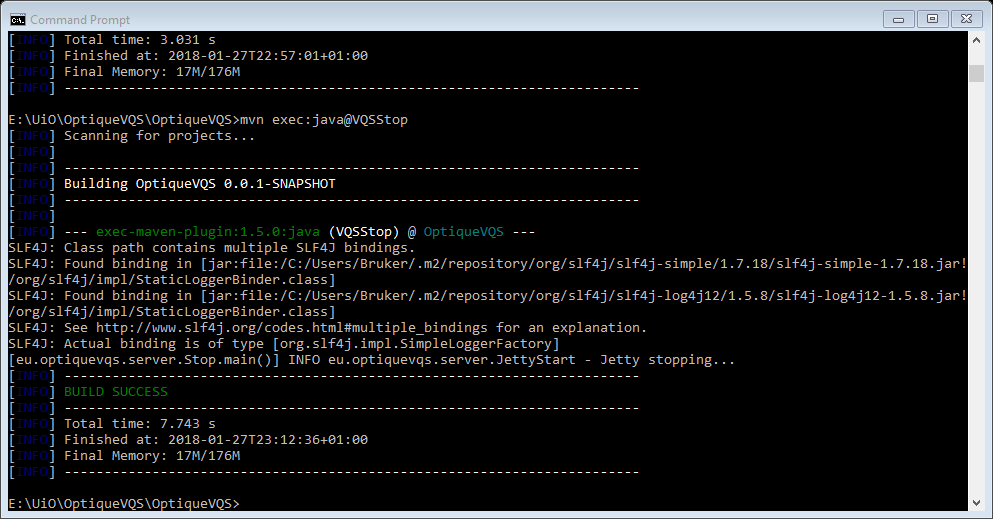
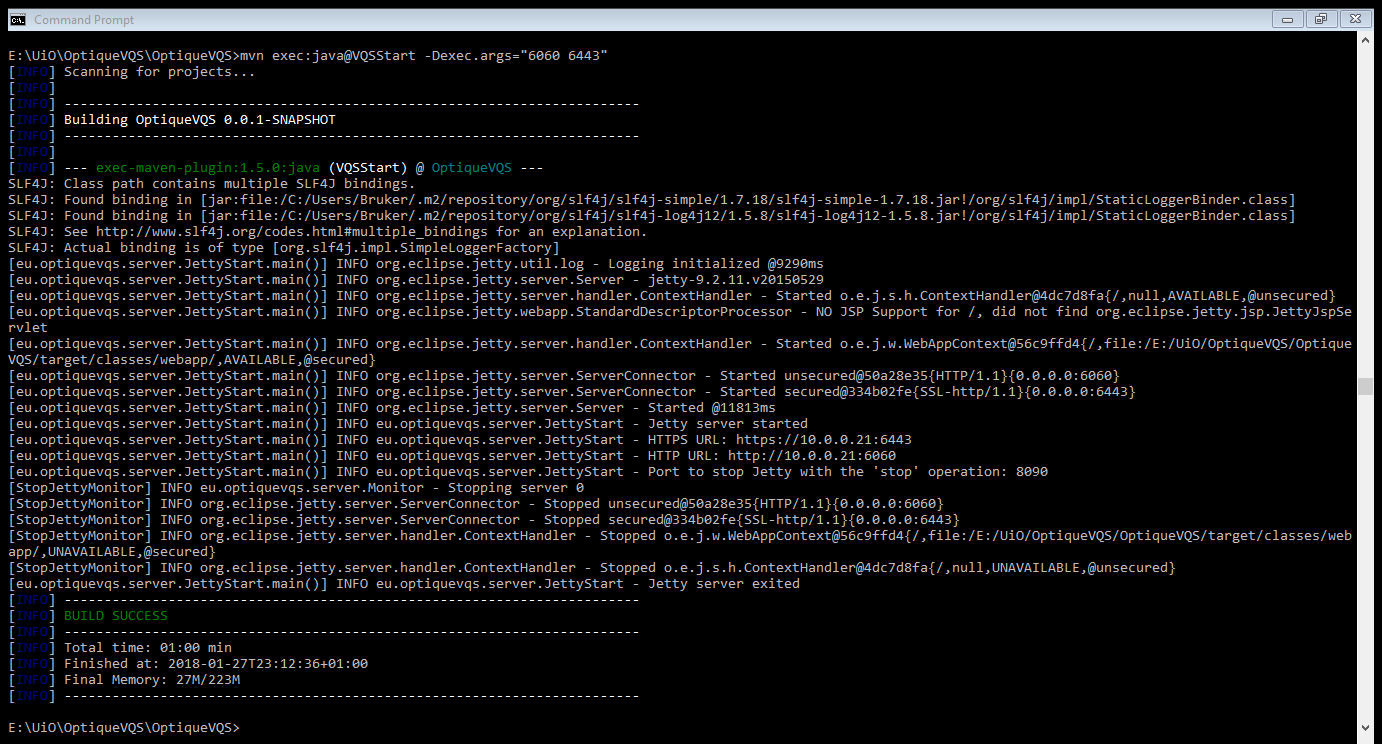
In case, a different stop port is passed when running the application the same port needs to be given as an argument to stop the server and free the port.

java -cp OptiqueVQS-jar-with-dependencies.jar eu.optiquevqs.server.Stop 5090

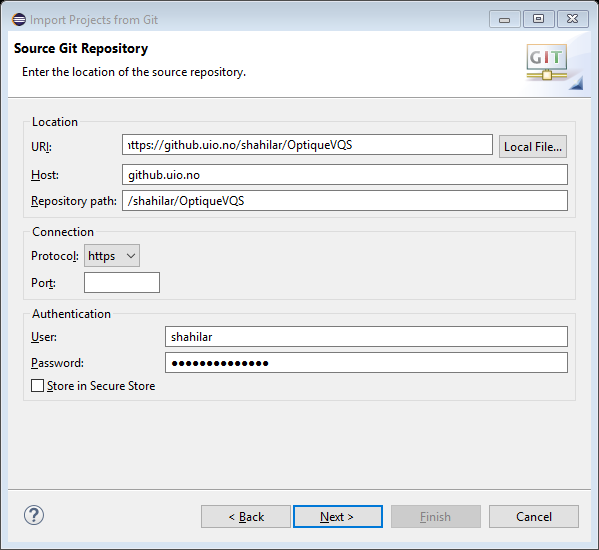
Screenshots are available [here](#_Packaging_and_deploying).

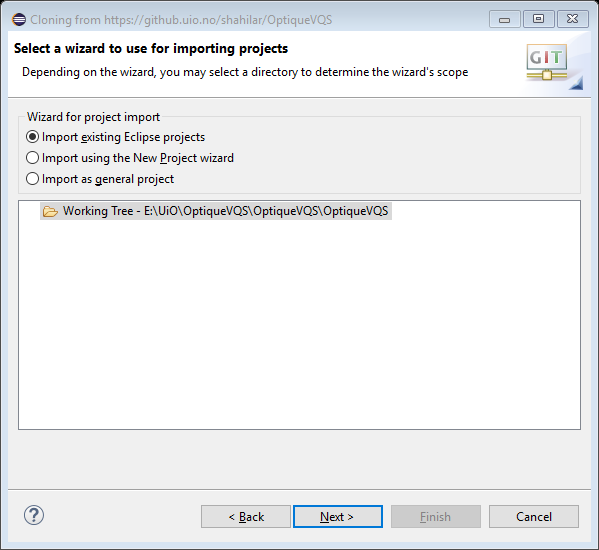
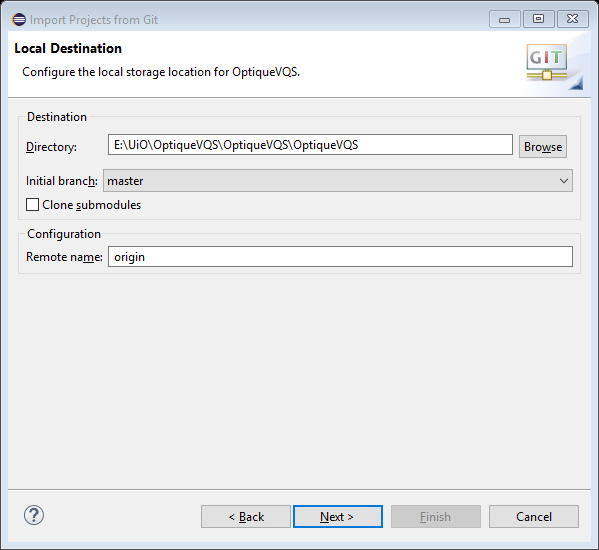
# Appendix

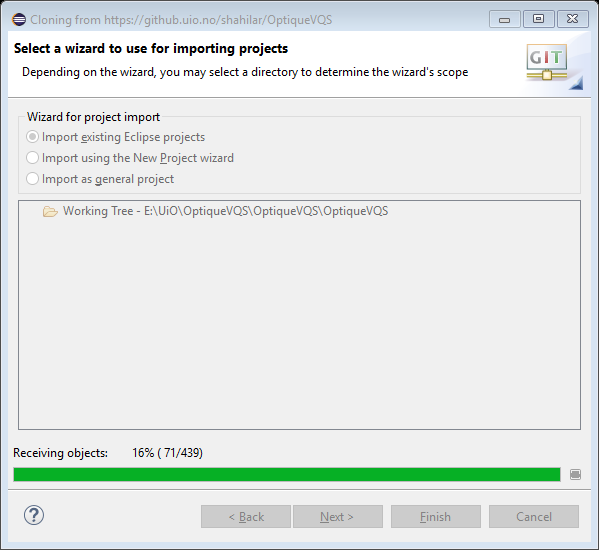
# Command Line Install

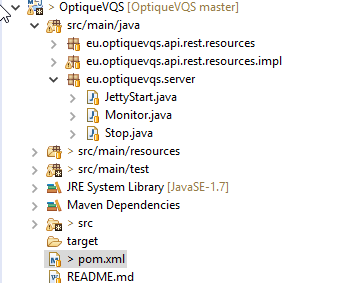


# Eclipse Import









# Packaging and deploying

