Development Manual

Table of Contents

[Change Log 2](#_Toc29375658)

[Environment Setting 2](#_Toc29375659)

[Java version 2](#_Toc29375660)

[Tomcat version 2](#_Toc29375661)

[Maven version 2](#_Toc29375662)

[Neo4j version 3](#_Toc29375663)

[Main Github CodeBase 3](#_Toc29375664)

[Backend codebase 3](#_Toc29375665)

[Frontend codebase 3](#_Toc29375666)

[Github Data Model 3](#_Toc29375667)

[SumoCollector 19.253-6 3](#_Toc29375668)

[newrelic-infra 1.8.19-1 3](#_Toc29375669)

[Dev 3](#_Toc29375670)

[QA 4](#_Toc29375671)

[Code Structure 4](#_Toc29375672)

[Folder Structure 4](#_Toc29375673)

[NameSpace 5](#_Toc29375674)

[General Overview and Design Guidelines 5](#_Toc29375675)

[System Design 5](#_Toc29375676)

[Backend 6](#_Toc29375677)

[Configuration 7](#_Toc29375678)

[Connect to Neo4j 8](#_Toc29375679)

[ERROR Handler 8](#_Toc29375680)

[Authorization (AOP) 9](#_Toc29375681)

[Logger Log4j 9](#_Toc29375682)

[Testing 9](#_Toc29375683)

[Frontend 10](#_Toc29375684)

[Frontend Backend integration 10](#_Toc29375685)

[APIs 11](#_Toc29375686)

[Prepare for testing 11](#_Toc29375687)

[GraphiQL’s Instruction 11](#_Toc29375688)

[Steps for testing 12](#_Toc29375689)

[POSTMAN’s Instruction 13](#_Toc29375690)

[Steps for testing 13](#_Toc29375691)

[API Testing by using POSTMAN 15](#_Toc29375692)

# Change Log

|  |  |  |
| --- | --- | --- |
| Date | Editor | Comment |
| 01/02/2020 | Yizhen Chen | Init Doc |
|  |  | Change Neo4j version from 3.5.5 to 3.5.7 |
| 01/08/2020 | Yizhen Chen | Add Backend codebase and front-end codebase |
|  |  | Add API testing by using POSTMAN |
|  |  | Add backend API testing criteria. |
|  |  |  |

# Environment Setting

## Java version

Version: java-11-openjdk

Download link : [https://www.oracle.com/](https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html)

## Tomcat version

Version: Tomcat 9.0.30

Download link : [https://tomcat.apache.org/](https://tomcat.apache.org/download-90.cgi)

## Maven version

Version:3.1

Download link: [https://maven.apache.org/download.cgi#](https://maven.apache.org/download.cgi)

## Neo4j version

Version: 3.5.7

Download link: <https://neo4j.com/download-center/#releases>

Command line to run server : ./bin/neo4j console

## Main Github CodeBase

Github: <https://github.com/CBIIT/ctdc-codebase>

\*Experimental

We try to separate the codebase into backend only and front-end only to fellow microservices structure.

To do so, we need to :

Clean up the backend and front-end codebase.

Update server deploy script

Evaluate the result.

## Backend codebase

<https://github.com/CBIIT/bento-frontend>

## Frontend codebase

<https://github.com/CBIIT/bento-backend>

\*Experimental

## Github Data Model

<https://github.com/CBIIT/ctdc-model>

## SumoCollector 19.253-6

## newrelic-infra 1.8.19-1

## Dev

GraphQL : endpoint : http://ncias-d2267-c.nci.nih.gov:7474/graphql/

Site URL : https://trialcommons-dev.cancer.gov/#/

Jenkins: <https://datacommons-jenkins.nci.nih.gov/jenkins/>

Neo4j: <http://ncias-d2267-c.nci.nih.gov:7474/browser/>

UserName : neo4j

Password : icdcDBneo4j0

Server : nciws-d2048-c, nciws-d2049-c

Neo4j server : ncias-d2267-c

## QA

GraphQL endpoint : http://ncias-d2267-c.nci.nih.gov:7474/graphql/

Site URL : https://trialcommons-qa.cancer.gov/#/

Jenkins: <https://datacommons-jenkins.nci.nih.gov/jenkins/>

Neo4j: http://ncidb-q325-c.nci.nih.gov:7474/browser/

UserName : neo4j

Password : icdcDBneo4j0

Server: nciws-q2074-c,nciws-q2075-c

Neo4j server : ncidb-q325-c

# Code Structure

## Folder Structure

ROOT

|-----src

|----main

|-----gov.nih.nci.ctdc

|----- gov.nih.nci.ctdc.controller

|----- gov.nih.nci.ctdc.error

|----- gov.nih.nci.ctdc.model

|----- gov.nih.nci.ctdc.service

|----test

|-----webContent

|----META-INF

|----WEB-INF

|----- doc

|----- pom.xml

|  |  |
| --- | --- |
| Src | Src is the folder where the project's source files are located. |
| main | Function implementation |
| test | test script (junit) |
| webContent | WebContent folder is always considered as client side code in Java web projects. It contains like HTML, Css, JavaScript, jQuery, JSP pages etc. |
| META-INF | The META-INF folder is the home for the MANIFEST.MF file. This file contains metadata about the contents of the JAR. |
| Web-INF | The WEB-INF directory is part of the directory structure that defines a particular "Web application". The WEB\_INF directory contains resources pertaining to the Web application including a web.xml file. |
| Doc | Project’s documents |
| Pom.xml | 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. |

## NameSpace

gov.nih.nci.ctdc.\*

# General Overview and Design Guidelines

A screenshot of a cell phone

Description automatically generated

Figure 1.1 Workflow

CTDC has two layers,Front-end and Back-end. Front-end is written in React.js and back-end follows API design is coded in JAVA. We put two layers together as a Java application. The figure above shows how system works and how back-end response the request.

Client as an API consumer send a request, the JAVA servlet will capture the event and dispatch to the right endpoint.

Client requests a web page, the JAVA servlet will capture the event and redirect the request to the React Component, React use Express to handle the request and render the HTML to the client.

Back-end provides two types of APIs , GraphQL and REST for purposes.

# System Design

## Backend

A close up of a map

Description automatically generated

Front-end and cloud resource will send the request to the backend. Front-end uses GraphQL endpoint by sending the GraphQL query to the backend through POST method.

The benefit for using GraphQL: reduce the number of requests needed. Front end can send the GraphQL query which indicates what front-end need, the backend will compose the data and response to request by one shot. Compare to the REST api, it may require many requests to get all kinds of data then compose in the front-end. Too many requests will cause the performance issue.

For the request from cloud resource, they may not familiar with GraphQL, so we provide the REST API for the cloud resource.

GraphQL API:

The GraphQL query will be accepted by the GraphQL endpoint. The backend will evaluate the request by checking the token and query. Token validation is part of Authorization and authentication which go through the fence. The query checking is to validate if the query is accepted or not, currently we only check the type of query, only the “query” is allowed (read only) , mutation is not permitted for now.

After validate the query and token, the GraphQL query will send to the Neo4j graphql endpoint, neo4j GraphQL plugin will translate GraphQL into Cypher and retrieve the data back to the client.

REST API :

When user send request to the REST API endpoint , REST API will take the value of parameters and compose it into a GraphQL Query. Then send it to the Neo4j graphql endpoint.

### Configuration

This project using class bean for configuration. Details can be find at class: gov.nih.nci.icdc.model.ConfigurationDAO.

This class will read setting from the configuration file application.properties and be used through out the whole java application.

Sample of configuration shows below:

spring.mvc.throw-exception-if-no-handler-found=true

spring.data.neo4j.username=neo4j

spring.data.neo4j.password=localhost

neo4j.jdbc.server=jdbc:neo4j:bolt://localhost

graphql.schema=graphql/person.graphqls,graphql/schema.graphqls,graphql/icdc.graphqls

neo4j.graphql.endpoint =http://localhost:7474/graphql/

neo4j.java.driver.server =bolt://localhost:7687

neo4j.authorization =Basic 123456

spring.mvc.view.prefix=/WEB-INF/

spring.mvc.view.suffix=.jsp

/\*\*Integration With Fence\*\*\*\*/

fence.url=http://localhost

fence.public.key=123

error.redirect\_url=http://localhost/error.html

api.version=v1

session.timeout=30

data.model.version = 1

allow\_grapqh\_query = true

allow\_graphql\_mutation =false

### Connect to Neo4j

A screenshot of a cell phone

Description automatically generated

The connection to the database neo4j go through Neo4j Graphql Endpoint. Neo4j has graphql plugin which will expose endpoint to consume the graphql request.

RESTFUL API will send the graphql request to the neo4j.

GraphQL API will validate the payload ,if is valid request then send the graphql request to the neo4j.

Implementation can be found at

Neo4JGraphQLService.java

The class send request to Neo4j.

Unirest.*post*(config.getNeo4jGraphQLEndPoint()).header("Content-Type", "application/json")

.header("Authorization", config.getNeo4jHttpHeaderAuthorization()).header("accept", "application/json")

.body(jo.toString()).asJson();

GraphQLController. isvalidQraphQL.

GraphQLProvider. isVaild

Validate if the request is vaild.

It check if the request’s payload is pre-defined . The pre-defined the graphql queries can be found at icdc.graphql.

**public** **boolean** isVaild(Document document) {

Validator validator = **new** Validator();

List<ValidationError> validationErrors = validator.validateDocument(**this**.schema, document);

**return** !(validationErrors.size() > 0);

}

### ERROR Handler

The error message have been reformatted.

The Error response will follow the structure blow.

{

"apierror": {

"status": "BAD\_REQUEST",

"timestamp": "13-06-2019 01:30:28",

"message": "Could not find the GET method for URL /v1/rest/studies/12312",

"debugMessage": "No handler found for GET /v1/rest/studies/12312",

"subErrors": null

}

}

Details can be found in the package gov.nih.nci.icdc.error

\* Using spring annotation @ControllerAdvice to customize default exception handler.

\* The 404 error will redirect to the “index.html” page, which is entrance point of react component.

### Authorization (AOP)

Details can be found at ValidationAspect.

Using AOP you can define method level control at before entering the function or after execution.

For the authorization, we add control before execute the method.

Authorization will do :

1. Check if the request’s session is expired or not
2. Check if the request has token
3. Check if the token is expired or not.

TBD: Token validation

### Logger Log4j

Using Log4j.

The configuration setting can be found at log4j.peroperties.

Details list below.

# Root logger option

log4j.rootLogger=DEBUG, stdout, file

# Redirect log messages to console

log4j.appender.stdout=org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target=System.out

log4j.appender.stdout.layout=org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

1

# Redirect log messages to a log file

log4j.appender.file=org.apache.log4j.RollingFileAppender

#outputs to Tomcat home

log4j.appender.file.File=${catalina.home}/logs/log.log

log4j.appender.file.MaxFileSize=50MB

log4j.appender.file.MaxBackupIndex=10

log4j.appender.file.layout=org.apache.log4j.PatternLayout

log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n

### Testing

The testing framework is Junit + Spring MockMVC.

MockMVC class is part of [Spring MVC](https://howtodoinjava.com/spring-mvc-tutorial/) test framework which helps in testing the controllers explicitly starting a servlet container.

Details can be found in the classes below.

GraphQLControllerTests.java

## Frontend

### Frontend Backend integration

Front end is react component, it requires compile and build to collect the libraries and generate the static html file(index.html) .

Java will redirect the 404 error to the index.html.

Frond end code check in location :

SRC/Main/frontend

Compile and build is controlled by maven.

In the POM file , the code list below, shows how maven able to do npm install and npm build.

<plugin>

<groupId>com.github.eirslett</groupId>

<artifactId>frontend-maven-plugin</artifactId>

<version>1.7.6</version>

<configuration>

<workingDirectory>src/main/frontend</workingDirectory>

<installDirectory>target</installDirectory>

</configuration>

<executions>

<execution>

<id>install node and npm</id>

<goals>

<goal>install-node-and-npm</goal>

</goals>

<configuration>

<nodeVersion>v12.4.0</nodeVersion>

<npmVersion>6.9.0</npmVersion>

</configuration>

</execution>

<execution>

<id>npm install</id>

<goals>

<goal>npm</goal>

</goals>

<configuration>

<arguments>install</arguments>

</configuration>

</execution>

<execution>

<id>npm run build</id>

<goals>

<goal>npm</goal>

</goals>

<configuration>

<arguments>run build</arguments>

</configuration>

</execution>

</executions>

</plugin>

After build, it will move the static files to output folder.

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>3.2.3</version>

<configuration>

<webResources>

<resource>

<!-- this is relative to the pom.xml directory -->

<directory>${project.build.directory}/generated-docs</directory>

<targetPath>/static/docs/api</targetPath>

</resource>

<resource>

<!-- Include the react app index.html and build.js -->

<directory>src/main/frontend/dist/</directory>

<targetPath>/WEB-INF/</targetPath>

</resource>

</webResources>

</configuration>

</plugin>

# APIs

## Prepare for testing

The URL for testing

POSTMAN for testing REST / GraphQL endpoint : (

download link : <https://www.getpostman.com/downloads/>

GraphiQL for testing GraphQL endpoint:

download link : <https://electronjs.org/apps/graphiql>

## GraphiQL’s Instruction

A screenshot of a social media post

Description automatically generated

### Steps for testing

The figure above shows how to use graphiQL.

Step 1: Type Endpoint URL (<http://localhost:8080/v1/graphql/>)

Step 2: Type GraphQL as an input

{

dashboard(){

}

}

Step 3: Click Run 

## POSTMAN’s Instruction

A screenshot of a computer screen

Description automatically generated

### Steps for testing

The figure above shows how to use POSTMAN.

Step 1: Type Endpoint URL (http://localhost:8080/v1/rest/studies)

Step 2: Choose Method as GETA picture containing screenshot

Description automatically generated

Step 3: Click Send A screenshot of a cell phone

Description automatically generated

Step 4: In the response body, choose Pretty and Json for the styling.

A screenshot of a cell phone

Description automatically generated

**GraphQL Testing by using POSTMAN**

Step 1: Type Endpoint URL (https://caninecommons-dev.cancer.gov/v1/graphql/)

Step 2: Choose Method as GETA screenshot of a cell phone

Description automatically generated

Step 3: Choose Body-> GraphQL in Nav bar. In the Query panel, type in the graphql query as show below.A screenshot of a social media post

Description automatically generated

Step 4: Click Send A screenshot of a cell phone

Description automatically generated

Step 5: In the response body, choose Pretty and Json for the styling.

A screenshot of a computer

Description automatically generated

## API Testing by using POSTMAN

Postman provides feature that can run bunch of Requests and we can define tests in each request.

**Backend only test the status of the request and response should contains data and no errors in the payload.**

Backend Test Script can be found in CTDC.postman\_collection.json

<https://nih.app.box.com/file/593271047528>

pm.test("Status test", function () {

pm.response.to.have.status(200);

});

pm.test("Body matches string", function () {

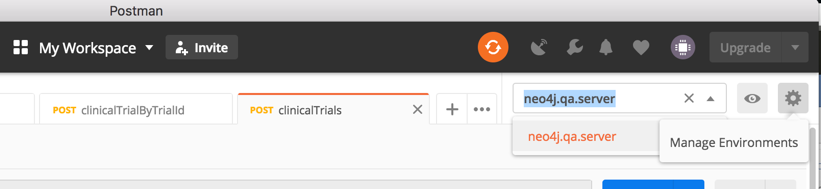
pm.expect(pm.response.text()).to.not.include("errors");

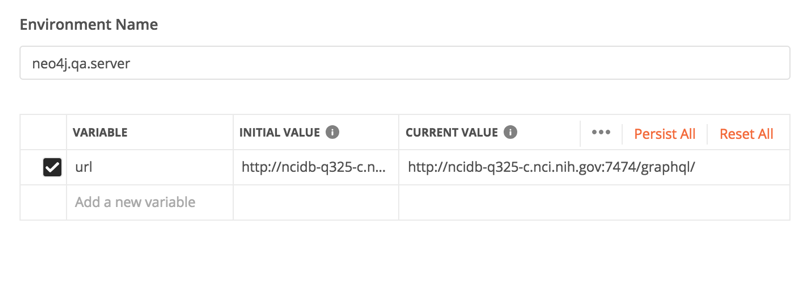
});

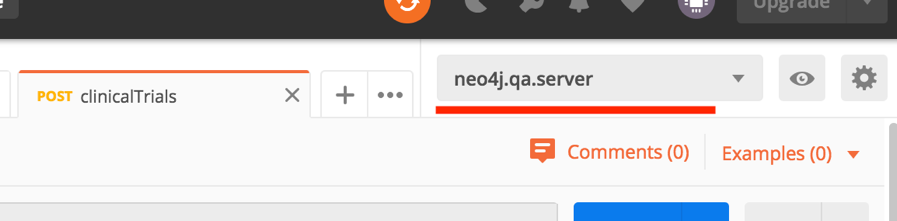
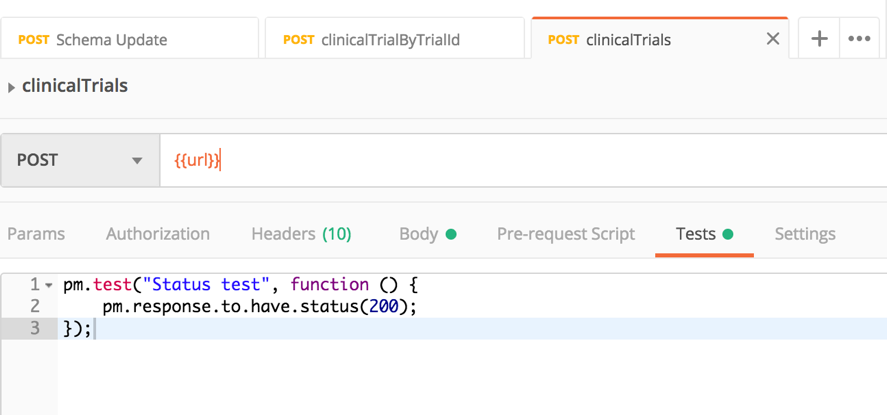
pm.test("Body matches string", function () {

pm.expect(pm.response.text()).to.include("data");

});

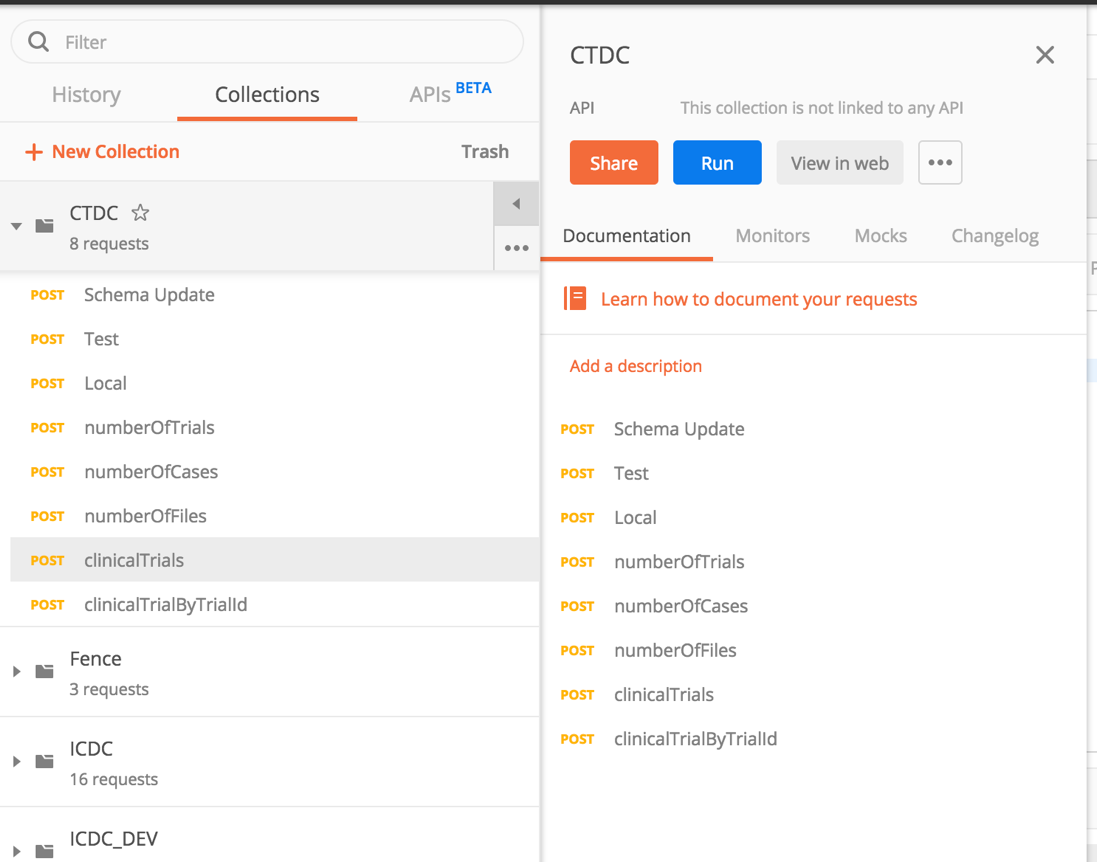
1. Define env variable   
    Refers :   
   <https://learning.getpostman.com/docs/postman/variables-and-environments/variables/>  
   - We use env variable to present QA, DEV server URL, then we can apply same request / test to the QA or DEV environment.
   1. Click manage environment setting 
   2. Add env variable

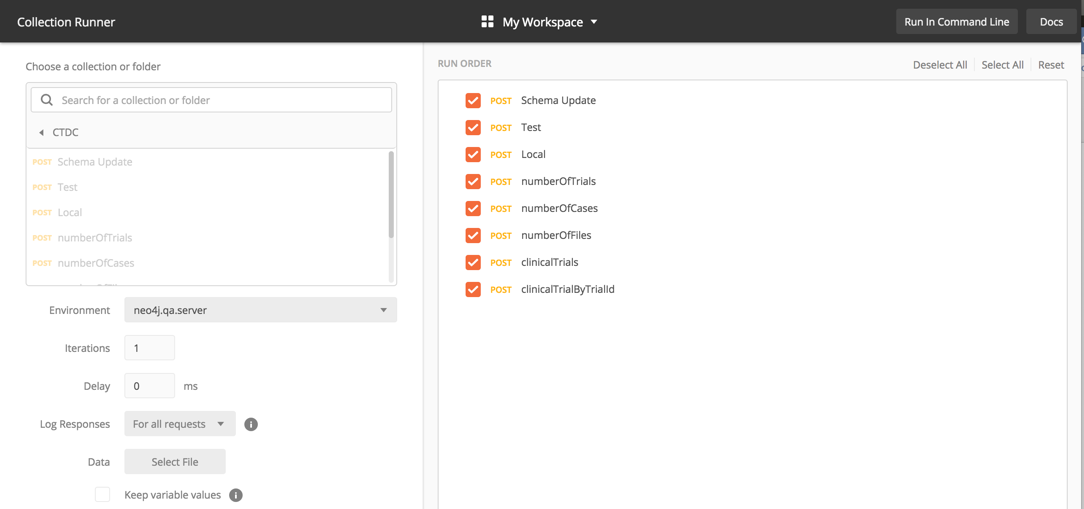


* 1. After click add button, in the manage environment variable panel, close the panel. 
  2. Use env variable ( using {{env variable }})

1. Define Test case  
   Refers : <https://learning.getpostman.com/docs/postman/scripts/test-scripts/>
2. Run collection of requests   
   Refers : <https://learning.getpostman.com/docs/postman/collection-runs/intro-to-collection-runs/>

Run collection



Click run get into runner panel

Execute

