**Coding Challenge for Technical Consultants**

**Date: April 10, 2024**

Table of Contents

[About the Candidate 1](#_Toc163652631)

[Setting up the environment 1](#_Toc163652632)

[REST APIs 2](#_Toc163652633)

[JSON Libraries 2](#_Toc163652634)

[Package Manager & Build Tool 2](#_Toc163652635)

[Java Runtime 2](#_Toc163652636)

[Configuring Java CLASSPATH 2](#_Toc163652637)

[Logging 2](#_Toc163652638)

[Testing Framework 3](#_Toc163652639)

[Building the App 3](#_Toc163652640)

[Git Hub 4](#_Toc163652641)

[Running the allocation is as easy as: 4](#_Toc163652642)

[Q & A 4](#_Toc163652643)

[Work Log 4](#_Toc163652644)

[Edge cases & other considerations 5](#_Toc163652645)

[Monolithic vs. Microservices systems 5](#_Toc163652646)

[Few modern UI frameworks 5](#_Toc163652647)

[Containers, Docker and Kubernetes 5](#_Toc163652648)

## About the Candidate

Thomas G. Peters is in Pittsburgh, PA having a number of years of experience in IT as a java coder and well as a certified Azure Solutions Architect. Thomas enjoys building, camping and riding motorcycles.

## Setting up the environment

It may seem like allot to do getting the environment configured but it’s basically a one time process. [VSCode](https://code.visualstudio.com/) is my IDE of choice.

### REST APIs

Java application coding to create a client app with REST connectivity to the Coding Challenge endpoint with JSON library integration. Several libraries were considered in this analysis phase including [RESTEasy](https://www.baeldung.com/resteasy-client-tutorial) and [Spring RESTClient](https://docs.spring.io/spring-framework/reference/integration/rest-clients.html) of which I settled on Spring.

### JSON Libraries

JSON parsing and processing can also be accomplished in many ways. I chose Fast XML [Jackson](https://github.com/FasterXML/jackson) data binding to complete this as these are easy to use and rock-solid.

### Package Manager & Build Tool

There are several package managers that can work in this situation including Maven and Groovy. Since I have experience with [Maven](https://maven.apache.org/), it was a no brainer. Setup is easy! Once you have maven installed, you can validate the cli using the ‘mvn’ cmd. Then it’s easy to initialize a blank project using the architypes as shown below.

### Java Runtime

Ensure you are running java version 1.8 or above. The project will operate in Maven with using a dependency list however you must configure your CLASSPATH separately if you want to execute the project outside the scope of the maven lifecycle phases.

### Configuring Java CLASSPATH

There are several ways to configure your Java runtime environment. You can specify the CLASSPATH using a runner script, or you can set your CLASSPATH globally in your shell. I choose to set my CLASSPATH globally in ‘.zshrc’ which is a script that runs automatically when you open a net terminal. It’s transparent to the user in this way.

export CLASSPATH=$CLASSPATH:$SPRING\_HOME/lib/spring-boot-cli-3.2.4.jar:~/Development/Java/camunda-challenge/target/camunda.challenge-1.0-SNAPSHOT.jar:$SPRING\_HOME/lib/spring-core-6.2.0-SNAPSHOT.jar:$SPRING\_HOME/lib/json-simple-1.1.1.jar:$SPRING\_HOME/lib/jackson-core-2.15.3.jar:$SPRING\_HOME/lib/jackson-databind-2.15.3.jar:$SPRING\_HOME/lib/jackson-annotations-2.15.3.jar:$SPRING\_HOME/lib/spring-web-6.2.0-SNAPSHOT.jar:$SPRING\_HOME/lib/spring-boot-cli-3.2.4.jar:$SPRING\_HOME/lib/micrometer-commons-1.12.5.jar:$SPRING\_HOME/lib/commons-logging-1.3.1.jar:$SPRING\_HOME/lib/micrometer-observation-1.12.5.jar

### Logging

Log4j using [slf4j](https://slf4j.org/) libraries.

### Testing Framework

[Junit](https://junit.org/junit5/) comes configured in the simple Maven architype, so when generating the app it’s already there.

## Building the App

Validate you Maven Installation by performing the following command:

|  |
| --- |
| %> mvn --version  **Apache Maven 3.9.6 (bc0240f3c744dd6b6ec2920b3cd08dcc295161ae)**  Maven home: /usr/local/Cellar/maven/3.9.6/libexec  Java version: 17.0.2, vendor: Azul Systems, Inc., runtime: /Library/Java/JavaVirtualMachines/zulu-17.jdk/Contents/Home  Default locale: en\_US, platform encoding: UTF-8  OS name: "mac os x", version: "14.5", arch: "x86\_64", family: "mac" |

Maven is a common package manager and build tool for java development. Maven enables you to generate new projects using **archtype:generate** using the **quickstart** archetype.

|  |
| --- |
| %> mvn archetype:generate -DgroupId=com.camunda.tgp -DartifactId=camunda-challenge -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false |

|  |
| --- |
| [**INFO**] Scanning for projects...  [**INFO**]  [**INFO**] **------------------<** org.apache.maven:standalone-pom **>-------------------**  [**INFO**] **Building Maven Stub Project (No POM) 1**  [**INFO**] **--------------------------------[ pom ]---------------------------------**  [**INFO**]  [**INFO**] **>>>** archetype:3.1.2:generate **(default-cli) > generate-sources** @ standalone-pom **>>>**  [**INFO**]  [**INFO**] **<<<** archetype:3.1.2:generate **(default-cli) < generate-sources** @ standalone-pom **<<<**  [**INFO**]  [**INFO**]  [**INFO**] **---** archetype:3.1.2:generate **(default-cli)** @ standalone-pom **---**  [**INFO**] Generating project in Batch mode  Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.pom  Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.pom (1.6 kB at 7.3 kB/s)  Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.jar  Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.jar (7.1 kB at 197 kB/s)  [**INFO**] ----------------------------------------------------------------------------  [**INFO**] Using following parameters for creating project from Archetype: maven-archetype-quickstart:1.4  [**INFO**] ----------------------------------------------------------------------------  [**INFO**] Parameter: groupId, Value: com.camunda.tgp  [**INFO**] Parameter: artifactId, Value: camunda-challenge  [**INFO**] Parameter: version, Value: 1.0-SNAPSHOT  [**INFO**] Parameter: package, Value: com.camunda.tgp  [**INFO**] Parameter: packageInPathFormat, Value: com/camunda/tgp  [**INFO**] Parameter: package, Value: com.camunda.tgp  [**INFO**] Parameter: groupId, Value: com.camunda.tgp  [**INFO**] Parameter: artifactId, Value: camunda-challenge  [**INFO**] Parameter: version, Value: 1.0-SNAPSHOT  [**INFO**] Project created from Archetype in dir: /Users/thomaspeters/Development/Java/camunda-challenge  [**INFO**] **------------------------------------------------------------------------**  [**INFO**] **BUILD SUCCESS**  [**INFO**] **------------------------------------------------------------------------**  [**INFO**] Total time: 2.888 s  [**INFO**] Finished at: 2024-04-08T16:29:27-04:00  [**INFO**] **------------------------------------------------------------------------** |

Do a simple test to ensure your project can run:

|  |
| --- |
| %>java -cp target/camunda-challenge-1.0-SNAPSHOT.jar com.camunda.tgp.App  Hello World! |

Maven has generated a project, created a **pom.xml** file, and create two java files. The first java is an App the second the TestApp.

## Git Hub

I have shared this app in a public repository on GitHub. Please clone the project with the following command into your local environment to run the project.

|  |
| --- |
| %> git clone https://github.com/thomasgpeters/camunda-challenge.git |

## Running the allocation is as easy as:

|  |
| --- |
| %> mvn clean  %> mvn package  %> mvn test |

## Q & A

### Work Log

|  |  |  |
| --- | --- | --- |
| Date | Tasks | Hours |
| March 8th, 2024 | Received the assignment to complete the Camunda Challenge. | 0.5 |
| Configuration and setup | 2.5 |
| March 9th, 2024 | Selecting and testing REST Client | 1.0 |
| Selecting and testing JSON library | 0.5 |
| Generate project using Maven archetype | 0.5 |
| Coding REST client  (multiple configuration challenges with **RESTEasy** and resorted to Spring **RestClient** for a rewrite) | 3.0 |
| Coding Response processing with JSON library | 2.0 |
| March 10th, 2024 | Coding Test cases and hardening java instances | 1.0 |
| Running test cases and packaging the project | 3.0 |
| Writing this document | 2.0 |
| Total | | **16** |

### Edge cases & other considerations

It seems to me that the application can continue to get more robust and provide new features such as querying the users with search criteria and updating (push or put) APIs. Also, the API provides some metadata such as the number of rows, as well as paging. The client can be enhanced using this data to provide additional robustness or validations. The output of one API can feed inputs into the next API call for example.

### Monolithic vs. Microservices systems

Monolithic systems pack allot of functionality into a single environment. Examples of monolithic apps are SAP or Pegasystems. One problem with monolithic architecture is if one functional area of the application goes down, everything goes down. The same application is a web server as well as a REST service broker as well and business logic and decisioning all wrapped up into a single large multi-purose application. Another problem is Change Management. When new features are released to GA, then the entire monolithic application needs to be released. Clients are forced to swap out the entire application stack in a big-bang approach with introduces allot of risk.

A logical solution to this problem is the microservices architecture. This approach delivers functionality in smaller services that can stand on their own. These smaller apps run in isolation and provide a loose coupling through data but are not otherwise dependent on each other. The microservices app can have its own deployment cycles and are able to have a separate and independent delivery schedule.

### Few modern UI frameworks

Angular, Reach and Vue are three modern UI frameworks for application delivery based on Node.js (TypeScript and JavaScript). Python also has some nice frameworks such as Flask. Java provides Spring boot and RxJava among others.

### Containers, Docker and Kubernetes

Containers run in isolation within a runtime providing portability and isolation from the local host operating system. Docker is a runtime container platform as well as **containerd**. When operating with containers, you first must build you docker image or images. A docker image is the development time definition of your container, while the container is the running version of those images. Docker images are built in layers. A developer uses the **DockerFile** to define how these layers should be assembled.

Kubernetes is a server-side container platform that runs at scale. This means we can support multiple nodes within a Kubernetes cluster. Written originally by Google, there are many brands of Kubernetes available AKS, EKS and GKE are three such brands. Azure Kubernetes Service (AKS), Elastic Kubernetes Service (EKS) and Google Kubernetes Engine (GKE) are all cloud-based services. However, you can download the Kubernetes runtime admin tool kubeadmin and run it locally.