Architecture Analysis for client X

Contents

[Architecture 2](#_Toc118108137)

[Connection to MS-SQL Server 2](#_Toc118108138)

[Spring Boot Architecture 2](#_Toc118108139)

[Architecture Overview 2](#_Toc118108140)

[Application Architecture 3](#_Toc118108141)

[Considerations 4](#_Toc118108142)

# Architecture

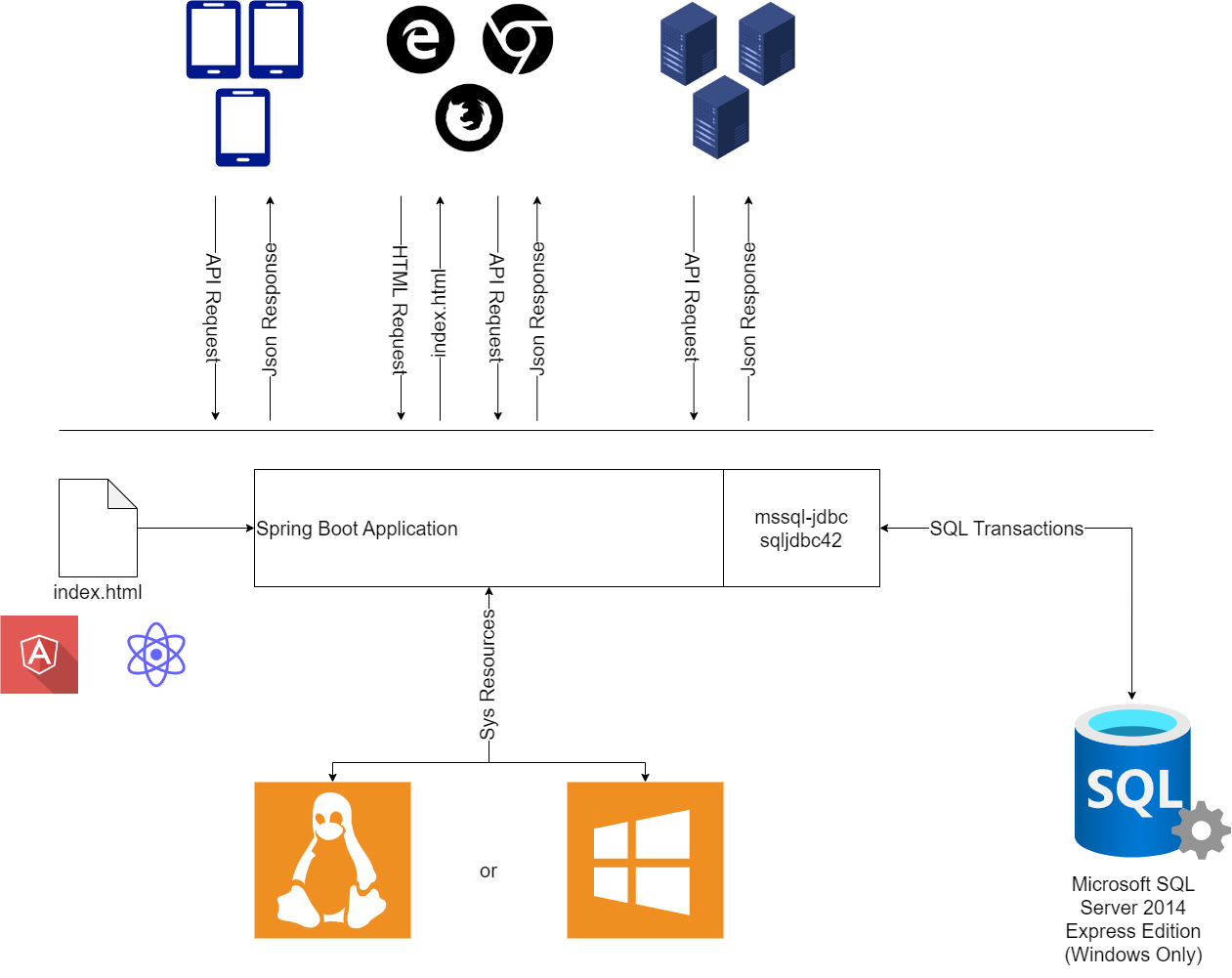
The objective is to have a browser application that presents the information available in a Microsoft SQL Server database. The information must be processed through a backend in Java in order to ensure compatibility to the previous developed code from an Integration Server.

For the server application the proposal is to use the Spring Boot framework. Reasons for Spring Boot:

* Support from a huge community
* Reliable framework
* Highly customizable to attend to all types of business logic
* Support for all types of ORMs
* Support for Spring Security plugin to ensure the safety of the APIs.

## Architecture Overview

The server application is to be hosted in an on-premise infrastructure accordingly to the next image:



The Operating System to support the server application can be Linux based or a Windows server machine. The server application uses the Spring Boot framework to handle the HTTP requests and process the data from the data source. The data source is the MS-SQL Server database, and the connections to the database are managed through the JDBC packages specific to that database.

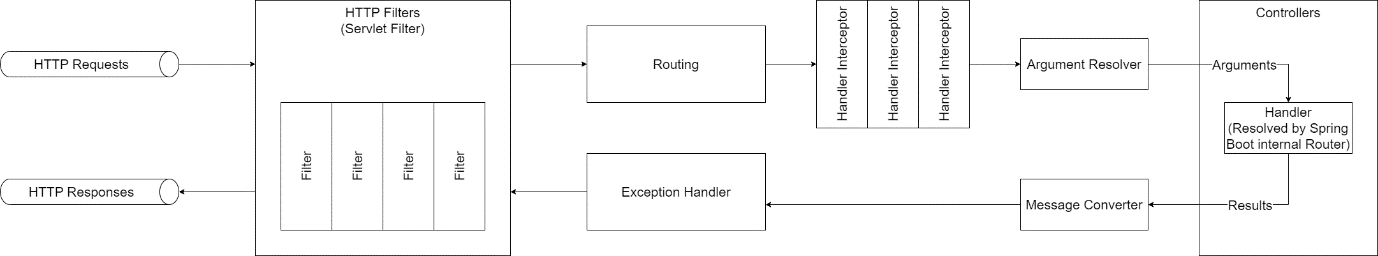
The frontend application can be developed using any framework that is then compiled to the resources/static folder in the server application. This ensures that the index.html is a resource that is part of the server application and that the \*.jar file also contains the browser application. As off this, the \*.jar file can be exported to any environment and will always contain the same compiled browser application.

Although the server application can respond with the index.html file a direct call to the APIs is also achievable. To call the API instead of the browser app, the client just needs to specify the correct URL. For example, a request to GET https://xyz responds with the browser app (aka index.html) and a request to GET https://xyz/api/sample responds with the Json representation of the resource Sample.

## Spring Boot Architecture

Spring boot manages the infrastructure of the application leaving the developer to only focus in the business logic. This is achieved by the starter dependencies that brings a ton of autoconfiguration classes which create beans with default configurations.

From the listening of a request to the action of sending the response everything is customizable:



The logic business is usually developed as a service class that is injected on the controller at runtime. The injection must be of the type Constructor Injection in order to assure immutable Controller objects. The developed services also receive the dependencies they need through Constructor Injection. These dependencies can be other services or the connection to the database. Typically, the database connection is not injected, what is injected is the Repository of data. This repository is a Crud Repository in its core with support to create, read, update and delete operations.

The Spring Boot framework infers the best handler to execute each request through the URI. This means that every handler is attached to a specific URI and also an HTTP method. In order to attach a handler to an URI and HTTP method is used the @RequestMapping annotation or its sub annotations.

## Connection to MS-SQL Server

The connection to the database can be managed in three ways:

* Spring JDBC: only manages connections
* Spring Data JDBC: basic Object Relational Mapping (ORM) features
* Spring Data JPA: full ORM features with support to pagination, caching and lazy loading.

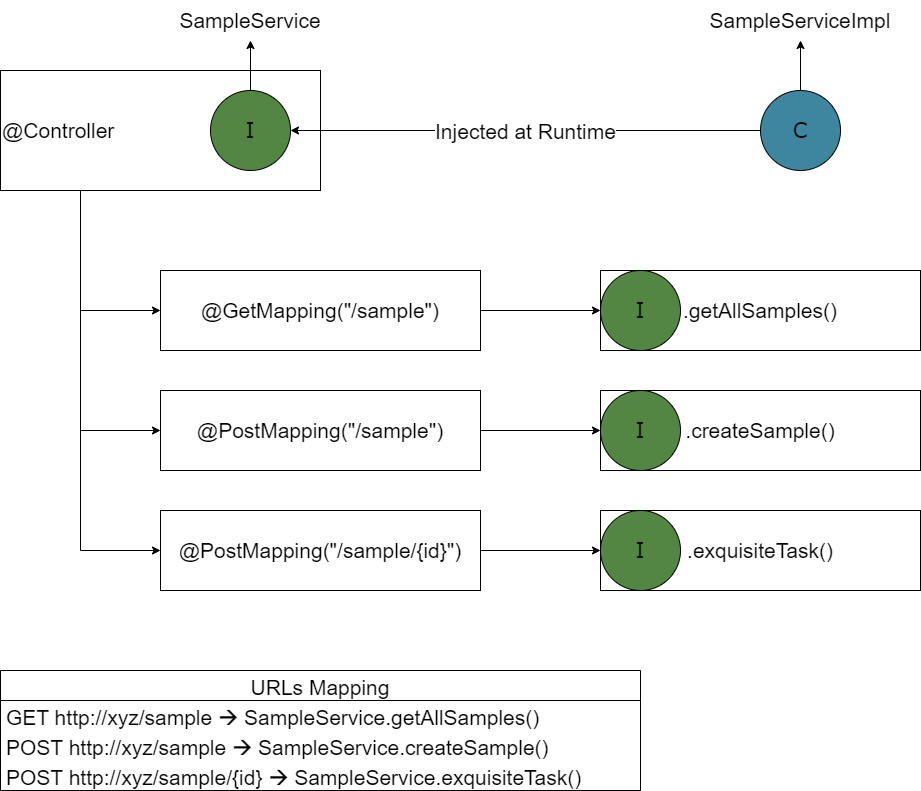
The compatibility with the database is guaranteed by JDBC package in all the options above. This means that all the projects need to include the com.microsoft.sqlserver:mssql-jdbc and sqljdbc42.jar dependencies.

## Application Architecture

The business logic is wrapped in a java interface and its implementation class. The class is annotated with @Service and is injected in the Spring Controllers at runtime. In order to the injection to happen the Spring Controller must receive the interface as a parameter in the constructor.

These services are called in the Controller Handlers (the methods annotated with @RequestMapping) to execute the business logic that is needed.

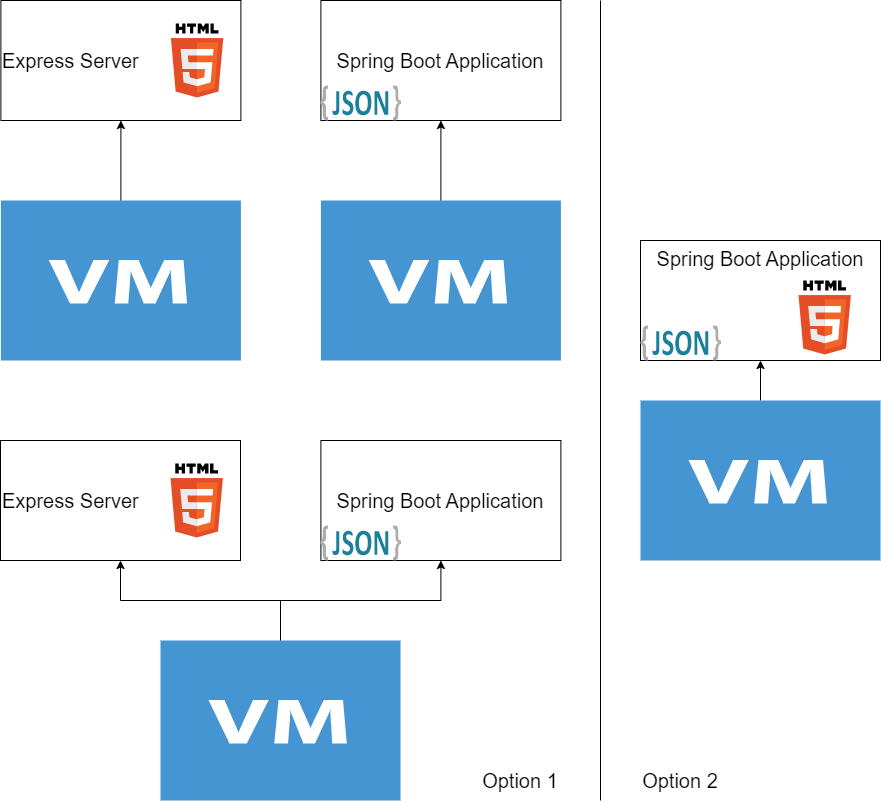
There is a lot of conventions when working with Spring Boot applications and one of them says that for each handler exists a single method in the service interface.



## Browser Application Architecture

In order to serve the browser application there are two solutions:

1. Specific frontend server: A server that only responds with the index.html and its resources.
2. Coupled with Application server: Tweak the application server to also serve the index.html and its resources.



# Considerations

As Spring Boot is a java application the code can run on any Operating System with an existing JVM. The Spring Boot package dependencies are usually managed by a build system like Maven or Gradle. This build systems can support our application and build directly a fat jar (a jar that contains all dependencies) that runs the application with command java -jar \*.jar. With the support of Gradle other systems can be installed and executed. For example, Gradle can install the node.js environment, run the build command to compile a react application and move the index.html and bundle.js to the resources/static folder providing a way for the Spring Boot application to not only serve the APIs but the Browser Application too (for this feature to work is necessary to implement a handler to respond with the index.html for all unknow URLs).

The database compatibility depends on the JDBC package imported and not the one provided by the Spring Data family libraries. In case of the MS-SQL Server database this is true, but there are pre-build starters dependencies with the driver pre-configured like Spring Data Postgres that are more performant for this specific database. In case of a NoSQL database the best case is to use the starter dependency for that specific database as is the case of Elasticsearch or MongoDB with Spring Data Elasticsearch and Spring Data Mongo dependencies respectively.

While managing dependencies in the application itself with Spring via dependency injection, it must be done using constructor injection in order to make the objects immutable. This is especially important for injecting dependencies in the services because this way the service can be reused in other applications not spring based. Not only makes the services reusable, but for unit testing, the possibility of instantiating the service without the spring framework is a must.