This document explains the :

1. Architecture / design principles / design patterns
2. Assumptions
3. Technology for development.
4. **Technology selection.**

**Spring boot** is used for development along with java 1.8

* we use Spring boot’s support for embedding the [Tomcat](https://spring.io/understanding/Tomcat) servlet container as the HTTP runtime, instead of deploying to an external instance.
* Spring boot has replaced lots of annotation with @SpringBootApplication
* The main() method in Application.java uses Spring Boot’s SpringApplication.run() method to launch an application. Not a single line of XML!! No web.xml file either. This application is 100% pure Java and you didn’t have to deal with configuring any plumbing or infrastructure.
* Spring boot makes packaging easy. It creates a single jar file with all the dependencies. (spring-test-service-1.0.0.jar)
* Supports Spring MVC internally when it sees spring-webmvc and related annotations (@RequestMapping etc) on the classpath. This flags the application as a web application and activates key behaviors such as setting up a DispatcherServlet.

**Hibernate (with JPA) along with h2 database** is used for persistence. Spring-boot provides in built support for both via configuration in pom.xml. Once the spring boot application is completely up and running you can view the h2 console at <http://localhost:8080/console>

ProductRepository.java performs the CRUD operations and Product and Category are domain classes saved in “products” and “category” tables respectively.

1. **Abstraction Used.**

* SpringTestService implements an Interface ISpringTestService so that another implementation can be plugged in SpringTestController if needed.
* Each class does exactly one thing (Single Responsibility Principle) such as Spring TestController just accepts requests and passes to SpringTestService.

1. **Assumptions / Limitations**

* The SpringTestController.checkout() is kept simple for receiving just a bunch of parameters. It can be enhanced to receive a map instead. POST can also be used but ideally implemented as a GET method since we are trying to "Get" report from server for the products to be checked out. Also the itemized bill is sent as a simple String report but can be enhanced later.

1. **How to Run**

The Itemized Report can be seen on console after running SprintTestComponentTest**.** testCheckout() component test. Search by “Itemized Report” in console output. Please do not limit the console output (uncheck “limit console output in console Properties”). It looks like :

Itemized Report

Product Name Category Product Cost Sales Tax Net Cost

===================================================================

Product\_A Category\_A 100.0 10% 90.0

Product\_B Category\_B 150.0 20% 120.0

Product\_C Category\_C 200.0 0.0% 200.0

===================================================================

Total Cost : 410.0

You will also notice an exception on console :

java.lang.Exception: Invalid User : Authentication failed

This is expected and coming from 2nd test testCheckoutInvalidUser();

1. **Logging**

Application level logging is enabled in spring boot’s **application.properties** file under src/main/resources. It defines the output logging file and the logging verbosity level. This is currently bundled within the project but can be easily externalised and passed as an argument to the java process. The **spring-test-service.log** file is created from where the process is run i.e under “target” folder if run from command line or under project directory if run from eclipse.

1. **Component / Unit Tests**
2. **SprintTestComponentTest** : Contains testCheckout() component test which saves the testdata in to database and fires a REST call for checking out products. Also contains testCheckoutInvalidUser() test for testing security in application.
3. **SpringTestServiceTest** : contains test methods for testing the service
4. No tests needed for testing DAO since we are using JPA Repository and no extra implementation needed for application as such.
5. **Security**

Spring Aspect (AOP) is used to implement security as a cross cutting concern. SecurityInterceptor intercepts the REST api to controller and checks valid user for now but this can be extended to talk to a datastore or an authentication provider.

1. **Software Used :**

IDE : Eclipse Luna

Build : Maven Eclipse Plugin 1.6.1

Project : Spring-Maven project

Java : 1.8

Spring-core 4.1.5.RELEASE

Spring boot 1.2.2.RELEASE ( For Rest Support )

Junit 4.11

Mock Framework : spring-test-mvc 1.0.0.M2

To build,

mvn clean install

1. **Code Location**

GitHub link : <https://github.com/nikhilmproject/SpringTest>

Repository : SpringTest

Branch : master