Considerations for additional modifications given time:

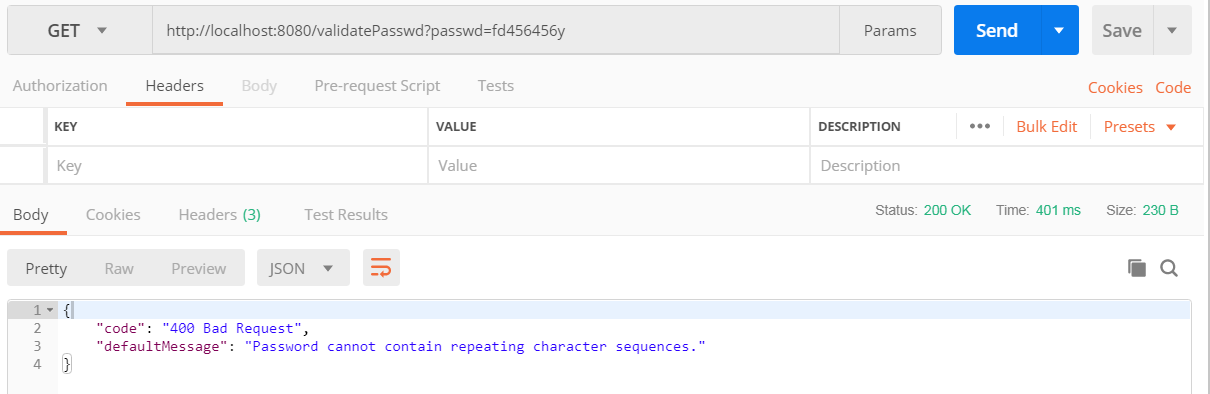
* This project was developed using top-down TDD (starting with Integration test cases down to Unit tests and through TDD cycles to completion.
* Used STS (“Springified Eclipse”) and Spring Boot for development however deployment to lower and upper enterprise environments would be to Tomcat, Weblogic, AWS, or even implemented via AWS Lambda and API Gateway as required.
* Project built using proper package structure as per requirements. Would include data tier including Repository Class and resolution and return of logical View if complete MVC were required.
* Built as an executable jar for Boot however could be modified to war for enterprise deployment with two simple changes to the pom and would have likely begun using a web project starter to include web descriptors, views, and other web artifacts.
* Using GET here as per these specific requirements but would use other http verbs as needed for business logic (i.e. POST persisting passwd history etc).
* Controller could return the entire ResponseEntity Object depending on View requirements.
* Implement react/WebFlux if UI was required.
* Implement Spring Boot 2.x Actuator for diagnostic endpoints such as /health, /info, /threaddump, /env and all other useful Actuator features for REST API monitoring and diagnostics.
* Perhaps alternatively implement validation constraints on Password Object attributes (min/max, NotNull with message etc).
* REST docs—perhaps RAML, Swagger etc.
* Add more robust logging— Splunk also useful for logs analysis etc.
* Implement Java 8 where beneficial (i.e. Stream operations and/or Lambdas for loops and test conditions where that may be beneficial without compromising execution cost.
* Code cleanliness: all code was run through format and all unnecessary imports cleaned for ideal readability.

Notes:

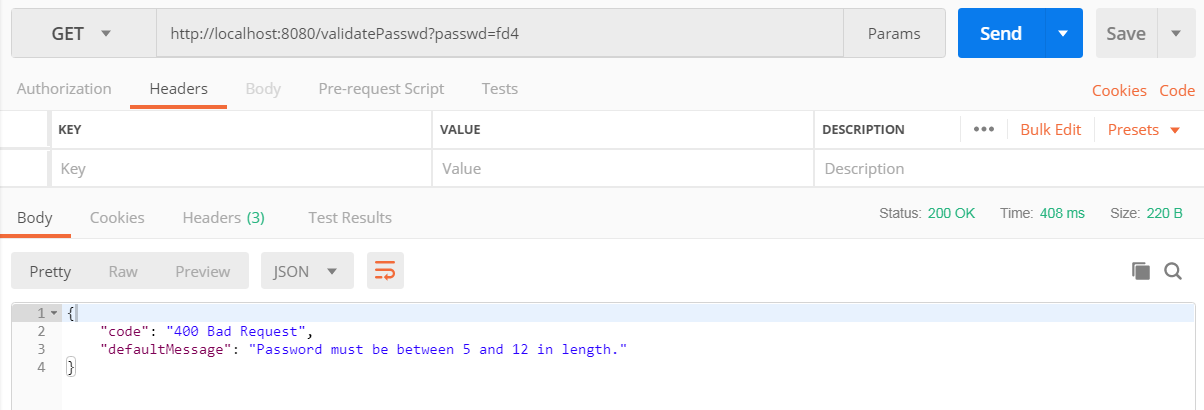
* This simple REST implementation would be a Service Component (CS) within a Microservices Architecture and possibly refactored to be more coarse or fine grained depending on the topology used, orchestration with other Service Components, and additional business requirements respectively.
* To examine this project simply import into STS or your IDE of choice with Spring support.
* Current Javadocs are located in /docs folder.
* Testing during development was done via spring boot wrapping mockito/junit/hamcrest via spring-boot-starter-test dependency. Each test case can be tested independently using run-as JUnit test or build via POM where Spring Boot annotations will run these tests at build time. **See example pics below**.
* Post dev testing done using postman. Execute PasswdValidatorProjectApplication and send requests. **See example pics below**. Example payload:

SCREENSHOTS OF TESTING POSTMAN

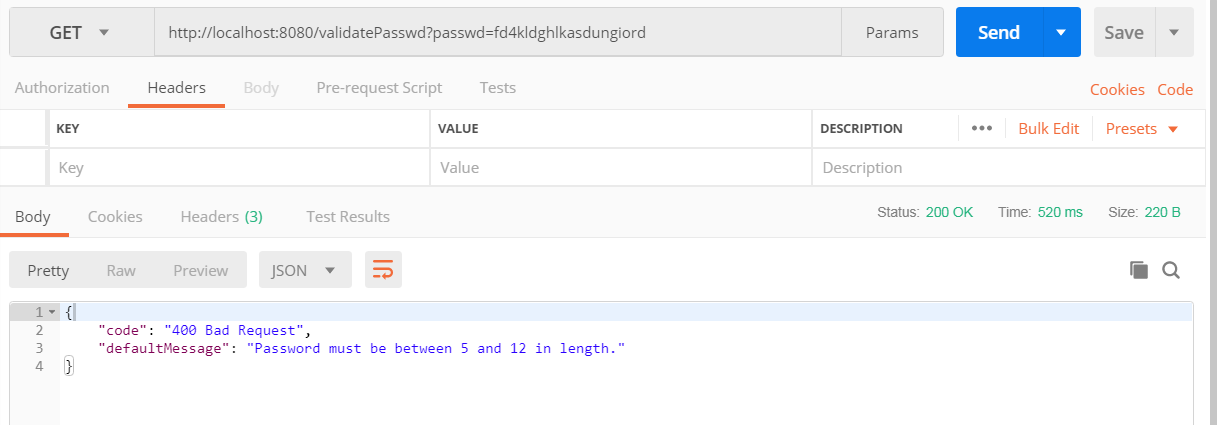
Postman test example (**http://localhost:8080/validatePasswd?passwd=fd456456y**) – to test for repeating char sequence (456456):



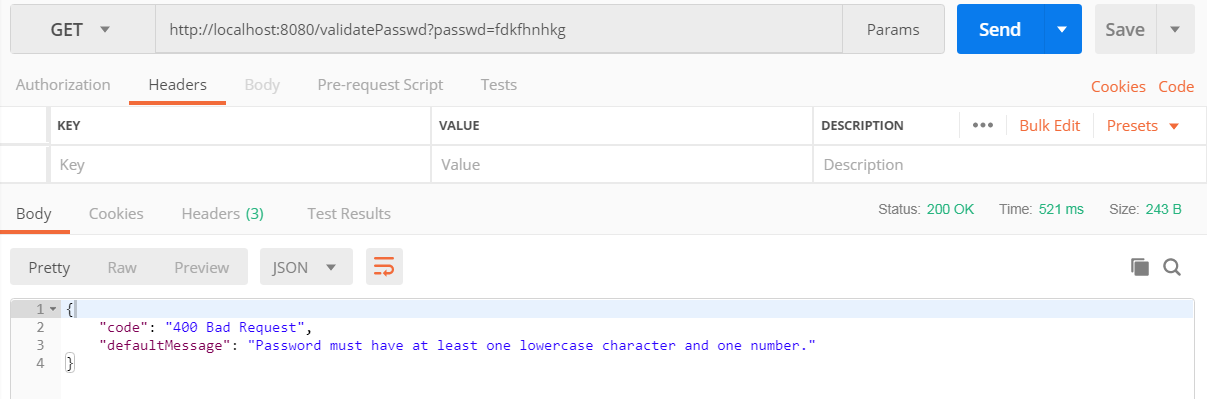
Postman test example (**http://localhost:8080/validatePasswd?passwd=fd4**) – to test passwd parameter length (too short):



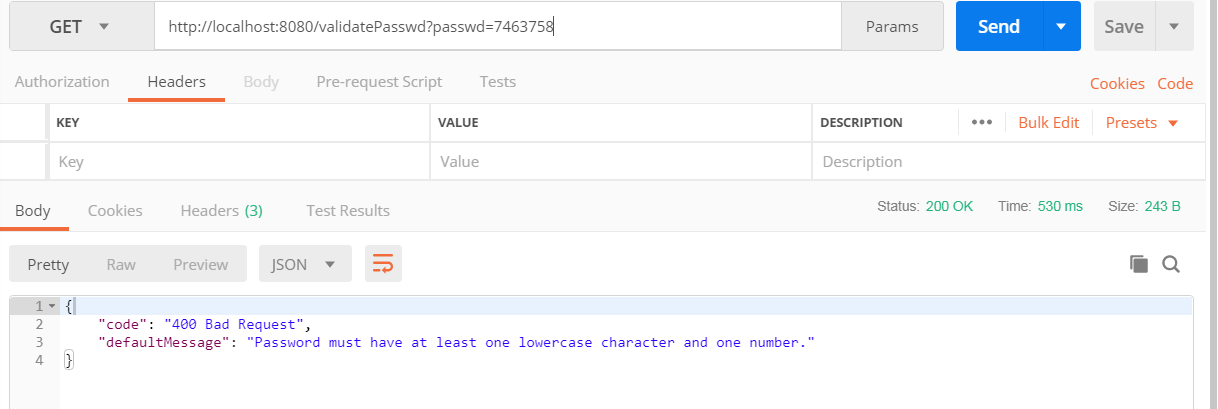
Postman test example (**http://localhost:8080/validatePasswd?passwd=fd4**) – to test passwd parameter length (too long):



Postman test example (**http://localhost:8080/validatePasswd?passwd=fdkfhnhkg**) – to test passwd parameter contains at least one number and one character (no number):

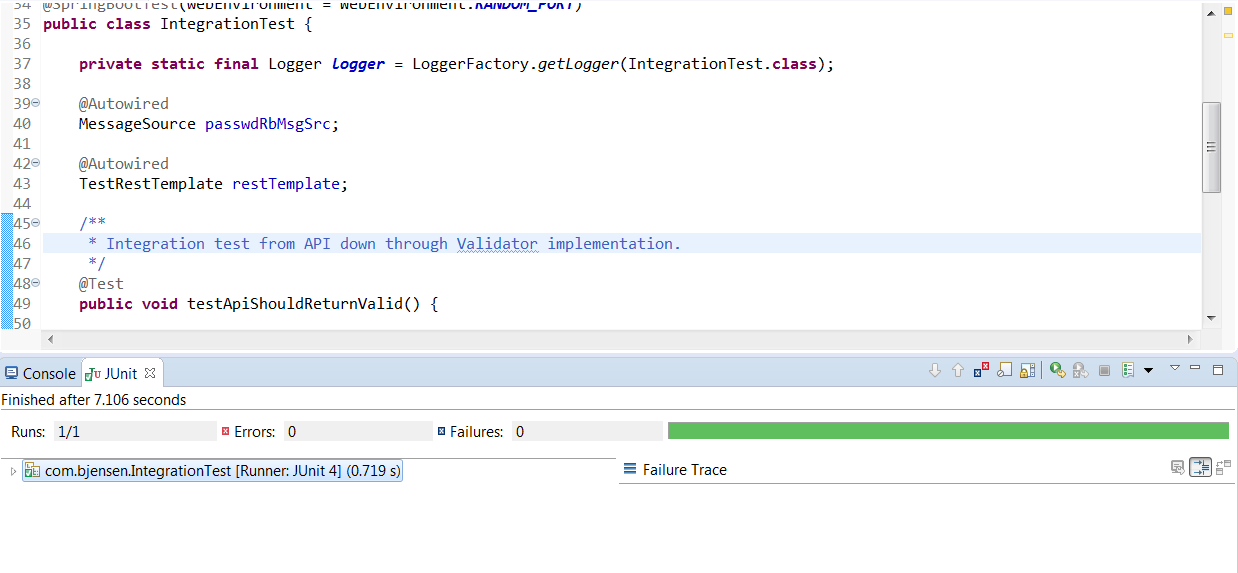


Postman test example (**http://localhost:8080/validatePasswd?passwd=7463758**) – to test passwd parameter contains at least one number and one character:

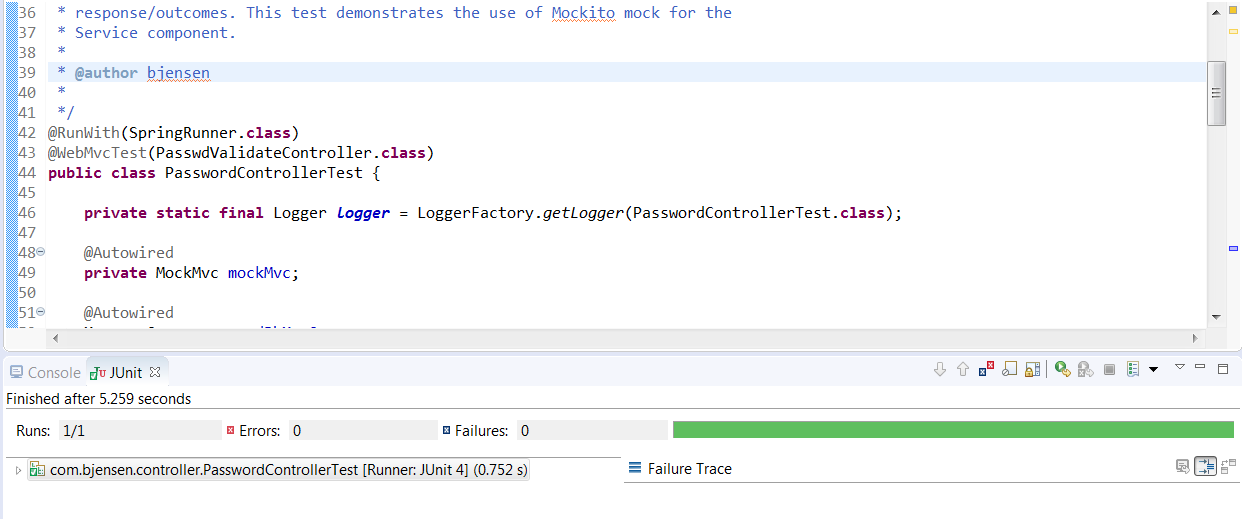


SCREENSHOTS OF INTEGRATION AND UNIT TESTING

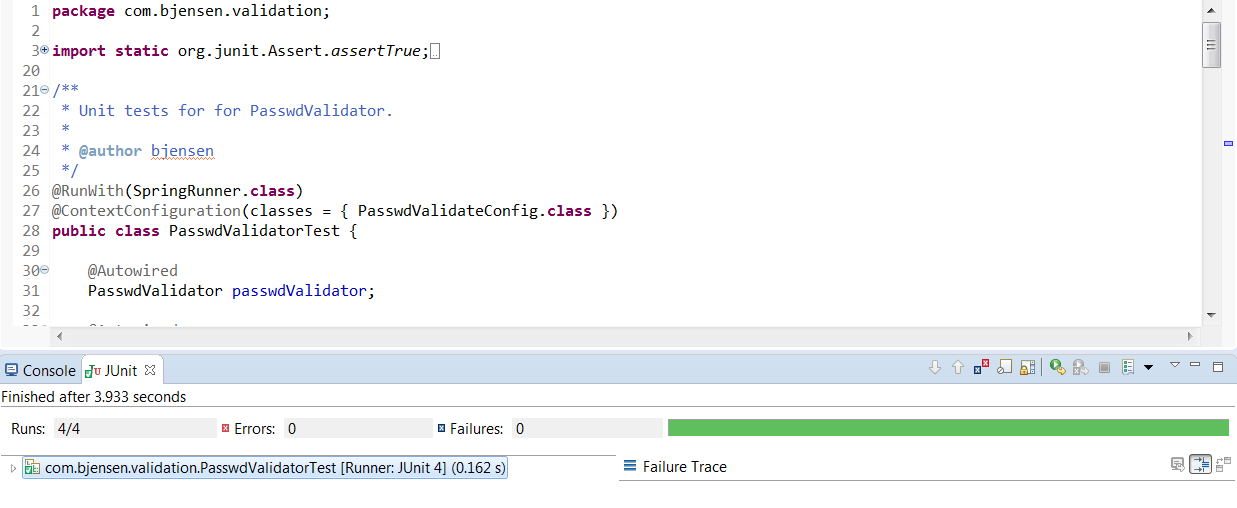
Successful Integration Test:



PasswdValidateController Slice Test:

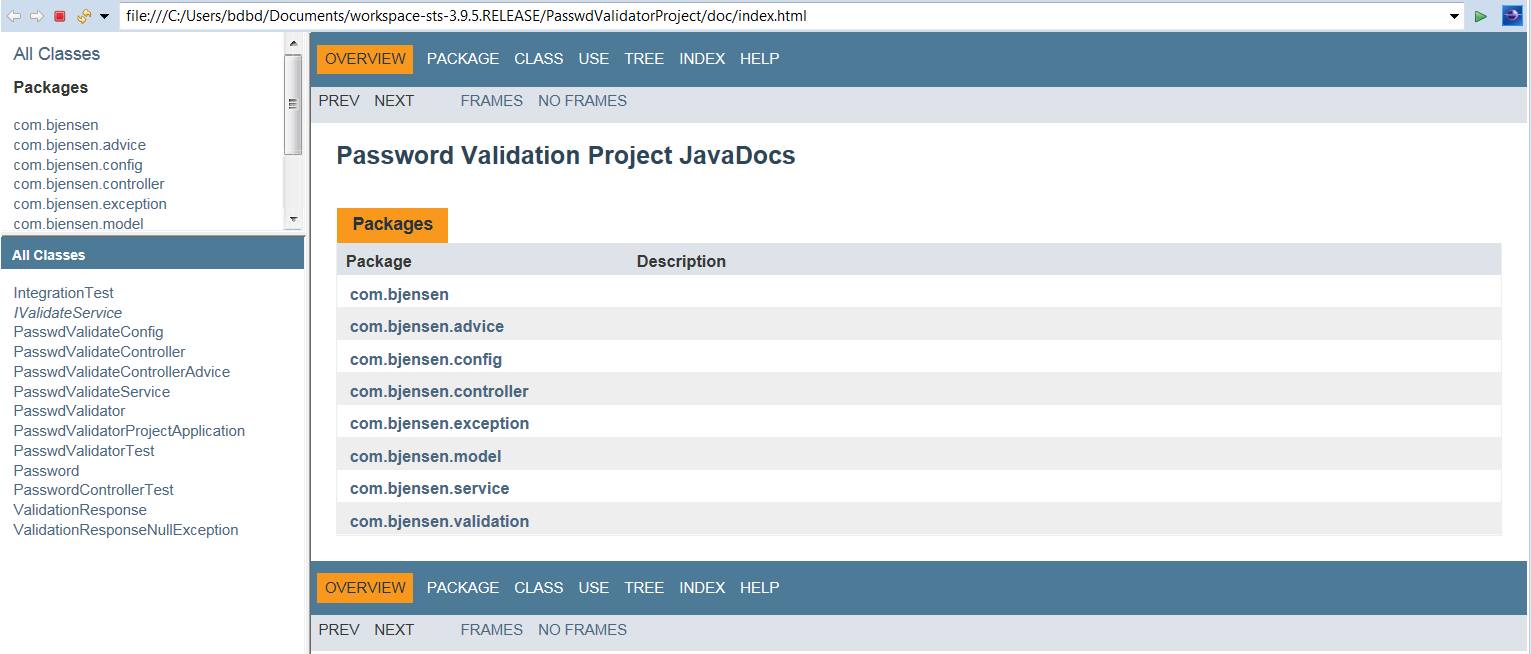


Successful PasswdValidator Unit Test:



JAVADOCS INDEX (openwith—web browser in IDE)

Javadocs in /docs folder:



PasswdValidateService API as an example:

