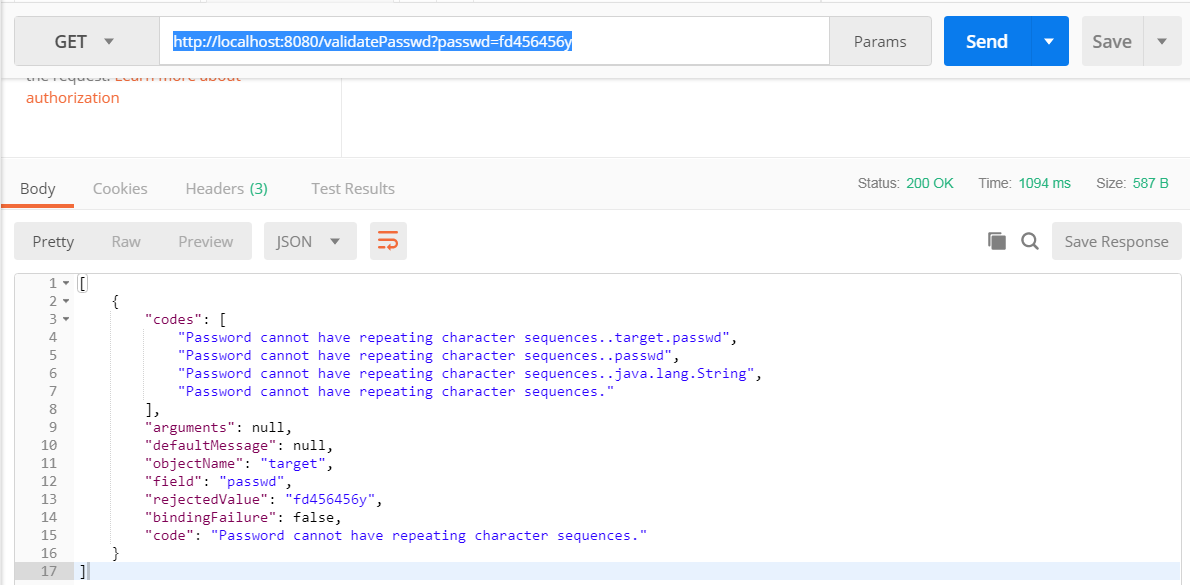
Considerations for additional requirements and modifications:

* Add logging—Spring logging, log4j, Splunk for logs analysis etc.
* If MVC, return error via modelview object to form field view.
* Enhanced Exception handling.
* REST docs—perhaps RAML, Swagger etc.
* Using GET as per these specific requirements but would use other http verbs as needed for business logic (i.e. persisting passwd history etc).
* Project built using proper package structure as per requirements. Would include data tier and view tier packages if MVC or other modified respectively as needed.
* Used STS as IDE (“Springified Eclipse”) and Spring Boot for development however deployment to lower and upper enterprise environments would be to Tomcat, Weblogic, AWS, and even implemented via AWS Lambda and API Gateway as required.
* Built as a jar for Boot however would be modified to war for enterprise deployment with two simple changes to the pom and perhaps would have begun using a web project template instead to include web descriptors, views, and other web artifacts.
* Refactor validation rules within PasswdValidator from if-else statements into separate methods or a separate PasswdValidationRules class for scale, and so this Validator could be used to validate other types of objects. This implementation evaluates rule violations using Spring BindingResult objects produced by DataBinder after rule evaluations.
* Project was begun from TDD--provide more robust test cases, especially if item above is considered for ideal coverage.
* Implement a ValidationResults object rather than to std out.
* Passwd input via JSON rather than just request param.
* Consider base Validator class to implement other types of Validators.
* Code cleanliness: all code was run through format and all unnecessary imports cleaned for ideal readability.

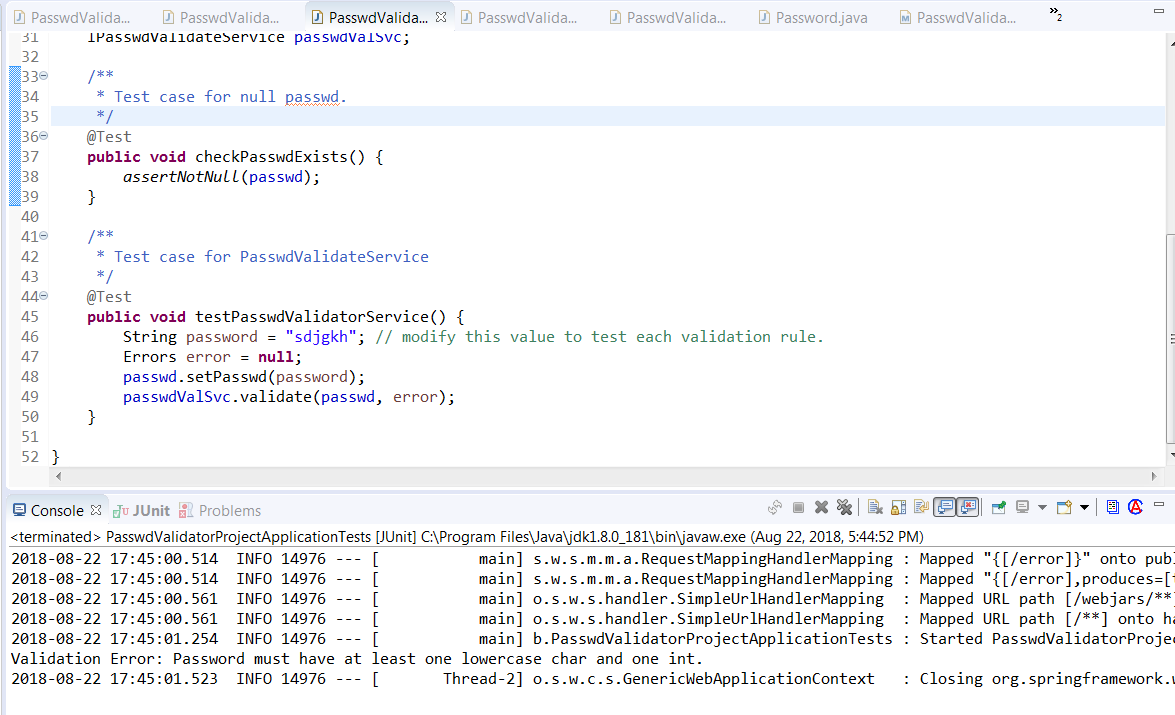
Notes:

* To examine and assess project simply import into IDE with Spring support or STS.
* Javadocs are located in /docs folder.
* Testing during development was done via spring boot wrapping junit. To test each rule during dev, change password value in testPasswdValidatorService test case and run-as JUint test. Spring Boot compiles and tests from there. **See example pic below**.
* Post dev testing done using postman. Execute PasswdValidatorProjectApplication class within IDE and send requests. **See example pic below**. Example payload:
  + <http://localhost:8080/validatePasswd?passwd=fd456456y> – to test for repeating char sequence.

Postman test example:



Junit test example during dev:



Javadocs in /docs folder:

