**JUnit Testing Documentation**

**REST Controller JUnit Testing and Mockito Set-Up**

The REST controller test was set up using a stand-alone mock up implementation instead of a so-called “integrated” version. An integrated test would have caused the test to load up with the Application context (when the AES application boots up). The stand-up implementation was chosen because it was easier to integrate the test into an existing project without changing any of the existing code base. This maintains loose coupling between the unit testing and the rest of the program since it uses Mocks to imitate the behavior of the service class that the controller calls.

The CandidateController was chosen for the Junit Mocking test due to its simplicity (it only has one method) and provided a way to experiment with different types of unit testing styles to see which kind would best work with the existing project. The following is a walk-through of what was implemented and why.

**AESJDBCImplUnitTest Junit testing**

The AESJDBCImpl unit tests are implemented to see if you have questions in our database for our assessments. If we ever have zero questions then our unit tests shall fail. We have test for each of our 12 categories and 4 types of questions.

**Security/Validation Testing**

Unit tests for

* com\revature\aes\config\CustomAuthenticationProvider.java
* com\revature\aes\service\CustomUserDetailsService.java

CustomAuthenticationProviderTest extends AbstractJUnit4SpringContextTests and creates a dummy user, passes it into a TestingAuthenticationToken instance, and passes it into a SecurityContextHolder to retreve authentication status. If successful, an assertThat call on isAuthenticated() should return true.

CustomUserDetailsServiceTest creates a dummy username, password, and role. If all are valid, then the Principal user that is created should also be valid.