## **1. Summary**

This portion of the proposal outlines the testing plan for Scribble. Proper testing is critical for the performance and functionality of our application and will instill confidence in our end-users that they can depend on our application to satisfy their needs. Through applying test-driven development principles and implementing appropriate tests, we will ensure the security, reliability, and expectations of our users.

## **2. Scope**

### **2.1 New functions and features**

All new functions and features that are public-facing immediately or indirectly are within the scope of testing.

### **2.2 Performance testing**

Performance testing on all supported environments to ensure responsiveness, scalability, and quality of service are within the scope of testing.

### **2.3 Security testing**

Security testing on all public and indirectly public modules to ensure data protection is within the scope of testing.

### **2.4 Compatibility testing**

Compatibility testing across supported environments on all public and indirectly public modules is within the scope of testing.

### **2.5 Usability testing**

Usability testing on all public and indirect public modules to ensure an innovative and seamless interface is within the scope of testing.

## **3. Schedule**

### **3.1. Unit Testing**

Ongoing throughout the software lifecycle.

### **3.2. Component Testing**

Parallel to unit testing.

### **3.3. System Testing**

Conducted after integration of all modules before any major release.

### **3.4. Regression Testing**

Conducted after each update and change to the software.

### **3.5. Performance Testing**

Conducted after the final stages of development and before major releases.

### **3.6. Integration Testing**

Concurrent with performance testing.

### **3.7. Security Testing**

Ongoing throughout the software lifecycle.

### **3.8. Beta Testing**

Scheduled before major releases for selected candidates.

## **4. Development Testing**

Various tests will be conducted during the development process to identify bugs and defects early. The testing will be carried out in three stages: unit testing, component testing, and system testing.

### **4.1. Unit Testing**

Guideline-based unit testing practices will be followed for formatting and selecting unit tests. This will streamline the testing process while ensuring that critical aspects of the code are covered. Unit tests will be written for individual components such as message sending, receiving, user authentication, and contact management.

### **4.2. Component Testing**

Component testing will focus on testing individual modules or components of the application such as user interface components, database interactions, and message encryption/decryption.

### **4.3. System Testing**

System testing will verify the integrated system against our requirements, allowing functional, performance, and compatibility testing across several of the defined environments specified.

### **4.4. Test Driven Development**

Test-Driven Development will be utilized such that tests are written before any code is implemented ensuring that the code meets requirements from the start.

## **5. Release Testing**

Testing will be conducted before any release of the application to end users.

### **5.1. Regression Testing**

Regression testing will be used to ensure that new updates and changes remain backward compatible and do not affect the existing features of the application unless it is imperative for further development.

### **5.2. Performance Testing**

Performance testing will assess the responsiveness and scalability of the application for supported environments and varying load conditions. Performance testing will include stress testing, load testing, and endurance testing.

### **5.3. Integration Testing**

Integration testing will help verify and collate the different modules of the application to ensure a seamless and intuitive system.

### **5.4. Security Testing**

Security Testing will be performed to identify vulnerabilities and ensure that the application is resistant to unauthorized access, data breaches, and other security threats to the best of our ability and in compliance with non-functional requirements SE.2 and SE.3.

## **6. User Testing**

User testing will involve real end-users to evaluate and provide feedback on the usability, functionality, and experience of the application.

### **6.1 Beta Testing Service**

The application will be released to a limited group of beta testers who will provide feedback before major releases. This group will help identify usability issues, bugs, and areas for improvement that need attention before public release.

## **7. Conclusion**

The following test plan outlines the approach for Scribble, which covers the development, release, and user testing phases. All of these phases contribute towards creating a reliable application that our users can trust. This aligns with our mission of providing high-quality, reliable, and secure instant communication to our users.

## 

## **Code for the Unit**

*(Full source files are available on* [*GitHub*](https://github.com/nick-donovan-utd/3345-TheSKidSociety/blob/main/Deliverable%202/src/main/java/dev/scribble/controller/UserController.java)*)*

package scribble.controller;

import scribble.exception.UserNotFoundException;

import scribble.dto.NewUserRequest;

import scribble.models.User;

import scribble.models.UserLogin;

import scribble.repository.UserLoginRepository;

import scribble.repository.UserRepository;

import jakarta.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

@RestController

public class UserController {

@Autowired

UserRepository userRepository;

@Autowired

UserLoginRepository userLoginRepository;

*/\*\**

*\* Creates a new user with the provided details.*

*\**

*\* @param newUserRequest The request object containing details for the new user.*

*\* @return The newly created user entity.*

*\*/*

@PostMapping("/user/add")

public User createUser(@Valid @RequestBody NewUserRequest newUserRequest) {

// String firstName, String lastName, String email

User user = new User(newUserRequest.getFirstName(), newUserRequest.getLastName(), newUserRequest.getEmail());

// Save login information

UserLogin login = new UserLogin(user, newUserRequest.getUsername(), newUserRequest.getPasswordHash());

userRepository.save(user);

userLoginRepository.save(login);

return user;

}

}

## **Test for the Unit**

*(Full source files are available on* [*GitHub*](https://github.com/nick-donovan-utd/3345-TheSKidSociety/blob/main/Deliverable%202/src/test/java/dev/scribble/controller/UserControllerTest.java)*)*

The following tests use MockMvc [3]. We are testing the createUser function to ensure that the user data is properly added to the server's database and returned.

// Imports removed for space, available on GitHub

@SpringBootTest

@AutoConfigureMockMvc

class UserControllerTest {

@Autowired

private MockMvc mockMvc;

private final String expectedFirst = "John";

private final String expectedLast = "Smith";

// Generate random email and username to prevent conflicts caused by uniqueness constraints.

private final String randomEmail = java.util.UUID.*randomUUID*() + "@test.com";

private final String randomUsername = java.util.UUID.*randomUUID*().toString();

@Test

void createUser() {

// Define a JSON string for the request body.

String newUserJson = """

{

"firstName": "%s",

"lastName": "%s",

"email": "%s",

"username": "%s",

"passwordHash": "$argon2i$v=19$m=65536,t=2,p=4$c29tZXNhbHQ$FgNukdXUKj/gS6Ur+fgQ6laMZLTrvSKo"

}

""".formatted(expectedFirst, expectedLast, randomEmail, randomUsername);

// Perform a POST request to the '/user/add' endpoint using mockMvc for testing

String result = *assertDoesNotThrow*(() -> mockMvc.perform(MockMvcRequestBuilders

.*post*("/user/add")

// Set the 'Content-Type' header to json and send the data

.contentType(MediaType.*APPLICATION\_JSON*)

.content(newUserJson))

.andReturn().getResponse().getContentAsString());

// Test the results

*assertDoesNotThrow*(() -> testResponseHelper(result));

}

private void testResponseHelper(String response) throws JSONException {

// Parse and test the data

JSONObject r = new JSONObject(response);

*assertEquals*(expectedFirst, r.getString("firstName"), "TESTING: user creation (firstName)");

*assertEquals*(expectedLast, r.getString("lastName"), "TESTING: user creation (firstName)");

*assertEquals*(randomEmail, r.getString("email"), "TESTING: user creation (firstName)");

}

}

The test result  
