### CRUD Application: Test Plan

#### Introduction

The application is a simple CRUD application where a user can create, read, update and delete a note. A note can have a title, content and a created date.

This document will describe the test plan for the application described above.

#### Tech stack

* Maven/Java project
* Postman for API testing
* GitHub
* Javalin

#### Test Objectives

Since this is a CRUD application that relies on database and CRUD operations, the main testing objectives will be to test if a user can use the application for its purpose.

**Functional testing**

* Test if the user can create a note with the correct input
* Tests to see if the user can access a note if a correct and valid ID is provided
* Test if the user can delete a note if a correct and valid ID is provided
* Test if the user can update a note if a correct and valid ID is provided

**API testing**

Much of the functional testing is to test the functionality of the application and if the user can communicate with the database through API. Therefor we also need to test if:

* The API returns correct HTTP status codes for different scenarios
* Confirm data validations (if the user misses something)
* Test error handling and edge cases

**Database testing:**

* Do at least one test with an in-memory SQLite database to ensure SQL functionality without API interference.

**Non-functional testing: Security**

* Only with a valid and existing username and password provided in Postman before note is created
* Valid and existing username and password provided in Postman before being allowed to update note. Username should also be owner of note
* Only owner can delete

#### Scope

* Test all functionality of CRUD operations
* Test all API endpoints and that we get a correct HTTP status code back
* Security check – check that existing user credentials is provided when creating a note. A note can only be created when it has an owner (a valid user ID)
* JMeter test plan and coverage report

#### Resources

We need a couple of frameworks for this plan:

* Maven (dependency management, etc.)
* Javalin (API work)
* Jacoco (Coverage report)
* SQLite (to test queries against in memory test database)
* JMeter for a test plan
* Unittest4 and 5
* Postman for testing

#### Risk

The biggest risk is that time is limited, and I am only one person to do all testing, so if anything goes wrong or any challenges happen, then it might be difficult to do it all in time.

### Reflection

The application is relatively simple, so there was not a lot of complex business logic to test. I experimented with both Postman and writing API tests in the APITest.java file. I found Postman very convenient for integration testing because of its clear interface and ease of use. The drawback, of course, is that it doesn’t integrate easily into a continuous integration pipeline. The test methods I wrote in the test folder, however, would work well in such a pipeline.

Using Jacoco gave me a good overview of code coverage. I learned from this experience that it’s better to write functions in smaller, testable pieces first, because the more code you write, the more there is to test.

I also realized that I had set my expectations too high for the test plan. For example, I wanted to implement security on most endpoints, but I only had time to secure one. This is something I will focus on next time.

JMeter worked well for heavy load testing, though it might not be very relevant for a simple CRUD application. I did face several setup challenges in the GUI, so I didn’t get to do as much as I wanted.

Finally, I also did some testing with SQLite to increase coverage, but I am not sure it contributed that much.