Issue Tracker – Understanding the brief

Table of Contents

[Technology Stack 1](#_Toc495599150)

[Java 1.7+ 1](#_Toc495599151)

[Maven/Gradle 1](#_Toc495599152)

[JUnit 1](#_Toc495599153)

[Database 2](#_Toc495599154)

[Brief Requirements 2](#_Toc495599155)

[Support two types of users: administrator and developer (1) 2](#_Toc495599156)

[User authentication (2, 3) 2](#_Toc495599157)

[User registration 2](#_Toc495599158)

[User login 3](#_Toc495599159)

[User logout 3](#_Toc495599160)

[Models (as defined in the brief) 3](#_Toc495599161)

[Actions needed 3](#_Toc495599162)

# Technology Stack

## Java 1.7+

Agreed upon because of highest familiarity, ease of use over Python, and proficiency.

## Maven/Gradle

Currently setup a Gradle project, but this can be migrated to the Maven build system if need be.

Advantages of Maven:

* Familiarity because of its use in previous projects

Advantages of Gradle:

* Encompasses everything Maven does, plus some more functionality.

Long story short: use either, it won’t really matter.

## JUnit

To be used for TDD, unit testing, and made effective with the combined use of Mockito and Hamcrest (as needed.)

All three technologies have low learning curve, and are very powerful.

## Database

We need to decide between a SQL database or a NoSQL database. General overview provided below:

|  |  |  |
| --- | --- | --- |
|  | Pros | Cons |
| SQL | * Can look back at 351 lectures for help with constructing complex SQL queries * Prioritizes domain model early on * Can look back at 325 to see how to combine ORM with SQL databases | * Not as flexible, since domain model is difficult to modify as development continues * Not as intuitive to use, and requires decent knowledge of relations, joins and query construction |
| NoSQL | * Easy to use and learn since most NoSQL databases are essentially big lookup tables (key-value pairs) * Firebase and DynamoDB are pretty cool and work really well with Java | * No formal introduction to NoSQL at university (not including triple stores of-course) |

Long story short: we need to decide which type of database (NoSQL vs SQL) early on, and then we need to decide which product to use based on that type (MongoDB vs Firebase vs Hibernate, etc.)

# Brief Requirements

## Support two types of users: administrator and developer (1)

An obvious way to design this would be to use inheritance; have a User class and have two subclasses, namely Administrator and Developer.

While this is definitely a valid and viable design, we have to remember how we’re going to persist these instances in our database. If we’re using SQL, might have to flick through 325 again to see how ORM handles inheritance.

Alternative option is just to have one class for each type of user, and no common superclass.

## User authentication (2, 3)

Not sure how extensive or secure they want us to implement this, but seeing as its just two weeks left, we could go simple first and build on it later.

### User registration

1. Take in a username/email and password
2. Check if the username/email doesn’t belong to another user
3. If it does, prompt a corresponding message
4. If it doesn’t, check the validity of the password (min length, etc.)
5. Pipe the password through a hashing function (SHA2?)
6. Create a new User instance with the username/email and hash
7. Persist the instance to our DB
8. Prompt a corresponding SUCCESS message
9. Sign in the User using the same credentials, as below.

### User login

1. Take in a username/email and password
2. Check if the username/email belongs to a user
3. If it doesn’t, prompt a corresponding message
4. If it does, pipe the password through the same hashing function and check they match
5. Update the status of the User instance to LOGGED\_IN
6. Persist the instance to our DB
7. Return an auto-generated token to the User

### User logout

1. Take in a token
2. Validate the token belongs to a LOGGED\_IN User
3. Update the status of the User instance to LOGGED\_OUT
4. Persist the instance to our DB
5. Prompt a corresponding SUCCESS message

## Models (as defined in the brief)

## Actions needed

* Sort issues. Sort categories by priority (based on the number of forum posts related to the issue).
* An administrator can add a forum post to a cluster.
* An administrator can remove forum post from a cluster.
* An administrator can delete a cluster – all related forum posts should be put into single question clusters.
* An administrator can assign an issue to a developer
* An administrator can unassign a developer from an issue
* A developer can mark an issue as resolved