Software Architecture Document

By GottMusIg

Version 1.1

Revision History

Date	Version	Description	Author
1/12/16	1.0	First version	Leon Gottschick
2/12/16	1.1	Added MVC UML	Leon Gottschick
3/07/17	1.1	Revision	Christoph Emig

Revision History 2

Introduction 4

Purpose 4

Scope 4

Definitions, Acronyms, and Abbreviations 4

References 4

Overview 4

Architectural Representation 4

Architectural Goals and Constraints 5

Use-Case View 6

Overview 6

Logical View 6

Process View 7

Deployment View 7

Implementation View 9

Data View 10

Size and Performance 11

Quality 11

Introduction

- - - - X

Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

Scope

This document describes the architecture of the DPS Difference use case

Definitions, Acronyms, and Abbreviations

n/a

References

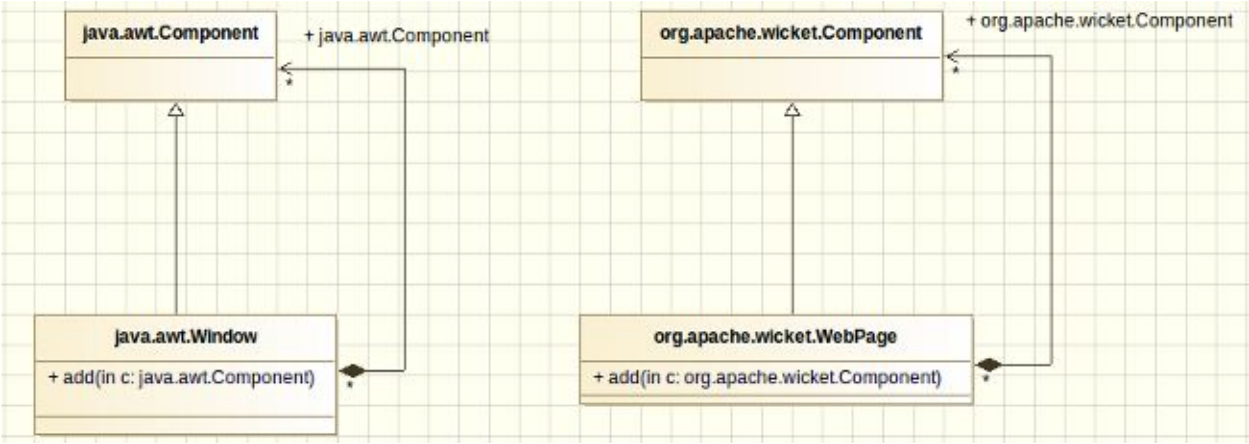
All relevant documents can be found on our dev site.

Overview

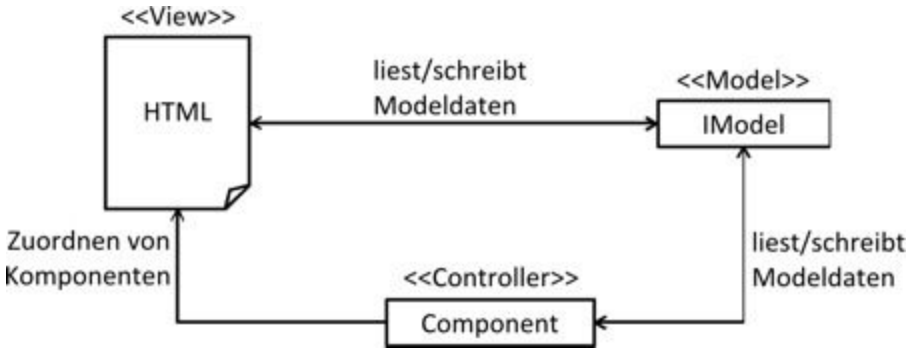
Architectural Representation

Apache Wicket is a component-based web-framework which separates the logic from the view. Every component is able to store its data in a data model.

It resembles SWING/AWT.



MVC represented in Wicket:



Architectural Goals and Constraints

Wicket is a powerful framework that often is employed in large software projects.

Wicket is very powerful when it comes to meet the complex requirements of a Web application. Business logic can be fully programmed in Java, without worrying about the special features of a Web application. The status of processing within a session is run server-side by wicket, because of that the developer of a Wicket-application must specifically solve the resource problem of many concurrent users. In

contrast to this many JavaScript-based Web frameworks keep the state on the client side and only if necessary reload data from the server.

Because JavaScript or even CSS frameworks can easily be incorporated in wicket, wicket represents a flexible Web development platform.

Wicket has a steep learning curve, which leads to a difficult entry.

Use-Case View

Look at our SRS for further information.

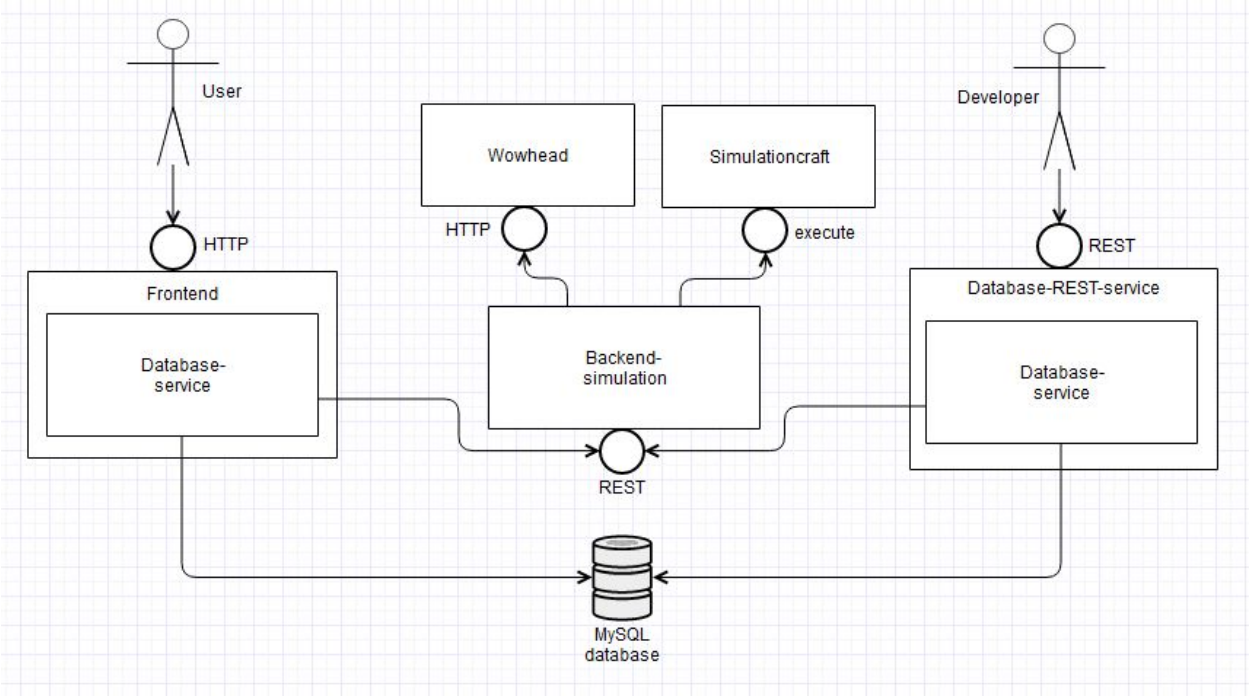
Overview

Look at our Use Case Specifications for further information.

Logical View

The project GottMusIg is made of four main components. The frontend which depends on the database-service, the backend and the database-REST-service. Normal users can interact with the frontend via HTTP-Requests. The frontend calls the database-service which manages the accesses to the database where accounts, items and simulations are stored. The database-service is also able to request the backend via a REST-Interface to start simulations. Simulations are executed by the backend with Simulationcraft. Additionally the backend communicates with Wowhead to get class specific item-information which will be needed to compare and simulate different items to create a ranking which will make it easier for users to get the best fitting items for them. The fourth component is the database-rest-service (not implemented yet).

It can be used by other developers over a REST-Interface to build their own frontends and let them get access to the database-service. For example to start own simulations.



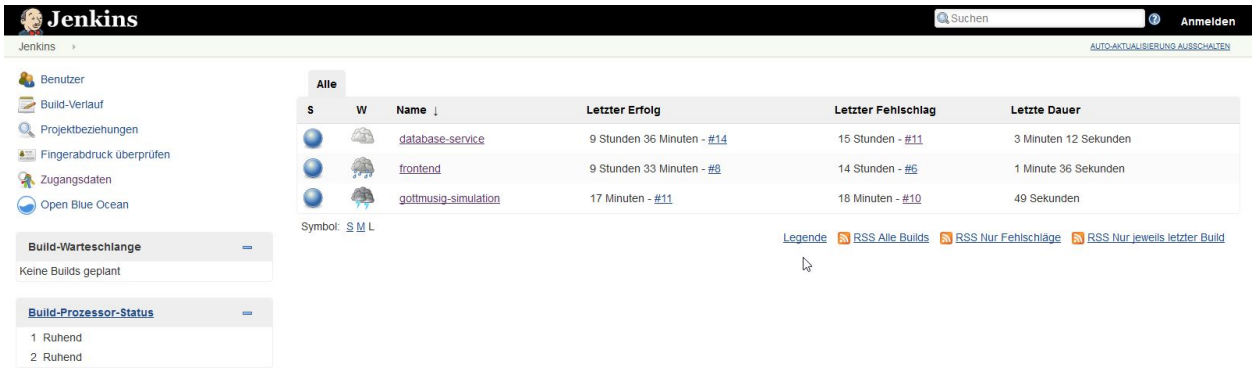
Process View

- - - - X

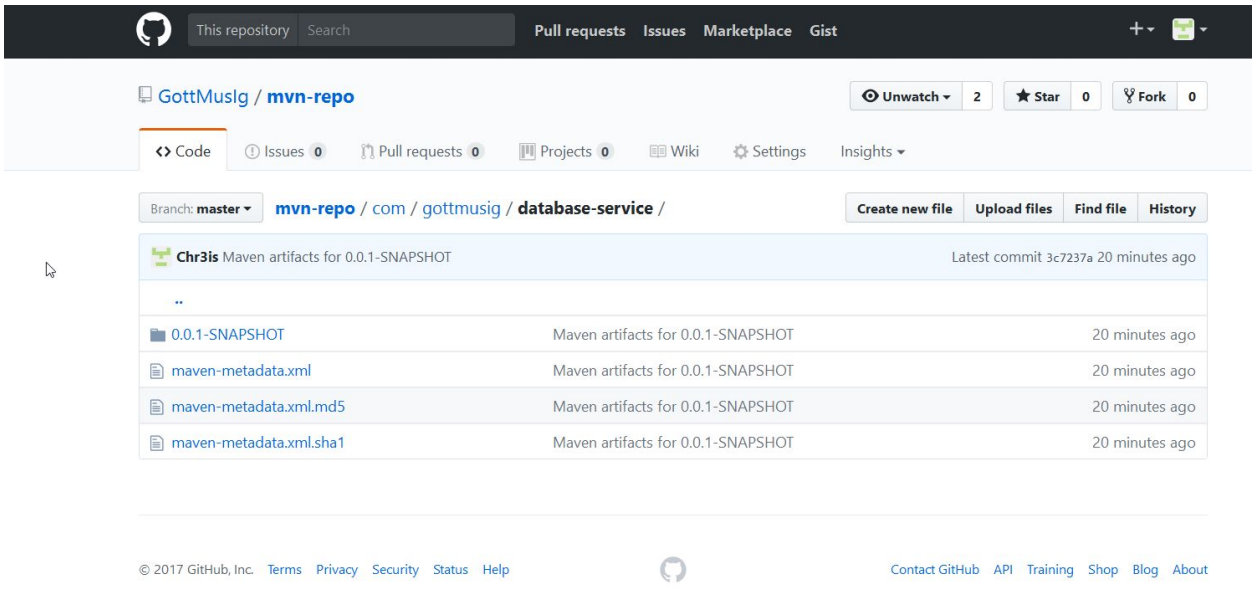
n/a

Deployment View

Our projects are built and tested with Jenkins. Every project will be tested one time per day or when something changed on our master branches. For example when something was merged. During the build metrics are generated which will be automatically uploaded to Sonar (https://sonarcloud.io/organizations/gottmusig-github/projects).



In the logical view the database-service was already mentioned. It's a separate project which is included by the frontend and the database-REST-service through a maven dependency. To handle the dependency we created a maven repository on GitHub.



Every time the database-service was successfully build by jenkins the artefact will be automatically deployed to the GitHub-repository.

```
Maven RedeployPublisher use remote  maven settings from : /var/jenkins_home/tools/hudson.tasks.Maven_MavenInstallation/M3/conf/settings.xml
using global settings config with name DeployToGithub
Replacing all maven server entries not found in credentials list is true
Maven RedeployPublisher use remote  maven global settings from : /tmp/global-settings7413951985368688469.xml
[INFO] Deployment in file:///var/jenkins_home/workspace/database-service/target/mvn-repo (id=internal.repo,uniqueVersion=true)
Deploying the main artifact database-service-0.0.1-20170703.005425-1.jar
Downloading: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/0.0.1-SNAPSHOT/maven-metadata.xml
Downloaded: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/0.0.1-SNAPSHOT/maven-metadata.xml
(779 B at 190.2 KB/sec)
Uploading: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/0.0.1-SNAPSHOT/database-service-
0.0.1-20170703.005425-1.jar
Uploaded: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/0.0.1-SNAPSHOT/database-service-
0.0.1-20170703.005425-1.jar (190 KB at 27050.8 KB/sec)
Uploading: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/0.0.1-SNAPSHOT/database-service-
0.0.1-20170703.005425-1.pom
Uploaded: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/0.0.1-SNAPSHOT/database-service-
0.0.1-20170703.005425-1.pom (12 KB at 5773.9 KB/sec)
Downloading: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/maven-metadata.xml
Downloaded: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/maven-metadata.xml (289 B at 94.1
KB/sec)
Uploading: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/0.0.1-SNAPSHOT/maven-metadata.xml
Uploaded: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/0.0.1-SNAPSHOT/maven-metadata.xml (779
B at 253.6 KB/sec)
Uploading: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/maven-metadata.xml
Uploaded: file:///var/jenkins_home/workspace/database-service/target/mvn-repo/com/gottmusig/database-service/maven-metadata.xml (289 B at 141.1
KB/sec)
[INFO] Deployment done in 0.38 sec
Warning: you have no plugins providing access control for builds, so falling back to legacy behavior of permitting any downstream builds to be
triggered
Triggering a new build of frontend
Finished: SUCCESS
```

With this solution it isn’t necessary anymore to checkout and build the database-service manually. With every change on the master-branch the depending projects (frontend and database-REST-service) will have the latest changes without doing something by their own.

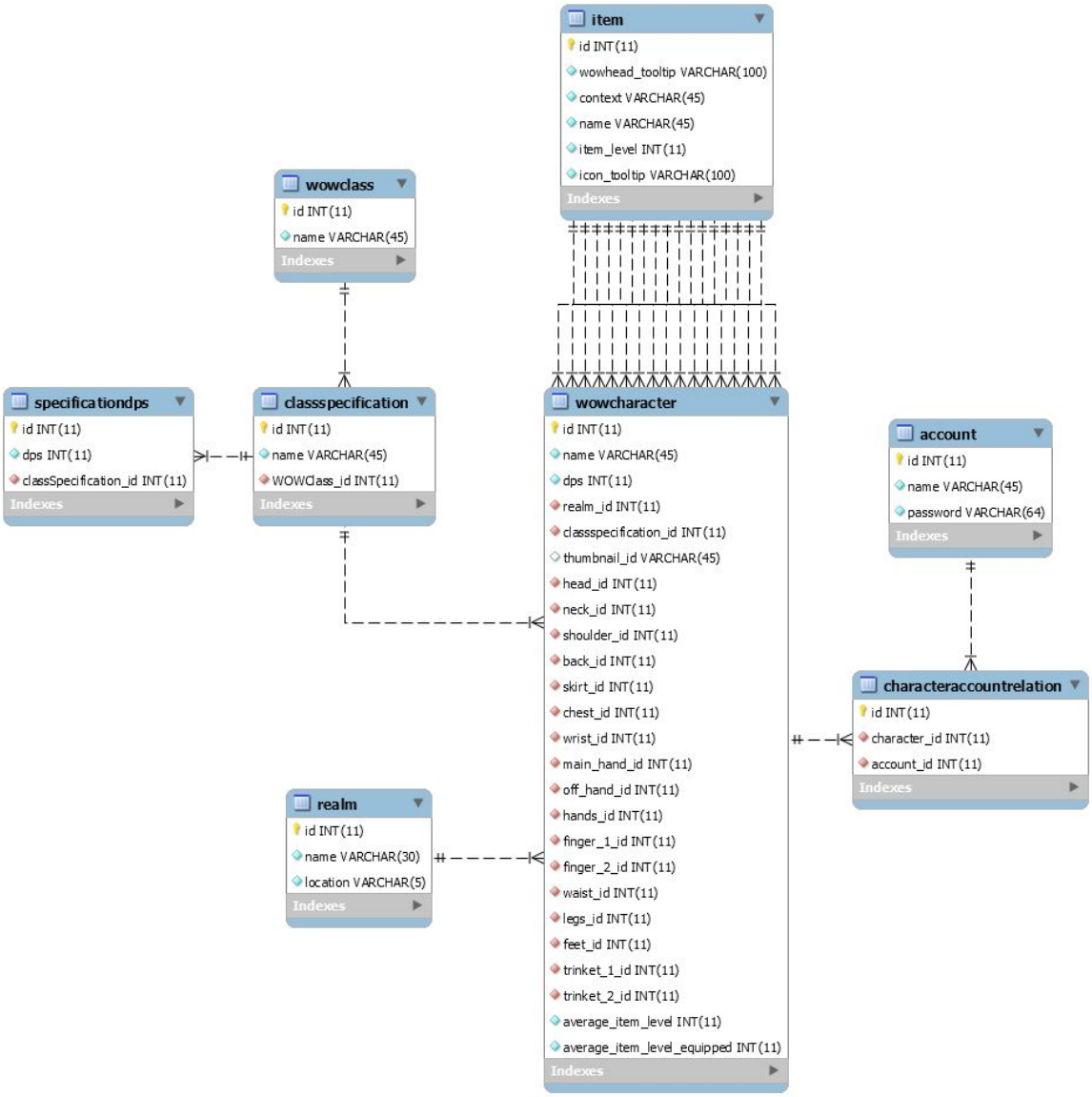
Implementation View

- - - - X

n/a

Data View

Database schema:



Size and Performance

- - - - X

n/a

Quality

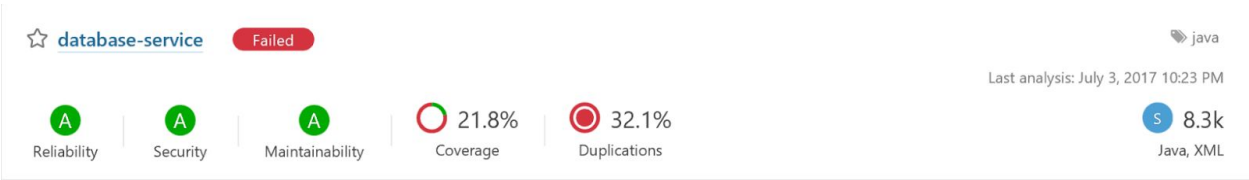
- - - - X

Metrics

The first Metric tool we use is sonarcloud.

Sonarcloud is fully integrated in our Continuous Integration process, it is part of our deployment process on our Jenkins Server.

Sonar has shown us some issues which we tried to improve and are on our way to a reliable software.



Now we’re gonna take a look at a specific Problem SonarQube made us aware of:

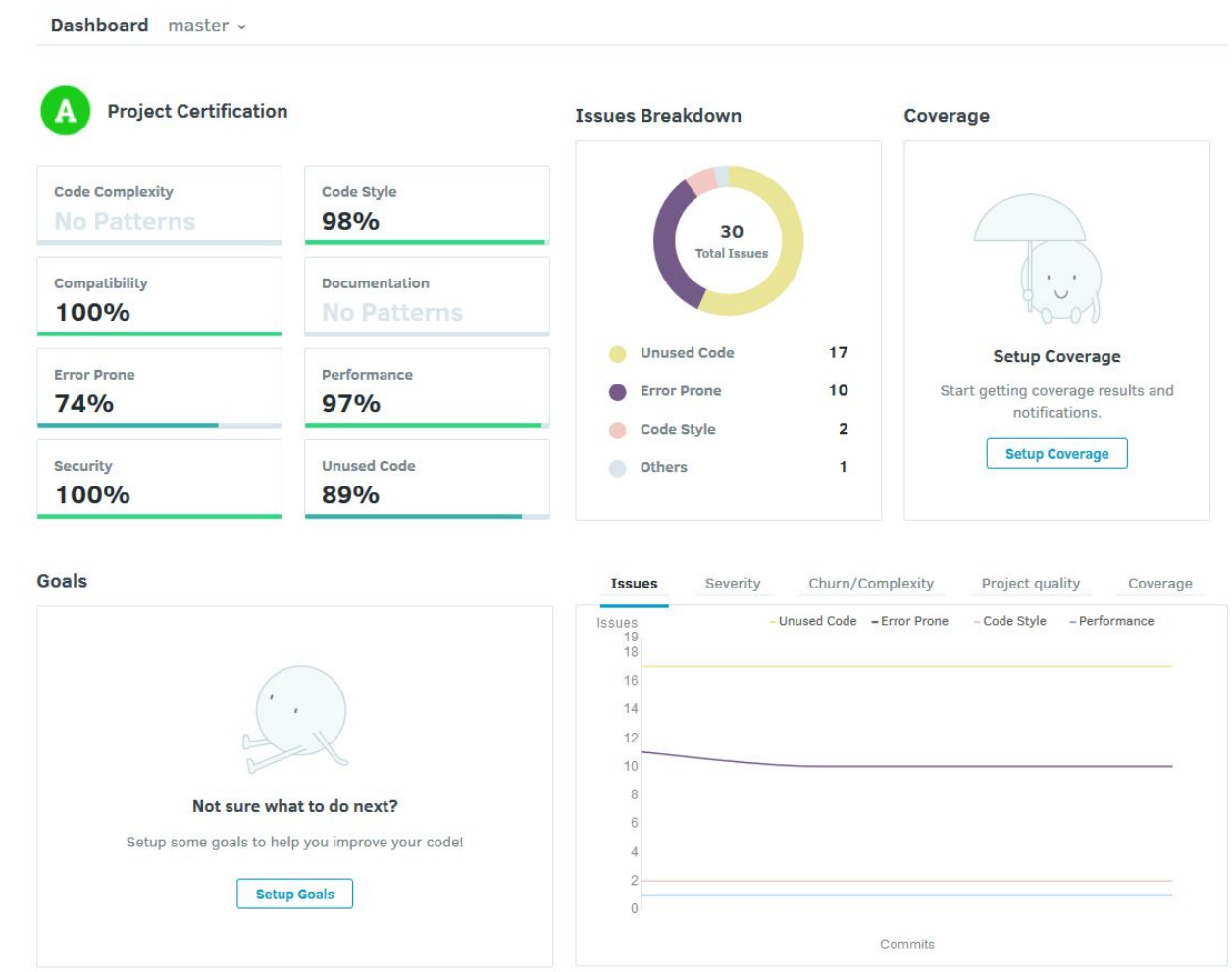
When we used the string „class“ several times, sonar suggested to make an own field for this string to avoid many repetitions.

Also sonar warns us if the classes have too many parents:

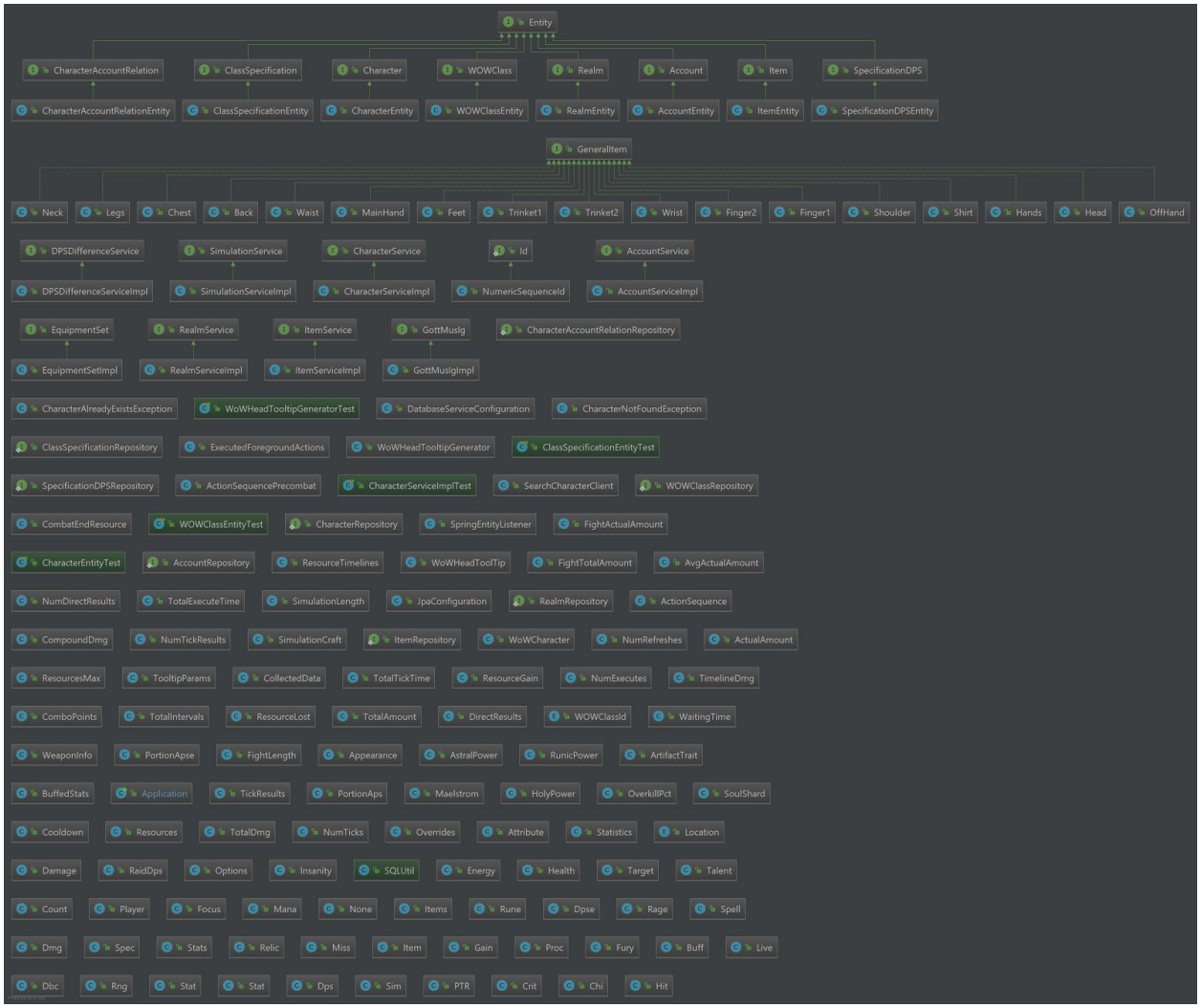
But we could not fix this problem because of the frameworks we use.

We also use [codacy] as our second metric tool. It shows us our Code Style, Security, Unused Code and alot of other stuff.

Here is a picture of our Dashboard:


Dashboard master
A
Project Certification
Code Complexity
No Patterns
Code Style
98%
Compatibility
100%
Documentation
No Patterns
Error Prone
74%
Performance
97%
Security
100%
Unused Code
89%
Issues Breakdown
30
Total Issues
Unused Code 17
Error Prone 10
Code Style 2
Others 1
Coverage
Setup Coverage
Start getting coverage results and notifications.
Setup Coverage
Goals
Not sure what to do next?
Setup some goals to help you improve your code!
Setup Goals
Issues
Severity
Churn/Complexity
Project quality
Coverage
Issues
Unused Code
Error Prone
Code Style
Performance
19
18
16
14
12
10
8
6
4
2
0
Commits

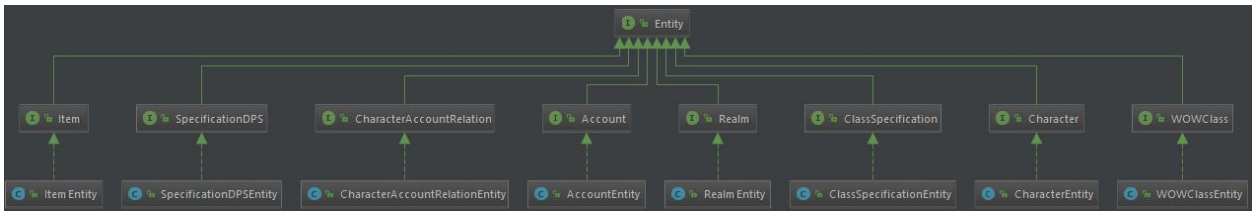
Class Diagram



Patterns

We decided to try out the dependency inversion pattern, which wants high-Level modules to depend on low-level modules and both should depend on abstractions.

The abstractions are seen in the picture below also to be discovered in the Class diagram right at the top:



Other modules of our Software use the abstraction „Entity“ to get the information from the several implementations. So other parts of our software don’t even know that there is an implementation, all they have is the interface where they can interact with the Entity below.