|  |
| --- |
| **Fontys - University of Applied Sciences - ICT** |
| Individual track project: Online inventory system |
| Individual track Design Document |

|  |
| --- |
| Martin Georgiev  Student number: 3782484  Eindhoven, Friday, January 22, 2021 |

Document Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| *Date* | *Version* | *Author* | *Comments* |
| 09-10-2020 | 1.0 | **Martin Georgiev** | Added the initial body structure and text of the design document |
| 22-11-2020 | 1.1 | **Martin Georgiev** | Adjusted the justification for the backend system and added a quality metrics section. |
| 27-11-2020 | 1.2 | **Martin Georgiev** | Added a section that showcases relevant development trends. |
| 27-11-2020 | 1.3 | **Martin Georgiev** | Added a security design decisions section where security choices are explained in detail. |
| 03-01-2021 | 1.4 | **Martin Georgiev** | New test plan section; Updated architecture models; Misc. content and styling changes |
| 13-01-2021 | 1.5 | **Martin Georgiev** | Added a CI diagram model; Added an additional Code (C4) level architecture model; Updated testing strategy and document structure. |
| 22-01-2021 | 1.6 | **Martin Georgiev** | Replaced outdated images; Updated design decision section; Updated test plan section to better reflect the testing that was conducted for both the frontend and backend of the application |

Table of contents

[1 Introduction 4](#_Toc62184976)

[1.1 Document Purpose 4](#_Toc62184977)

[1.2 Document Overview 4](#_Toc62184978)

[2 Choice of front-end language & framework 5](#_Toc62184979)

[2.1 Why React? 5](#_Toc62184980)

[2.1.1 Time and cost-efficiency 5](#_Toc62184981)

[2.1.2 Reusable components 6](#_Toc62184982)

[2.2 The disadvantages of React 6](#_Toc62184983)

[3 Choice of back-end language & framework 7](#_Toc62184984)

[4 Architecture Models 8](#_Toc62184985)

[4.1 System Context (C1) 8](#_Toc62184986)

[4.2 Containers and Technology (C2) 9](#_Toc62184987)

[4.3 Components (C3) 10](#_Toc62184988)

[4.4 Code (C4) 11](#_Toc62184989)

[4.4.1 REST API Mainframe 11](#_Toc62184990)

[4.4.2 Authentication/Authorization System 12](#_Toc62184991)

[5 Project UX Design 13](#_Toc62184992)

[6 Quality Assurance Metrics 14](#_Toc62184993)

[7 Development Trends 17](#_Toc62184994)

[8 Security Design Decisions 18](#_Toc62184995)

[9 Test plan & strategy 19](#_Toc62184996)

[10 CI/CD Workflow 20](#_Toc62184997)

# Introduction

## Document Purpose

The purpose of this document is to provide an in-depth look at the project’s design structure and choices, part of the individual track. The aim is to provide a concise and well-structured document with more information on the project’s choice of front-end and back-end languages and frameworks, the architecture models that define the project’s scope and a simple UX design that should provide a clearer picture of the final product.

## Document Overview

This document will go through all sections, considered essential to all partaking stakeholders (in this case – teacher/mentor and student). Namely, these sections are:

* **Choice of front-end language/framework** – More information on my choice for a front-end language and framework/library with justification as to why I believe that this is the best choice for this project.
* **Choice of back-end language/framework** – More information on my choice for a back-end language and the framework/library with a justification of my decision.
* **Architecture Models** – A C4 Software architecture model set of my project (including a simple UML diagram) with an explanation of the design choices made and some justification why I approached the design in that particular way.
* **UX Design** – A simple UX design, mostly consisting of wireframes with some information on my plans for the design of the final product.
* **Quality Assurance Metrics** – A collection of well-established metrics to guarantee a high-quality software product.
* **Development Trends** – An insight into the development trends within the project.
* **Security Design Decisions** – A justification of the security design decisions that had to be made throughout the development of the software application.
* **Test Plan & Strategy** – A general overview of the types of testing that are planned for implementation in the scope of this project.
* **CI/CD Workflow** – An in-depth look at the CI/CD workflow of the project.

# Choice of front-end language & framework

The final product’s front-end will be written primarily, as expected, in JavaScript. While the choice of language, in this case, is simple and, frankly, obvious, the choice of the framework was a little bit tougher. Some research has been done to figure out exactly which framework best matches the needs and requirements of this project. At the end of the day, I decided to pick **ReactJS** (also known as React).

## Why React?

There are many popular frameworks for JavaScript such as Vue.js, Angular, Ember, Backbone, etc. As of recently, React has gained a lot of traction and at the moment of writing, is the most popular one among them. And while React is not a framework, but a library, it still an extremely powerful tool that completely changes the basic website structure. Instead of a communication link between the HTML code and the JavaScript code using the DOM (Document Object Model), a JavaScript-based application takes the main role of the website as it generates HTML code and links all components accordingly. The following points will provide more information on the advantages of this structure:

### Time and cost-efficiency

A normal website uses the Document Object Model (DOM) – a programming interface for HTML and XML documents that connect the document structure, style and content together. Unfortunately, the Document Object Model (DOM) is slow and inefficient as it has to recalculate all CSS components, recreate the whole website layout, and essentially, remake the whole web page every time the DOM is changed. Facebook developed React to combat that inefficiency by using a virtual DOM. Unlike other frameworks that work a real DOM, ReactJS uses a copy of it that is also known as a virtual DOM. When changes are made to the virtual DOM, instead of reloading the whole page, only the changed bits of information get updated. This drastically increases the speed and cost-efficiency of the website.

### Reusable components

Similarly to Laravel, a PHP framework that was used as part of a project during semester 2, ReactJS offers the so-called components. With components, one can split the user interface (UI) into smaller, independent, and most importantly, reusable pieces of code. With a thriving community behind this popular library, many free lightweight components are available on the web. The advantage of using a popular library is the abundance of resources online that one can learn from and potentially improve upon.

## The disadvantages of React

While ReactJS has many pros, it is also important to mention some of the cons that I have taken into consideration when picking this framework/library:

* **The high pace of development** – One of the unfortunate facts of new popular framework/libraries is that they are in a constant state of flux – new features and changes are introduced regularly and old features quickly become deprecated. This is something that some developers are not comfortable with as it requires one to quickly adapt and follow its development.
* **Steep learning curve** – React is a new library (released as open-source in 2013) and pretty large. The documentation is still lacking in some sections and not polished. Without access to comprehensive official documentation, it becomes difficult to learn by yourself without depending on the available online resources – videos, community-based tutorials, etc.

# Choice of back-end language & framework

When it comes to the backend language and framework, the final decision was difficult. The project’s RESTful API web service was built in Java using the Jersey framework, which is based on JAX-RS. While I have strong opinions about Java as a programming language and was uncertain whether I should continue development using this language, I have decided to stick with it for the duration of the project. Here are the following benefits of using this programming language:

* **Object-oriented programming language** – Most of my programming experience has been exactly with Object-oriented programming (OOP) languages. Other than new syntax, it is not too much out of my comfort zone when it comes to using Java.
* **Low-security risk** – After some research, I found that Java is used on enterprise-level for its low-security risk. While Java is not a secure language by itself, its features save people from common security flaws. A security manager is also present – a security policy that limits users to its specified access rules.
* **Popularity** – Java is immensely popular as a language for network operations. As a result, there are many resources available to the public that could potentially ease development (frameworks, libraries, guides and overall online presence).

Java has many strong advantages. Nevertheless, it is also noteworthy to mention that the programming language has significant drawbacks. The following points justify why Java might not have been the best backend programming language choice:

* **Verbose and complex code** – Verbose code results in less readability. By trying to make the syntax as similar to English, the language introduces too much noise. Compared to other languages, some code pieces are much larger, which results in more points of user failure. In addition to that, it is also harder to memorize/learn Java syntax exactly due to this verbose and complex syntax structure.
* **Bad performance** – Compared to other languages, especially C and C++, it is easy to notice that it is significantly slower. While it is at similar speed levels, languages such as Python offer a marginally better syntax and a lower learning curve. Moreover, my programming experience with Java, compared to the abovementioned language is very small.

# Architecture Models

## System Context (C1)

The system context is pretty self-explanatory. The current plans are to have two types of users (two user roles): the default user with its provided functionality and management type user that further expands the default user’s functionalities with some manager-only actions.

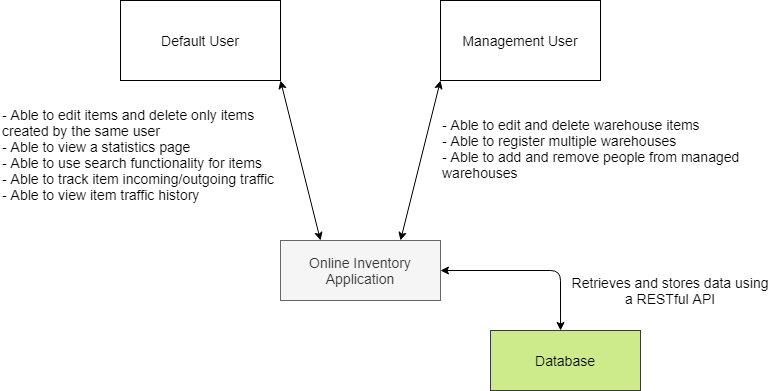


Figure 1 - System Context (C1)

## Containers and Technology (C2)

The containers and technology context provides an insight into the online inventory application’s internal structure. It is divided into two main segments: the API Application and the Web Application. The API Application will be built using Java and the Web Application will be based on JavaScript’s React library (as already mentioned in this document). Communication between these two components is done via JSON/HTTP.

Diagram

Description automatically generated

Figure 2 - Containers and technology (C2)

## Components (C3)

The component's context dives further into the API application and shows the components which might form the final product. The controller is needed for user authentication for login/register. Two controllers are required to provide and summarize information when needed.

Timeline

Description automatically generated

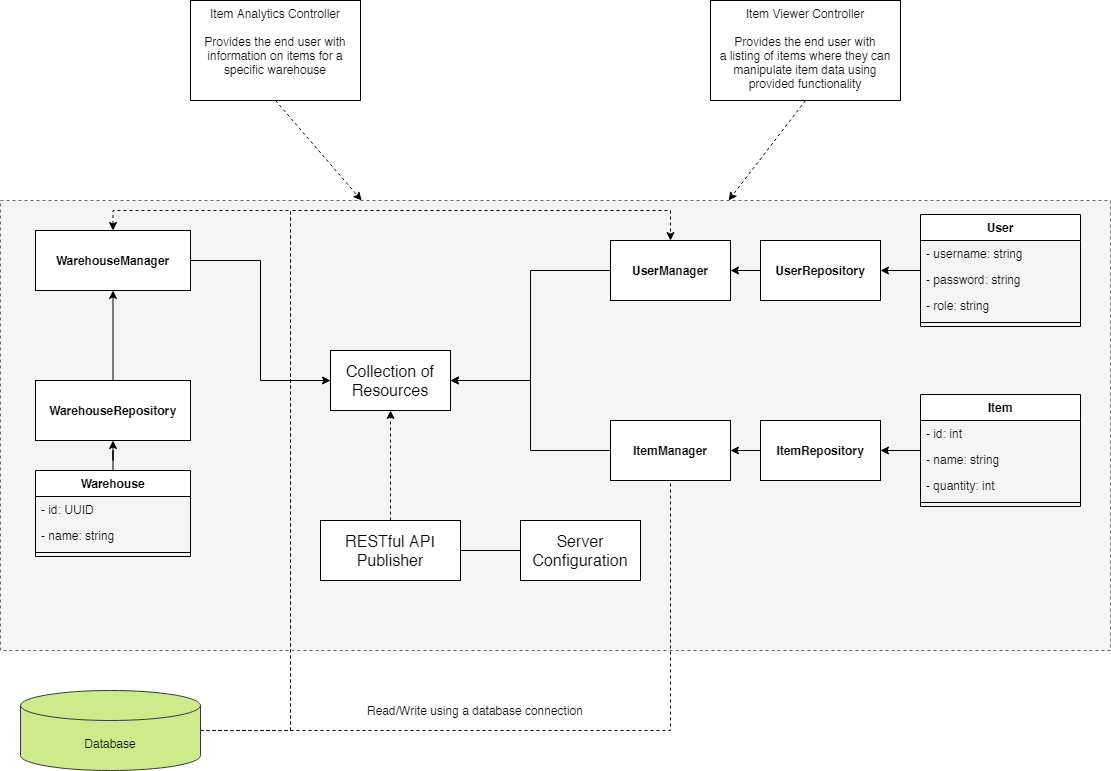
Figure 3 - Components (C3)

## Code (C4)

### REST API Mainframe

The code context of the REST API Mainframe shows how it will be structured to communicate with all other resources (endpoints) within the whole backend application.

Figure 4 - Code for REST API Mainframe (C4)



### Authentication/Authorization System

Graphical user interface

Description automatically generatedThe authentication/authorization system plays an integral part in the application as it restricts access only to users who have valid credentials and the required role to view data that is of sensitive nature.

Figure 5 - Code for Authentication & Authorization System (C4)

# Project UX Design

Graphical user interface

Description automatically generated

The main goal is to provide a simple-to-use and minimalistic interface to the end-user. The pages should provide only information that is important for the given context. The planned colour palette for the frontend web application is also going to be of a simplistic nature (e.g. white and an off-colour).

Once the user logs successfully into the web application, they will be redirected directly to the main dashboard where they can see and manipulate item data with as little actions as it is necessary to decrease use complexity. This page will feature two navigation bars – one for the main web components and one strictly for item navigation through a search system with filters.

The analytics/statistics page will highlight commonly used item data, show all incoming and outgoing item traffic and other related data through the use of different charts.

Finally, the user login/registration page should follow the same design principles – minimalism and simple use. That is why only the main form will be displayed on the screen with a stylized form.

## Design decisions

***Field:*** *Explore user requirements*

***Showroom:*** *Peer review, Benchmark testing*

***Library:*** *Available product analysis, Best good and bad practices*

***Lab:*** *Unit testing, Component testing*

***Workshop:*** *Brainstorm, Requirement prioritization*

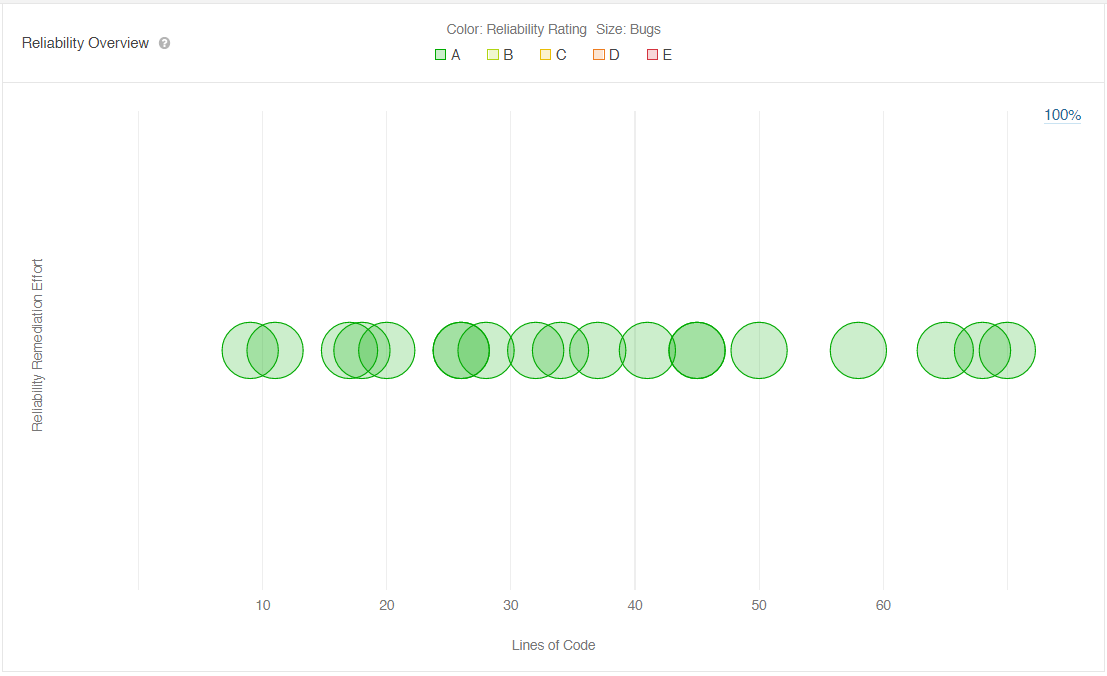
The abovementioned list highlights the methods that were used from the DOT Framework to conduct my research for the individual project. I believe that this collection works well in unison with each other. The whole process starts with the workshop and field methods – brainstorming ideas based on user requirements. That is how I concluded that I should develop an online inventory system as it fits well with the provided requirements and deliverables. Additionally, both frontend and backend development environments were picked based on popularity (available online resources), ease of use, learning curve and functionality. After a stable code base was established, I followed up with unit and component testing to assure that issues in future builds of the application are easily detected and fixed early before deployed into production. Last but not least, was the peer review and benchmark testing that was conducted to confirm that I took the right development approach.

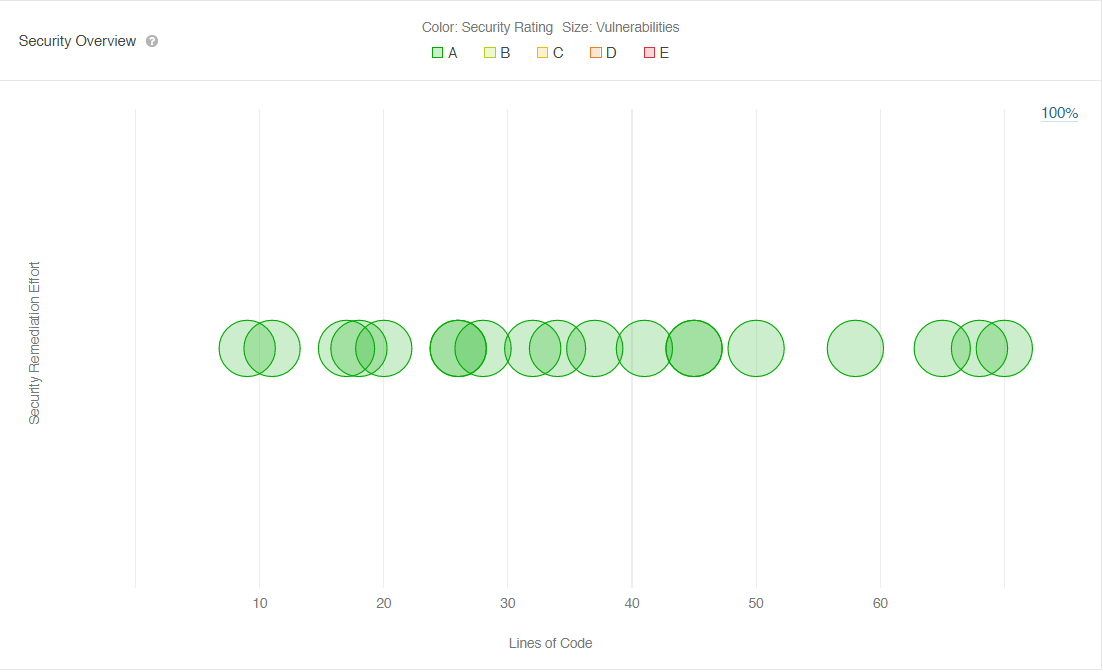
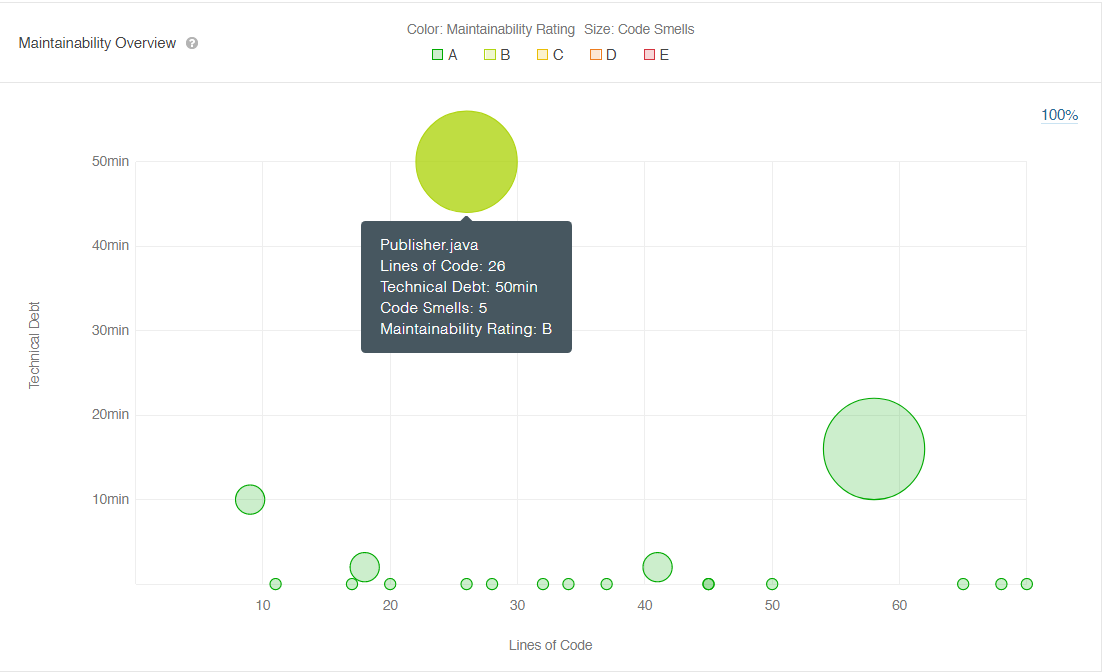
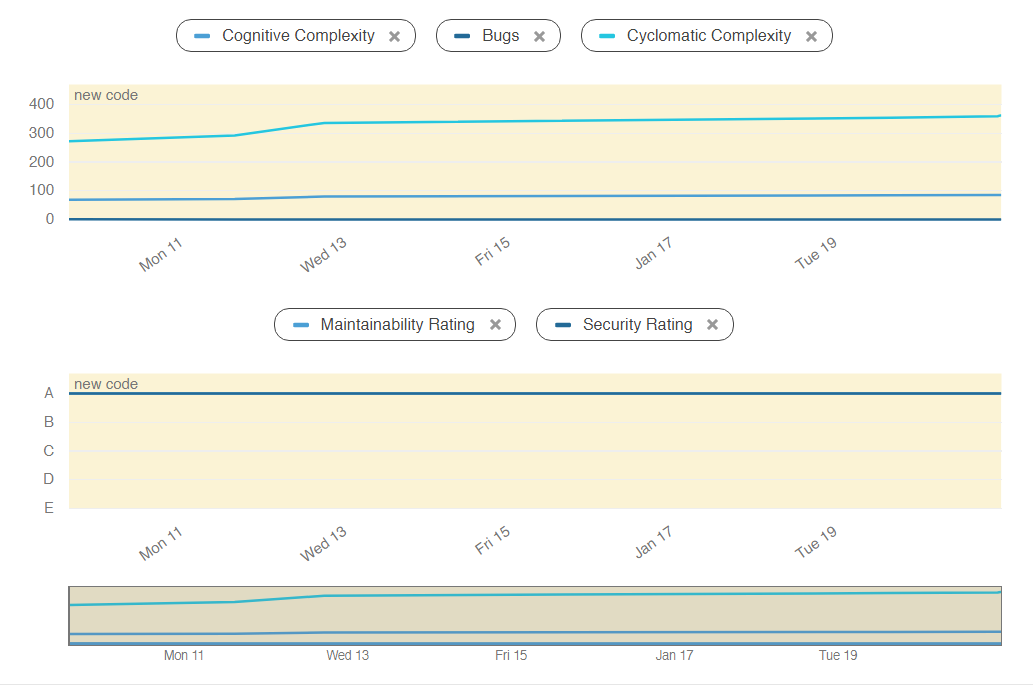
# Quality Assurance Metrics

Over the years, the software development industry has established quality assurance standards (ISO 25010) to guarantee a software product’s quality. These standards not only deal with the quality of the application’s code but can also be used to significantly improve the process efficiency of the Software Development Life Cycle and Testing process.

For the individual track assignment, I have decided to utilize a tool (SonarQube) to measure these quality assurance metrics. While they might not guarantee the final product’s completeness, they are excellent indicators that can be used to handle potential bugs that were not found by the compiler, security vulnerability, guaranteeing code maintainability and code test coverage.

* Reliability – This metric showcases that all pieces of code follow a strong reliability rating (**A**).

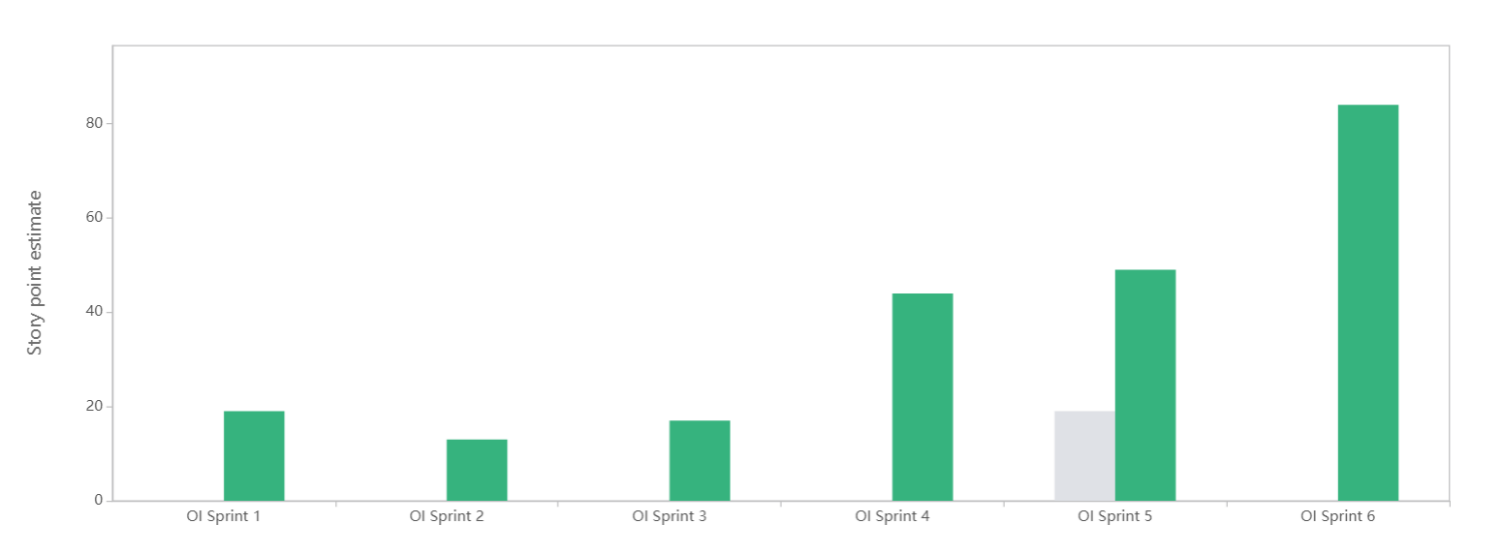


* Security – This metric showcases that no security vulnerabilities were detected in the code. (**A**)
* Maintainability – The metric displays that most of the code was graded well with a high overall rating (**A**)
* Code Complexity

After a thorough analysis of the provided metrics by SonarQube, it is reasonable to point out that although the software code’s complexity is experiencing an upward trend, the application is maintaining a strong rating in many of the quality metric fields.

# Development Trends

As the project adheres to the Scrum-based agile development methodology, different project management tools and utilities are used to simplify planning, ease development cycles and better prioritize user stories. The **Jira Software** is an excellent tool that does just that: backlog and sprint management, automatically generated reports and integration with code though different repository platforms (GitHub, Gitlab, GitBucket, etc.).

Each user story is scored between 1 and 10 (1 – easiest; 10 – most difficult) to indicate the estimated complexity, effort and size of the issue. This method is also known as “planning poker” or even “scrum poker”.

The velocity report above shows project completion based on the abovementioned user story scoring practice. The chart signifies that consistent effort is being put into each of the individual track assignment’s sprints. The difference between the latest sprint and the ones prior to it can be explained by each sprint’s time dedication. The first sprints prioritized project technologies research and documentation writing. Priorities from research to actual software development (writing code) shifts with the start of Sprint 4.

# Security Design Decisions

Security is arguably the most important aspect of a public software product that anyone could potentially use. That is why it is important that all technologies behind this product are secured accordingly to prevent information leaks and access breaches. While both the frontend and the backend of a full-stack web application require the implementation of several security measures, the backend significantly more prone to attacks due to the sensitivity of the information that is stored and processed in that part of the software. That is why just enough information should be transferred between the two sections to provide and maintain the service.

The following measures were made to mitigate any potential security risks:

* **Injection flaws** - Injection flaws allow any user to provide data containing keywords that could modify the behaviour of queries built on the database. To detect such flaws, thorough manual source code review needs to be done to check all database queries. The way this is handled in the individual track assignment is primarily the use of prepared statements with parameterized queries.
* **Authentication** – It is important to be able to restrict access to content from individuals who do not have valid credentials. An authentication system was developed for the backend of the project to authenticate only people with correct credentials. After successful authentication, a token is generated to authenticate users to any of the other backend API endpoints without the need of a password. It is important that any password data is transferred between the frontend backend as less as possible. Still, there is a risk that another entity gets hold of that token. That is precisely why tokens have an expiration date and are refreshed upon expiration.
* **Authorization** – This is another important security aspect that should be taken into consideration if the backend contains any functionality that only a select few authenticated individuals should have access to. The backend implements a role-based authorization system that prevents end-users from using any role-restricted API endpoints if their role does not match.
* **Data exposure** – As mentioned previously, the goal is to share as little information as possible in order to provide the software product without sacrificing any functionalities. That is why all backend endpoints share only crucial data to the frontend.

# Test plan & strategy

Testing is an integral part of software development. That is why I plan to cover a few levels of testing – unit, system and acceptance testing. Moreover, it is expected that the project’s unit tests on the software code will reach at least 50% code coverage.

* **Unit testing** – The lowest level of testing strategy, unit tests will be written for the final software code and will be executed automatically to verify code validity.
  + **Planned coverage:** Unit test targets: model (Item, User, Warehouse, etc.) , service (the database managers), utility (JWTUtils, MessageDecoder, MessageEncoder, etc.) classes and main frontend components.
  + **Used technologies:** Junit (Java Testing library), Mockito (Java Mocking library), Jest (ReactJS Testing library)
* **Integration testing** – Testing strategy that assures that components work correctly when used together.
  + **Planned coverage:** Integration test targets: Jax-RS (Jersey) HTTP Endpoints;
  + **Used technologies:** REST Client
* **System testing** – Second to the highest level of testing strategy, this will be done manually to check if the application produces all desired outputs and confirm that components interact with one another and with the system as a whole without issues.
* **Acceptance testing** – The highest level of testing strategy. Part of this will also be done manually to see whether the system satisfies the acceptance criteria – user needs and requirements, business processes, etc. Another part will be automated to verify that all CRUD features function without any issues.
  + **Planned coverage**: The application’s core CRUD item functionality
  + **Used technologies:** Selenium

# Shape, polygon Description automatically generatedCI/CD Workflow

The presence of a stable and well-defined continuous integration/continuous delivery (CI/CD) system creates a semi-autonomous workflow process that eases the development of large scale and/or long period projects. That is why it is essential to know the parts that will create the CI/CD pipeline. The graph above highlights six important steps in the planned workflow for this project:

1. **Development** – This is the core and beginning of the project cycle. What is important is to have a good integrated development environment (IDE) that can be easily customized to the developer’s tastes. For this project, I decided to use IntelliJ IDEA as it is a good development environment for Java-based applications. The development process can be divided into research, code writing and debugging.
2. **Build** – This process validates the integrity of the code. If the application is successfully built without any runtime errors, it means that the codebase can now proceed to testing without the risk of such errors interrupting the process. As the backend (the RESTful API webservice) is developed using Java, a set of tools, also known as the Gradle environment, are being used to make sure that project dependencies are properly managed.
3. **Testing** – After a successful build of the application, unit and integration tests can be performed to confirm that all processes work as intended. As functionality often changes over time, it is important to have these tests execute after every commit to the git repository to confirm that the code is still properly functioning. Unit and integration tests are going to be written using the JUnit framework for Java.
4. **Quality Assurance** – Before the code base is deployed to production, it should go through a set of quality metric gates that assurance that there are no bugs, vulnerabilities or bad code practices. To do that, I plan to integrate both SonarQube and JaCoCo (a java library that creates testing reports) into the pipeline to confirm that no issues that have been looked over are found.
5. **Acceptance Testing** – Just before deployment, the application needs to go through a set of acceptance-level tests to confirm that it can run on different systems. For that, I plan to utilize the Selenium framework.
6. **Deployment** – The final step of the CI/CD cycle is deployment. Using a docker-in-docker style integration, the project will be packaged and deployed into a production environment. In this case, the software is going to be packaged into a docker image and uploaded to a registry such as Docker Hub, from where the developer can pull the image at any time and run a container off of it. Furthermore, the Docker Compose extension was also chosen so that multiple containers can be configured and ran at once in a given order to assure that the Rest API backend and a database are running at the same time.