Architecture Documentation

Table of contents

1. Introduction and Goals

Simple Example Full-Stack Application with simple CRUD operation.

1.1. Requirements Overview

CRUD-Operations should be working.

1.1.1. Functional Requirements

- create Task
- read Task(s)
- update Task
- delete Task
- Responsive Design
- Java Backend and JavaScript Frontend

1.1.2. Non-Functional Requirements

- Clean Code
- Low Fault Tolerance
- Any Platform
- Documentation

1.2. Quality Goals

Quality	**Motivation**
Efficiency	*The Adapter should be fast and doing only adapter/format stuff.*
Understandability	*The Adapter should be simple, so that non java experts can develop there.*
Testability	*The architecture should allow easy testing of all main building blocks.*

1.3. Stakeholders

Role/Name	**Contact**	**Expectations**
Myself	---	*Develop the application.*
Curious people	---	*Learn about my developer skills.*

2. Architecture Constraints

2.1. technical boundary conditions

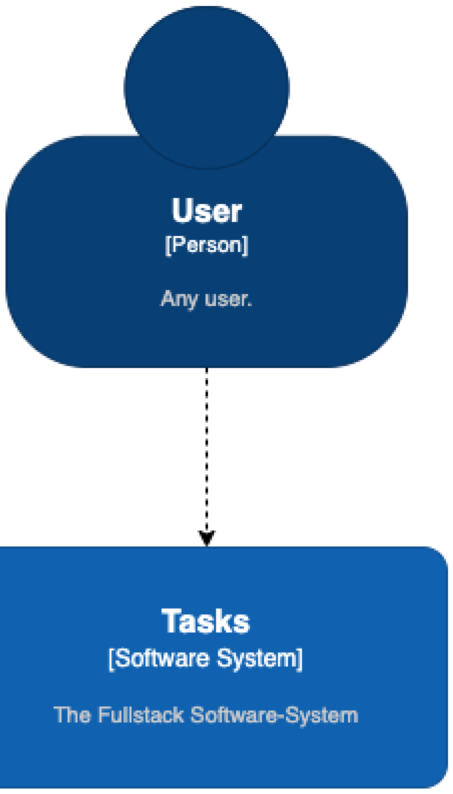
boundary condition	Explanations, background
Hardware specifications	There were no specific availability requirements or the recoverability of the hardware formulated by the client.
Software Defaults	The GitHub-Stack is in use.
user interface	The user interface works with Angular (responsive). The communication with the frontend works via REST.
specifications of system operation	The operating system can every device with installing the development requirements.
programming defaults	Implementation should be done in Java with check for jacoco, pmd, spotbugs and owasp.
data structure	Data structure is simple JSON.

2.2. organizational boundary conditions

boundary condition	Explanations, background
organization and structure	This is a one-man team.
Time schedule	Not time critical.
organizational standards	The software is designed to be incremental and iterative using the Scrum frameworks and the rules defined therein and procedures are developed. To document the architecture arc42 is used. The project documentation should be in Atlassian Confluence Wiki to be created. The management of work packages should be done using Atlassian JIRA.
development tools	Creation of the Java/Javascript source code in any development environment. The software must also be buildable alone with Gradle, i.e. without an IDE, in order to enable automation of the deployment. As version Control System should use Git. The administration of Git repositories should be done using GitHub.)

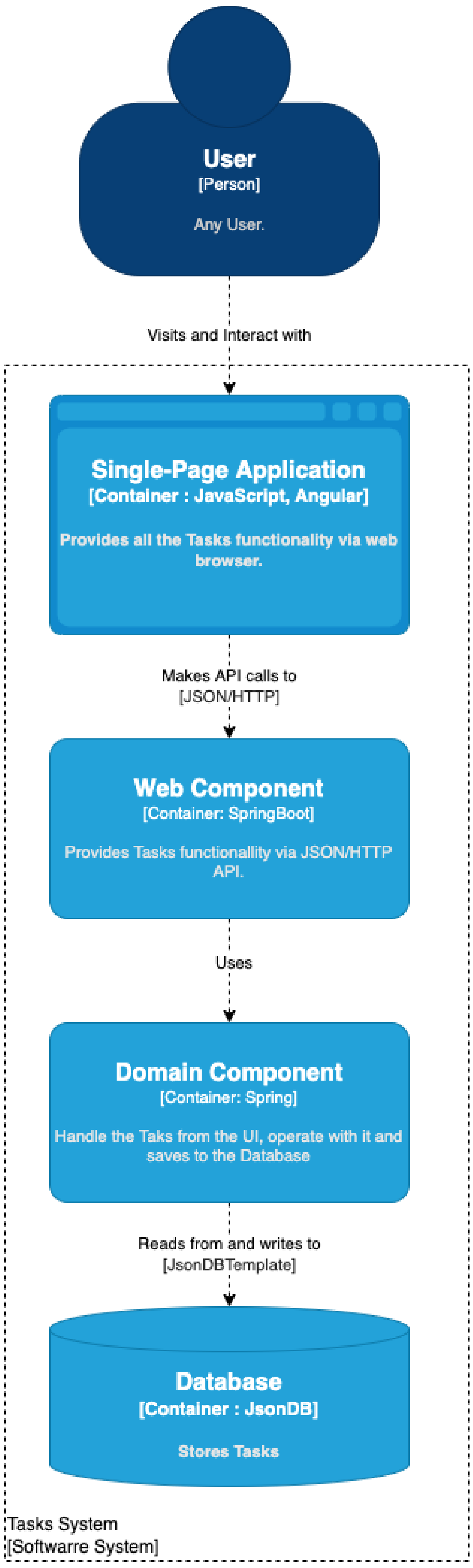
3. System Scope and Context

System Context diagram for Fullstack-Tasks



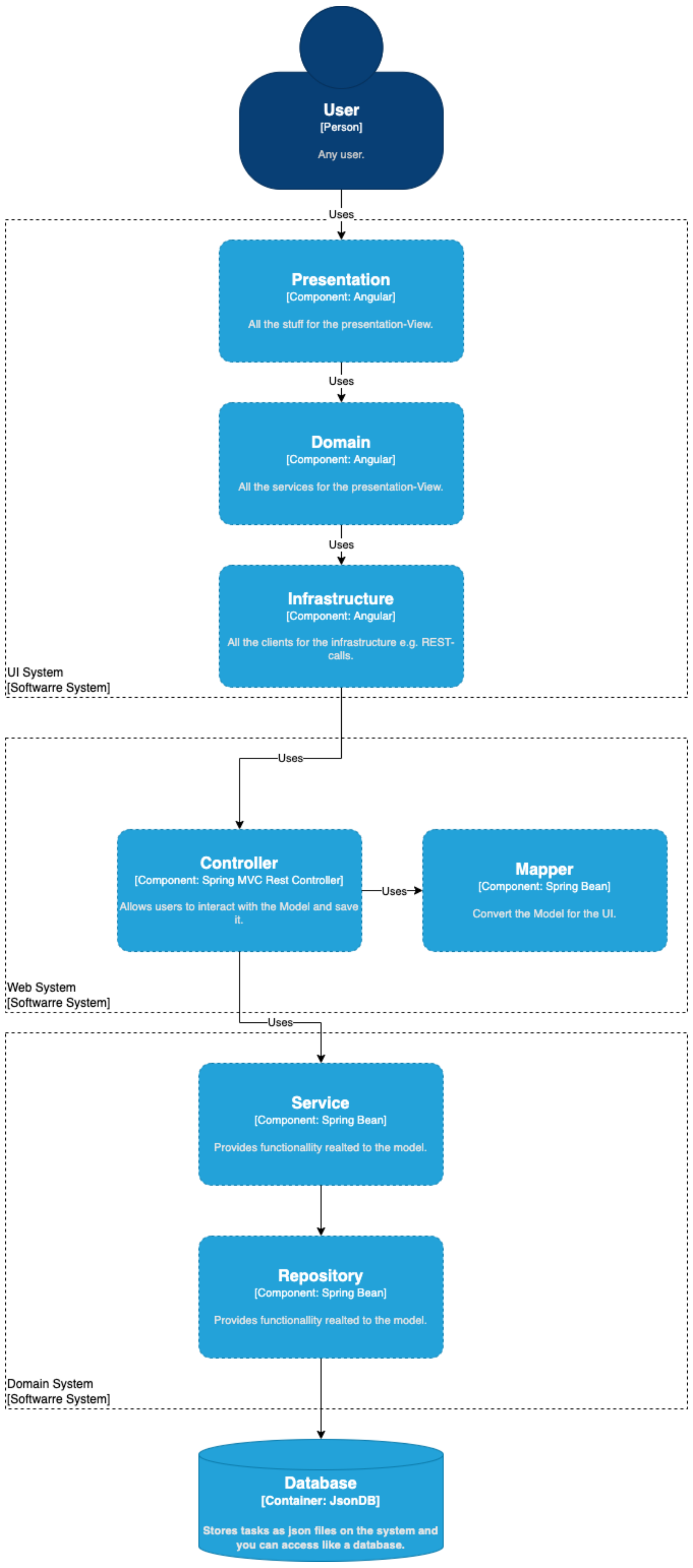
3.1. Business Context

Container diagram for Fullstack-Tasks


User
[Person]
Any User.
Visits and Interact with
Single-Page Application
[Container : JavaScript, Angular]
Provides all the Tasks functionality via web browser.
Makes API calls to
[JSON/HTTP]
Web Component
[Container: SpringBoot]
Provides Tasks functionallity via JSON/HTTP API.
Uses
Domain Component
[Container: Spring]
Handle the Taks from the UI, operate with it and saves to the Database
Reads from and writes to
[JsonDBTemplate]
Database
[Container : JsonDB]
Stores Tasks
Tasks System
[Softwarre System]

3.2. Technical Context

Component diagram for Fullstack-Tasks


User
[Person]
Any user.
Uses
Presentation
[Component: Angular]
All the stuff for the presentation-View.
Uses
Domain
[Component: Angular]
All the services for the presentation-View.
Uses
Infrastructure
[Component: Angular]
All the clients for the infrastructure e.g. REST-calls.
UI System
[Softwarre System]
Uses
Controller
[Component: Spring MVC Rest Controller]
Allows users to interact with the Model and save it.
Uses
Mapper
[Component: Spring Bean]
Convert the Model for the UI.
Web System
[Softwarre System]
Uses
Service
[Component: Spring Bean]
Provides functionallity realted to the model.
Repository
[Component: Spring Bean]
Provides functionallity realted to the model.
Domain System
[Softwarre System]
Database
[Container: JsonDB]
Stores tasks as json files on the system and you can access like a database.

4. Solution Strategy

Although data centric, the application resigns from using too much SQL for creating reports, summaries and such but tries to achieve that with Java 8 (or above) features around streams, lambdas and map/reduce functions.

Building the application with Spring Boot is an obvious choice as one of the main quality goals is learning about it. But furthermore using Spring Boot as a "framework" for Spring Framework allows concentration on the business logic. On the one hand there is no need to understand a complex XML configuration and on the other hand all building blocks are still put together using dependency injection.

Regarding dependency injection and testability: All injection should be done via constructor injection, setter injection is only allowed when there’s no other technical way. This way units under tests can only be correctly instantiated. Otherwise one tends to forget collaborators or even worse: 20 injected attributes may not hurt, but a constructor with 20 parameters will. This hopefully prevents centralized "god classes" that control pretty much every other aspect of the application.

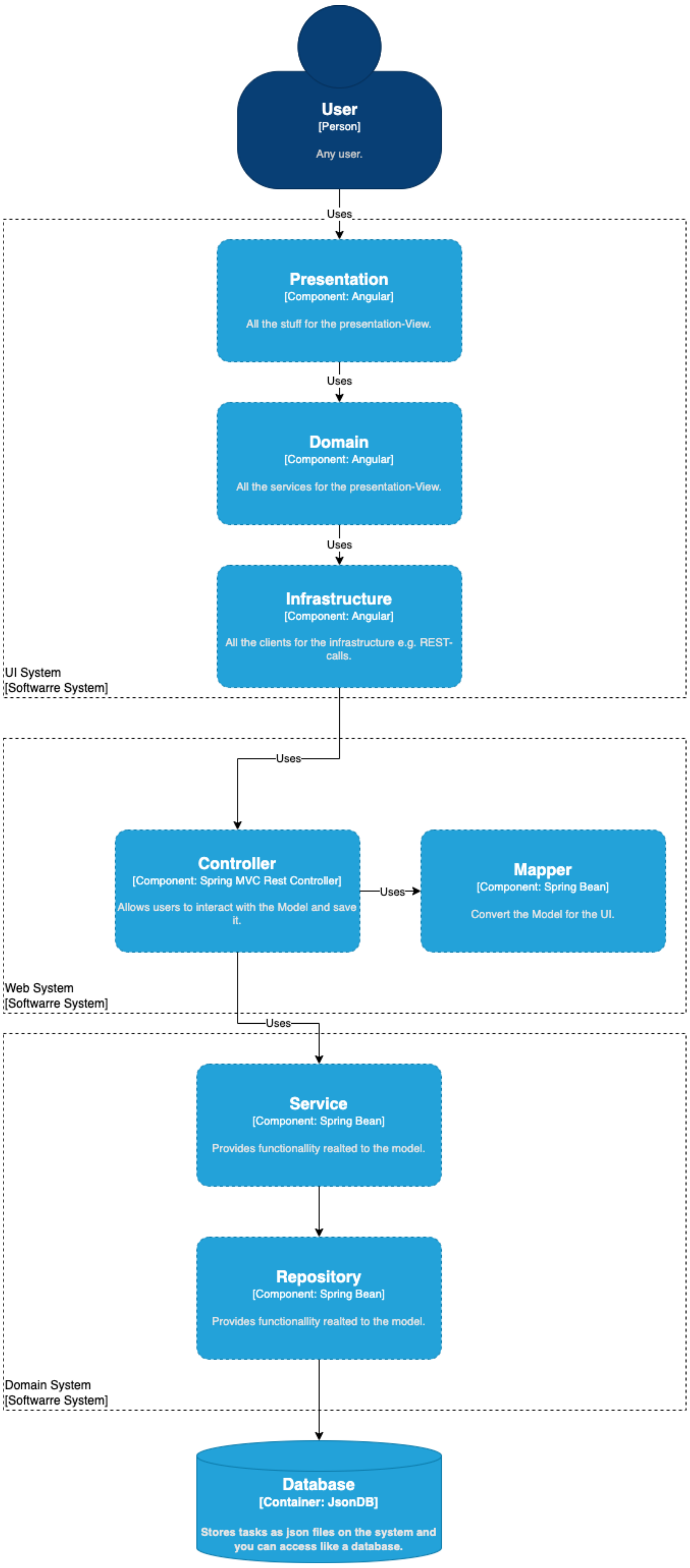
Spring Boot applications can be packaged as single, "fat jar" files. Since Spring Boot 1.3 those files contain a startup script and can be directly linked to /etc/init.d on a standard Linux systems which serves.

Interoperability will be achieved by using JSON over simple http protocol for the main API. Security is not the main focus of this application. It should be secure enough to prevent others from tempering with the data, confidentiality is not a main concern (read: passwords can be transmitted in plain over http).

The internal single page application shall be implemented using Angular. The deployable artifact will contain this application so there is no need for hosting a second webserver for the static files.

5. Building Block View

Component diagram for Fullstack-Tasks

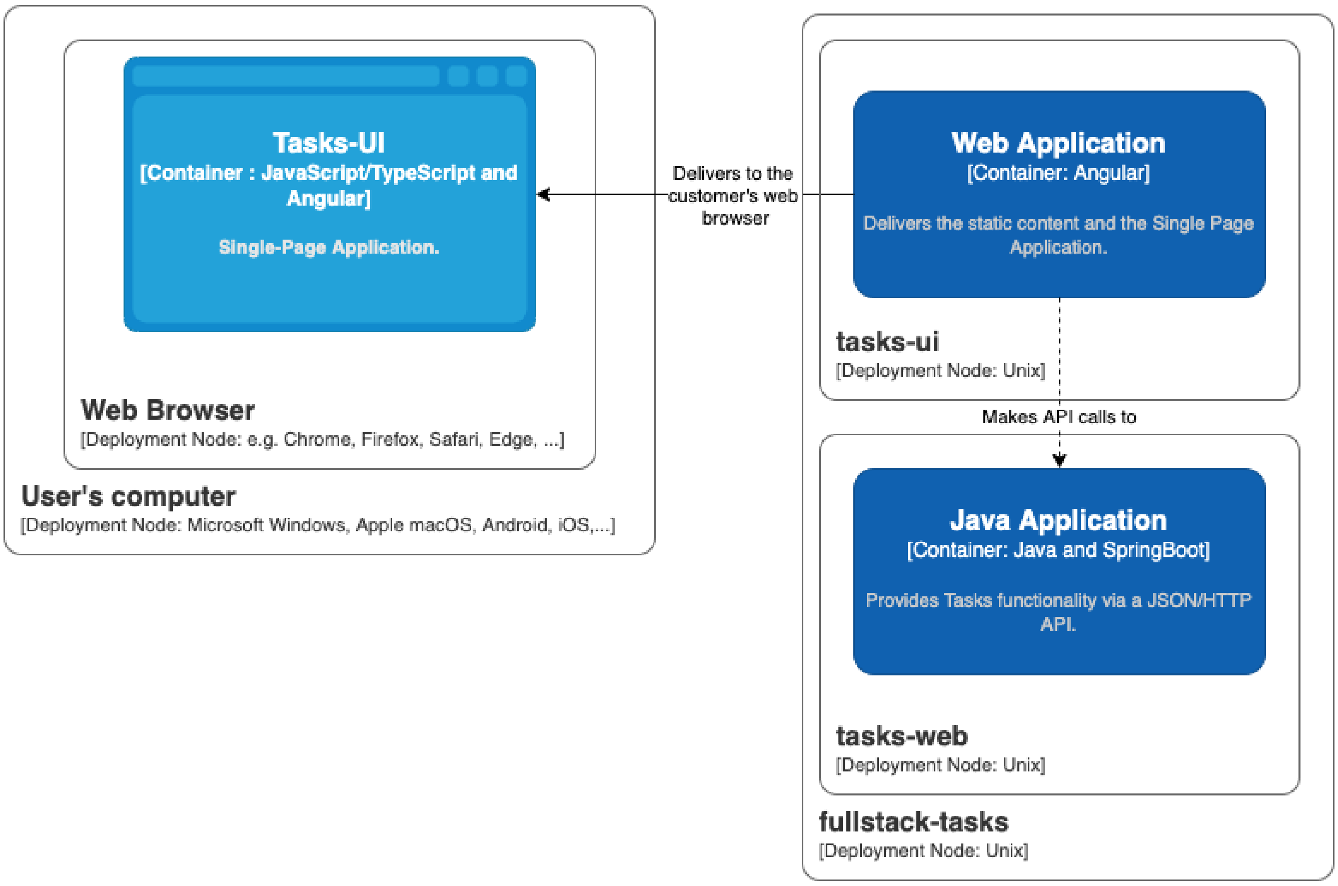

User
[Person]
Any user.
Uses
Presentation
[Component: Angular]
All the stuff for the presentation-View.
Uses
Domain
[Component: Angular]
All the services for the presentation-View.
Uses
Infrastructure
[Component: Angular]
All the clients for the infrastructure e.g. REST-calls.
UI System
[Softwarre System]
Uses
Controller
[Component: Spring MVC Rest Controller]
Allows users to interact with the Model and save it.
Uses
Mapper
[Component: Spring Bean]
Convert the Model for the UI.
Web System
[Softwarre System]
Uses
Service
[Component: Spring Bean]
Provides functionallity realted to the model.
Repository
[Component: Spring Bean]
Provides functionallity realted to the model.
Domain System
[Softwarre System]
Database
[Container: JsonDB]
Stores tasks as json files on the system and you can access like a database.

6. Runtime View

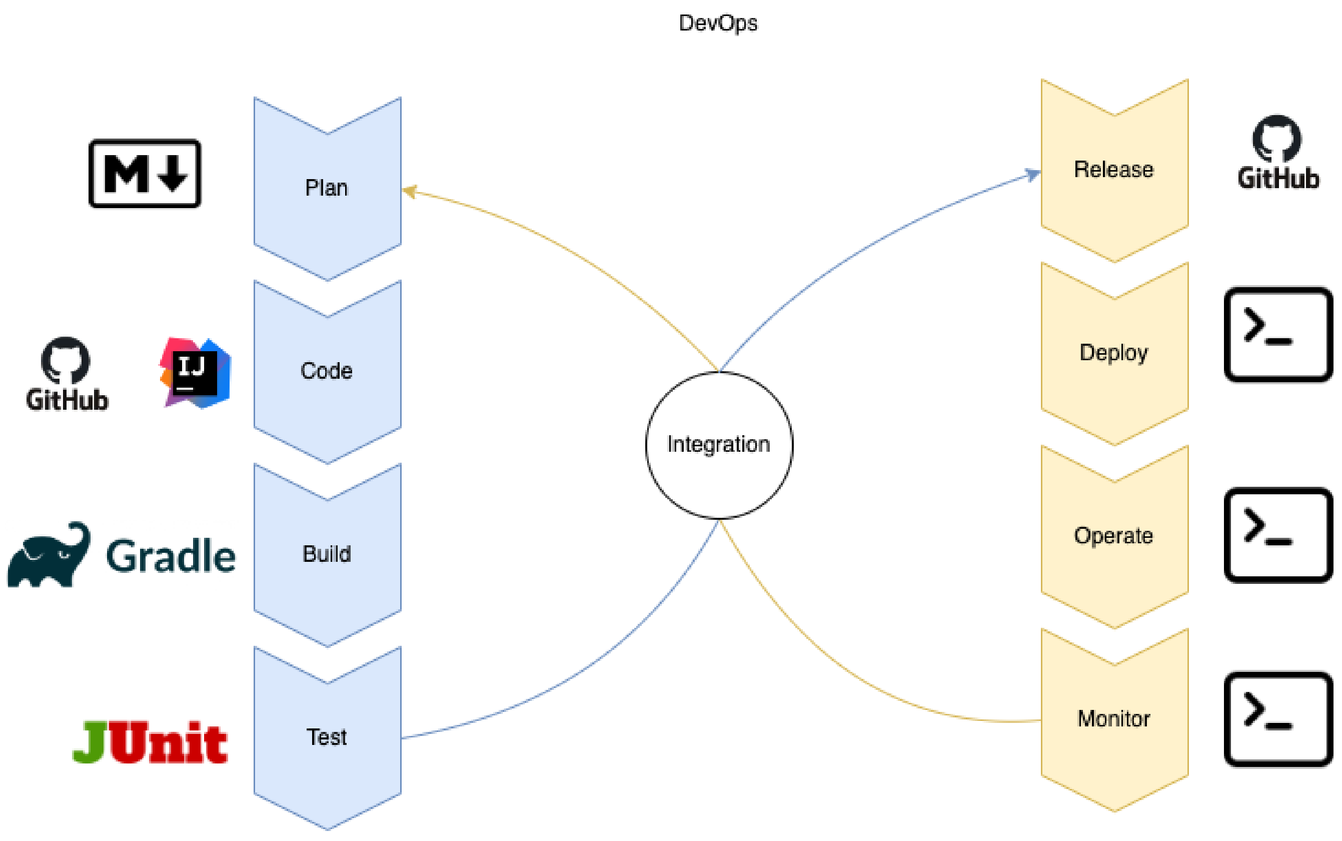
Not appropriate for this system due to very simple implementation.

7. Deployment View

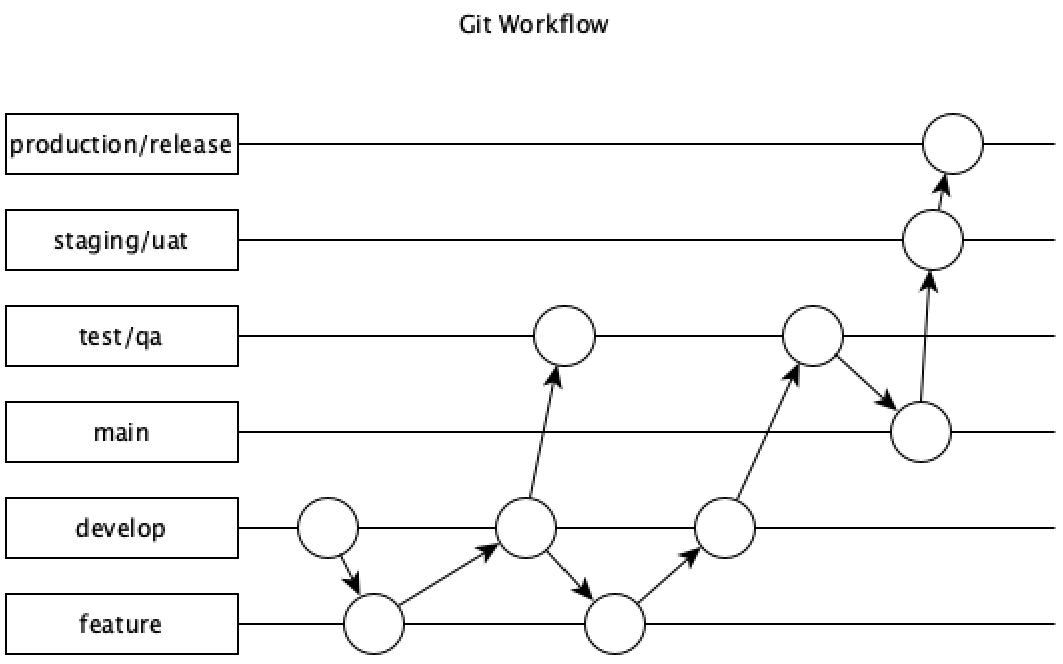
Deployment diagram for the fullstack-tasks stystem.


Tasks-UI
[Container : JavaScript/TypeScript and Angular]
Single-Page Application.
Web Browser
[Deployment Node: e.g. Chrome, Firefox, Safari, Edge, ...]
User's computer
[Deployment Node: Microsoft Windows, Apple macOS, Android, iOS,...]
Delivers to the customer's web browser
Web Application
[Container: Angular]
Delivers the static content and the Single Page Application.
tasks-ui
[Deployment Node: Unix]
Makes API calls to
Java Application
[Container: Java and SpringBoot]
Provides Tasks functionality via a JSON/HTTP API.
tasks-web
[Deployment Node: Unix]
fullstack-tasks
[Deployment Node: Unix]

7.1. *DevOps*



7.2. *Git Workflow*



8. Cross-cutting Concepts

This simple Application has no cross-cutting cases.

9. Architecture Decisions

9.1. Architecture Principle

Date: 2022-01-01

9.1.1. Status

Accepted

9.1.2. Context

How I want structure the project?

9.1.3. Decision

package-by-component concept.

9.1.4. Consequences

I can not access from the web-component to the database, but that should impede the principle.

9.2. File Based Database

Date: 2022-01-01

9.2.1. Status

Accepted

9.2.2. Context

First approach was a Docker Container with a MongoDB, but at the start of the project it was not possible to install docker. So we will choose a different MongoDB similar DB without a Docker Container or big database overhead.

9.2.3. Decision

JsonDB is a lightweight Json-file database.

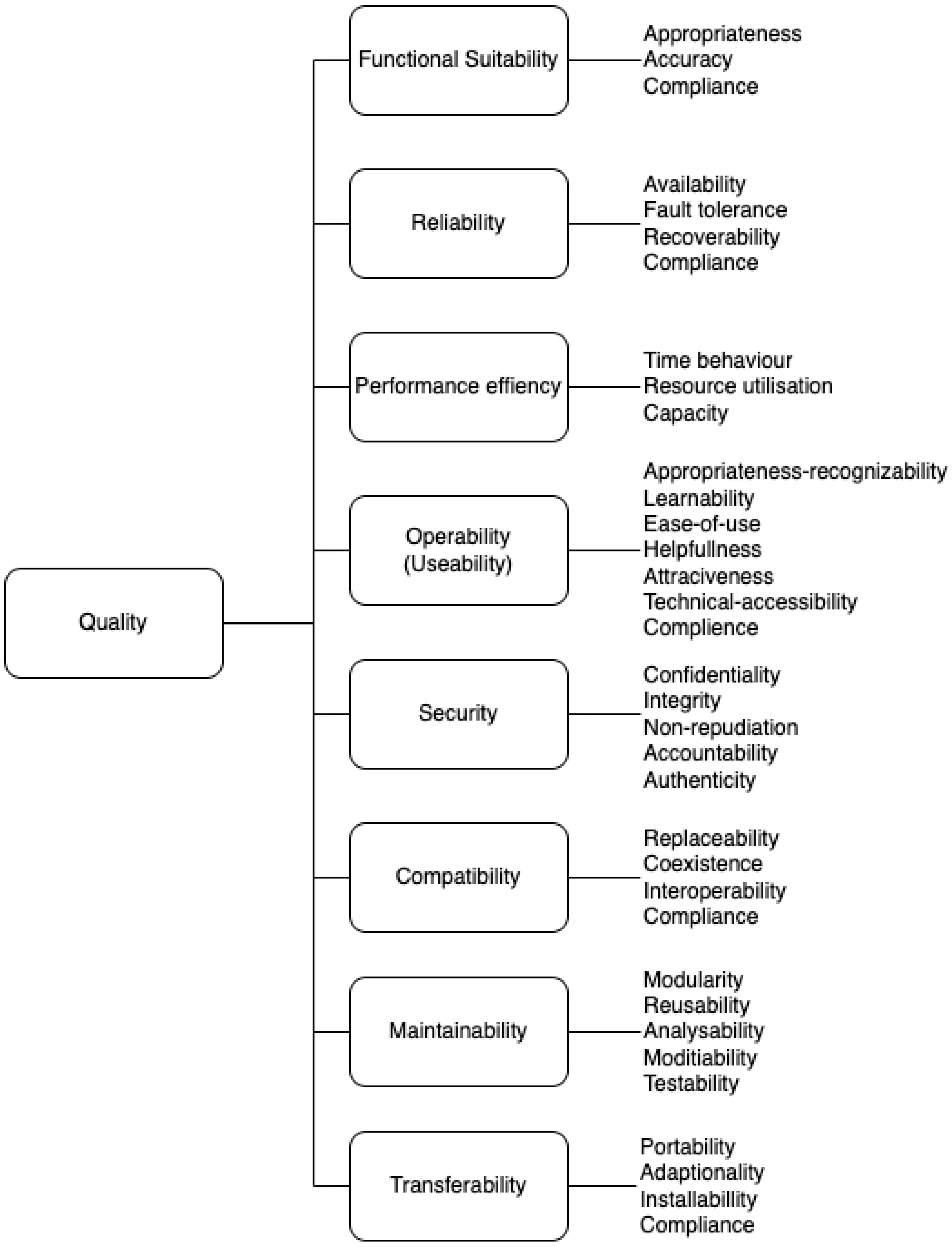
9.2.4. Consequences

File-system database is not central, but for this example it is good enough.

10. Quality Requirements

During the build process, the following quality checks should already be checked and if one is not correct, then the entire build process should fail.

10.1. Quality Tree



10.2. Quality Scenarios

Tools during build process:

- run all Tests (do not skip Tests)
    - run gradle without the option `-x test` !

- Tests have a test coverage of 80% (business logic, no pojo)
    - Tool: Jacoco
- static Codeanalysis
    - Tools: PMD, Spotbugs
- check security of the software
    - Tool: OWASP

Additionally, watch the console.

11. Risks and Technical Debts

- High effort at the implementation
- Low software quality due to the lack of change and quality management
- Other Systems (especially Windows only with cmd) could be a little difficult

12. Glossary

Term	Definition
Task	*The Todo-element*