Software architecture document (SAD)

Yordan Doykov

Software architecture document

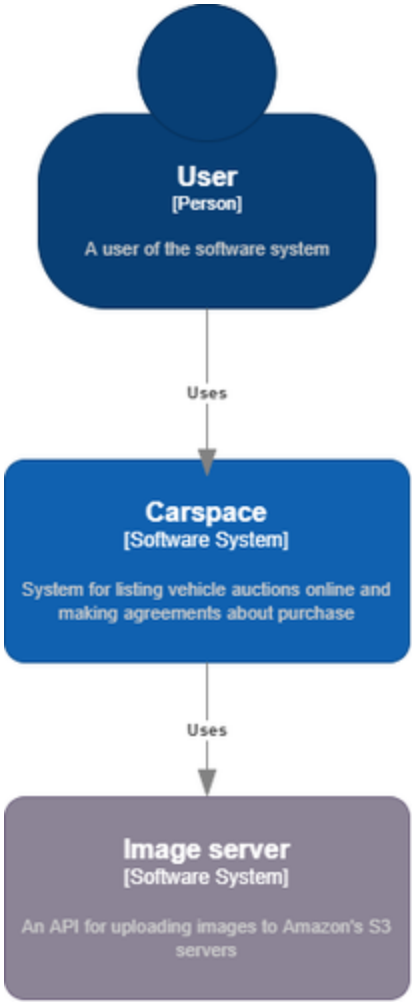
For S3 Individual track project

Revisions			
Verison	Description	Date	Author
1	Added base SAD structure and C4 diagrams	24.09.2022	Yordan Doykov
1.1	Updated C2 and C3	2.10.2022	Yordan Doykov
1.2	Updated C3 with repository layer	24.10.2022	Yordan Doykov
1.3	Updated document with CI setup	14.11.2022	Yordan Doykov

Introduction

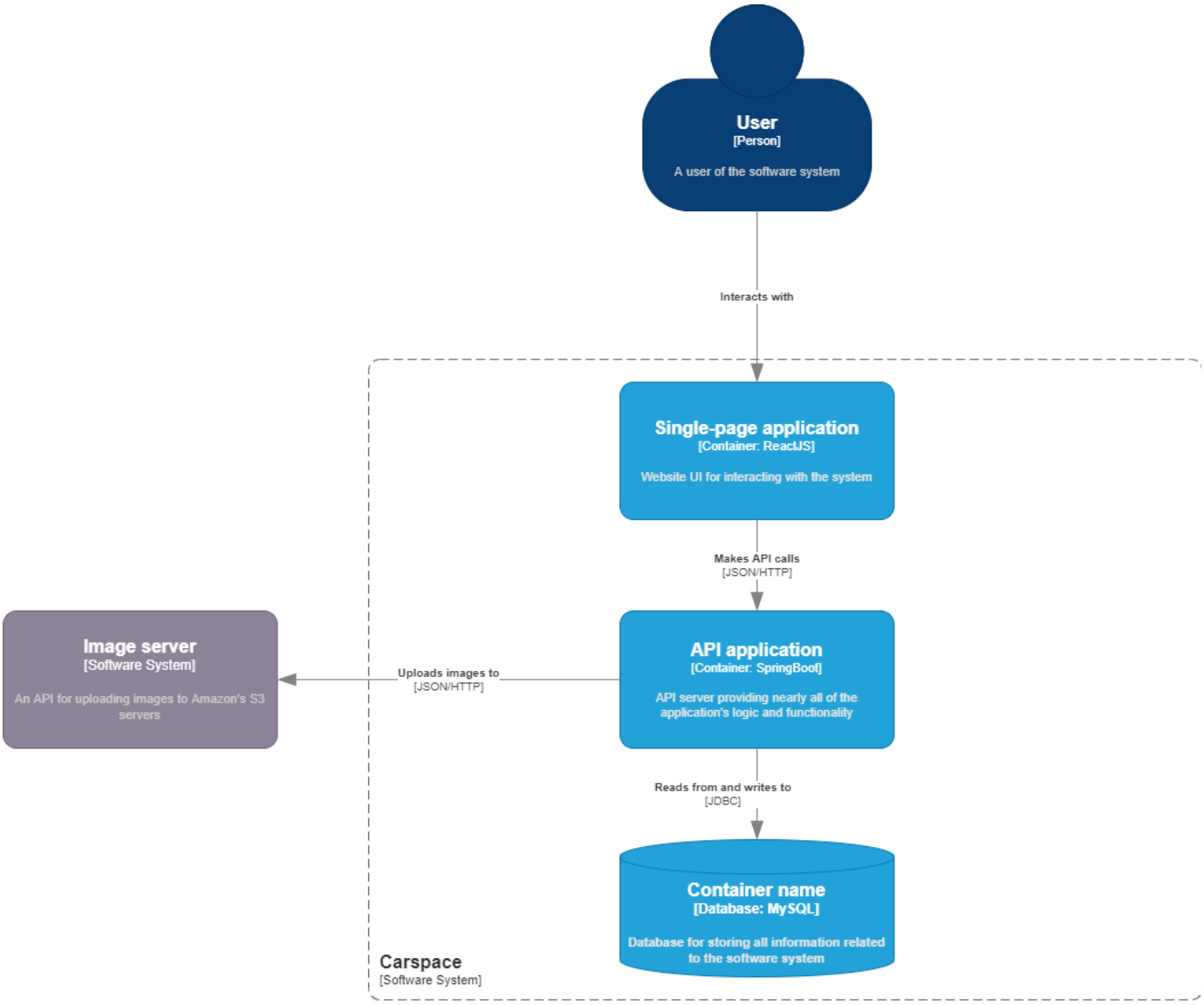
This document provides a comprehensive architectural overview of the system, using a number of different views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made to implement the software solution. It also to help developers more easily grasp the software's modules and components, without digging into the code, while also being incredibly useful as it gives an oversight of the entire system and exposes any potential pitfalls the developers might experience.

System context - C1



From the system context diagram it can be seen that the software is going to make use of an external image server, which will be used for uploading all of the images to a remote Amazon S3 bucket. This way such a concern is completely separate from the system and it requires no integration

Containers and technology - C2



From the diagram it can be seen that the software system is composed from three main containers:

- Single-page ReactJS application
- REST API using SpringBoot
- MySQL database

Separation of concerns

With the three main containers divided in this way it is ensured that the different divisions of the software are properly separated. The single-page application takes care of all of the UI and does not deal with any business logic. The REST API is where all of the business logic resides and is essentially the brain of operations. It provides all of the functionality of the software solution and passes data from the database to the Single-page application. The final block of the system is the database where all of the information persists.

Why ReactJS

ReactJS is a very popular Javascript framework. It is incredibly powerful and well-documented, and if used properly, you can masterfully build the front-end of your application with both style and functionality. It has exceeding performance as it is lightweight and works on the single-page application principle, where all of the front-end's scripts are loaded at once and the page never actually refreshes. Regular maintenance and updates make sure that it's always up to date and its popularity means that there are plenty of tutorials and support in case you run into any kind of issues. It is also supported by a lot of third-party libraries, so in the event of needing something incredibly niche and specific, chances are that someone already made it specifically for React.

Why SpringBoot

The spring framework runs on Java, which is a very popular programming language and gets tons of support. Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

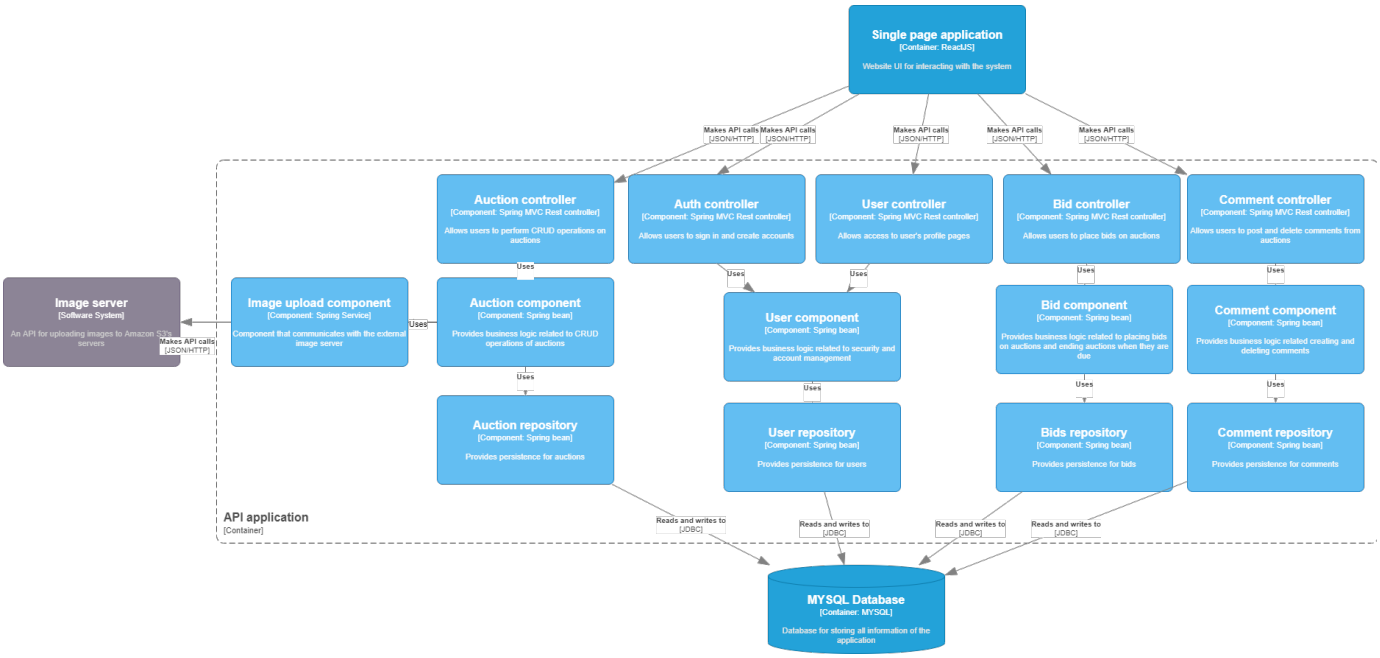
Spring Boot offers the following advantages to its developers

- Easy to understand and get going
- Increases productivity
- Reduces the development time

Spring boot provides a flexible way to configure Java Beans, XML configurations, and Database Transactions, while also providing a powerful batch processing and managing REST endpoints.

In Spring Boot, everything is auto configured, dependency management is made easier and the overall development time is shorter, which is perfect for the applications of this software system.

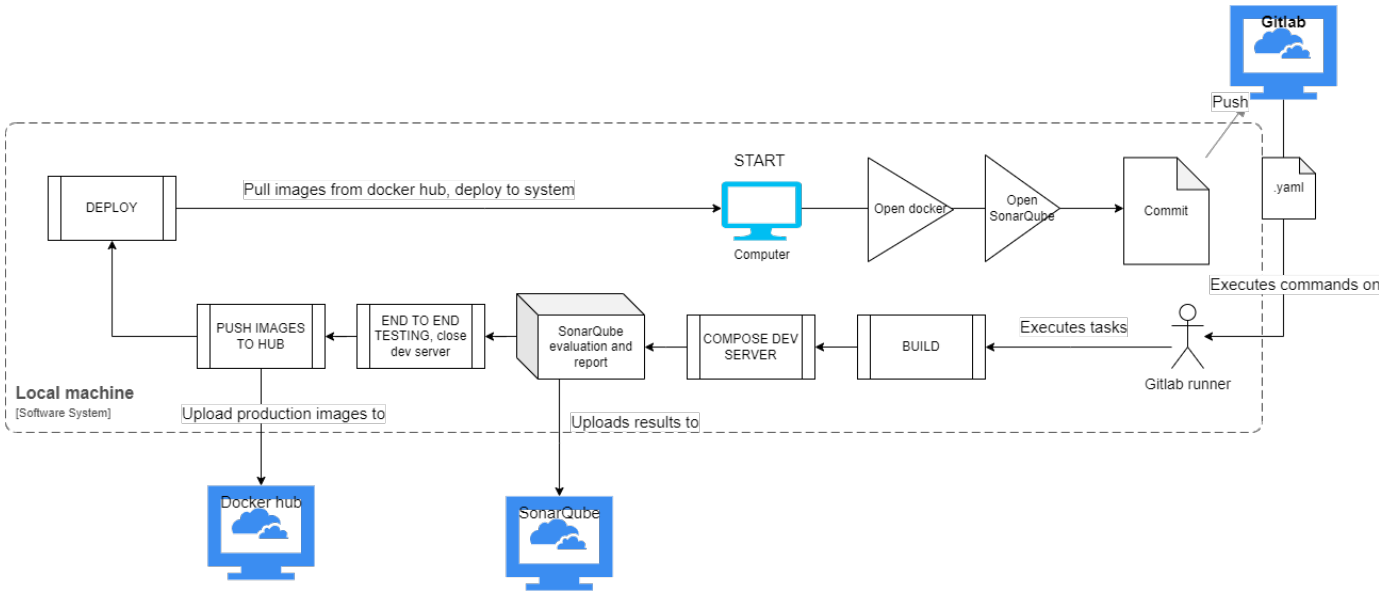
Components - C3



From the component diagram it can be seen in more detail how the API of the system will be structured. Here all logical operations related to some kind feature are separated in their own components. This ensures the Single responsibility principle in **S**OLID. These components are used by the API controllers which manage all incoming requests. Within the components are implemented the other principles of solid like the Liskov substitution, Interface segregation and Dependency inversion through the use of proper inheritance, Dependency injection and the relying of abstractions, rather than concretions - any class implementation would be injected with an interface, rather than an implementation.

CI/CD Setup

The image below illustrates how the CI pipeline of the development is structured



Service approach over use-case class separation

Structurally, the API will be using the service approach, rather than the use-case per class approach. This is done in order to avoid a messy project structure with too many classes that you can get lost in. There will be special attention paid to how much functionality a class bundles, and it case it becomes too much it will be separated in a suitable way in order to avoid the potential god-class problem. With this approach the problem of having “helper classes” can also be avoided, which is something the use-case method causes.

Lombok

Lombok will also be used in the API for convenience purposes, as it avoids having messy classes with boilerplate code. It is very powerful and a lot of functionality is hidden “under the hood”, so there is special attention paid to which annotations are used where in order to avoid security concerns and leaking of information.