INFOSAFE

System Architecture

InfoSafe

Seed Analytics

Team Name:	FrAgile
Team Members:	Christof Steyn
	Chris Mittendorf
	Karel Smit
	Yané van der Westhuizen
	Alistair Ross
Team Contact:	fragile.cos301@gmail.com

Table Of Contents

Introduction

InfoSafe is an application that is used by the Information Security team, System Administrators, Managers and other employees to automate the data security process. It is a single tool that can be used to monitor and manage any and all operations involving a company's data and projects within its Information Security Management System (ISMS).

Architectural Design Strategy

Our team has decided to approach our system design using the strategy of **design based on quality requirements.** After multiple discussions with the client and the numerous breakdowns of the requirements given to us we believe this design strategy is the best choice going forward. We have many quality and functional requirements that need to be implemented and thus have a well defined base onto which we can start implementing the system on.

Architectural Style

Although our system can be considered as a Multitier Architectrure, that consists on the client and server side, and more specifically considered a Model-View-Controller (MVC) Architecture we believe that the system can be described more in depth by the following three architectural patterns; Database-centric, Event-driven and Domain-driven designs.

Although the system does fall under the former stated architectural patterns we believe these patterns can be quite a broad description of a large system and thus decided it would be best to construct and describe our system with the latter mentioned patterns with the below justifications.

Database-Centric Design

This design pattern was chosen due to the crucial role that the database plays in the system. As Infosafe is an Information System it needs to be constantly referring and reading data from the database. As we are also using a relational database management system (RDBMS), as opposed to a file-based data structure or an in-memory system, we believe this architecture pattern aptly encompasses the base structure of the system.

As stated above the system does form the broad structure of an MVC pattern and thus the database makes up the model portion of it, which further justifies the need to construct the system using this approach.

Event-Driven Design

An Event-driven design focuses mainly on the change in state of the system. Due to the nature of the core requirements to consistently show, fetch and update data in the database this pattern aptly describes the process of how this happens. The event takes the form of the user making a request on the View (Client side), and the request being processed by the controller on the server side to return or manipulate data in the database as per the user request.

A positive to using this design approach is that when systems are built around an Event-driven Architecture, it simplifies the horizontal scalability across multiple users and makes the overall system more resistant to failure and increases its availability, a quality requirement that will be described later.

Domain-Driven Design

The third design pattern we are focusing on is the Domain-driven design. This approach focuses on structuring the controller and functions to match the domain according to the client. Under this approach we aim to structure the classes, methods, variables and functions in order to match the business logic provided to us by the client. This way it is easy for both us as the developers and clients to understand the logic in the system and leaves little room for ambiguity or errors.

This approach will also allow us to modularise the code and thus increase the reusability and maintainability of the functions in the system for efficiency.

Architectural Quality Requirements

Availability

Specification

The system should be operable at all times to provide the most accurate and up to date version of the database to ensure that the company and all users involved can work to the highest standards. Any number of users should be able to use the system and there should be no noticeable change in performance or speed. Due to the system being of a Database-centric design it is important that the system is able to connect to the database at all time and display the most recent and accurate data stored.

Quantification

The system should allow for at least 70 users to use the system concurrently without any hindrance in performance or efficiency. The system will be able to support a much larger user base as another quality requirement is the scalability which will be described below. Another important aspect to note is that all database transactions should be locked while performing any write changes so that data is not lost or the wrong data is returned at a later stage.

Security

Specification

Security is another very important aspect to the Infosafe system. Due to the personal and sensitive data both on the employee and datascope side, access to the system needs to be limited and properly secured. No unauthorized access to the system can take place as it is important that data cannot be altered or deleted by anyone except the ISO/DISO and data custodians.

Quantification

To limit unauthorized access to the system, only the ISO may create new users. Users will be supplied with a generated password, that meets security requirements, that will give users first time access to the system where they will be expected to create their own password. All passwords will be encrypted (hashed and salted) and only the encrypted strings will be saved in a separate table to the main user data.

Durability

Specification

As the system is a Database-centric design, the durability is part of the ACID (atomicity, consistency, isolation and durability) property which guarantees transactions have been committed and stored properly and permanently in the database. As there may be multiple read/writes to the system happening simultaneously it will be important that the information stored is the correct and most up to date version.

Quantification

We can achieve durability by writing all transactions to a transaction log that, incase of any failure, can be used to restore the system to the state before the error took place.

Scalability

Specification

The system should be scalable to allow for growth in the company and the addition of new users into the system and their data in the database. The database should also be able to grow in size without worry of reaching storage capacity and thus the system not being able to run as intended.

Quantification

Hosting and the online database will take place through Amazon Web Services (AWS) which allow for any growth in a system at additional costs. Through this the system and database will be able to grow without any hindrance to performance to the user or company. AWS also adjusts and allows for load balancing, so should the system at all become congested, AWS will be able to still supply the required services at an acceptable level.

Usability

Specification

The system should be easy to use and easy to navigate. The interface should be easy to understand and allow the user to come familiar with it with little to no effort.Interactions in the systems should be seamless and promote productivity and help increase the throughput of work.

Quantification

As stated above, the interface should be easy to understand and to navigate the website to utilize all aspects of the system. A user manual will be provided to help users understand the system and to be able to use it. The user manual will also provide some deeper insights to the system that can help with future development and system growth.

We will do some user acceptance testing to ensure that the system is actually easy to use. We will allow our client to use the system at their leisure to make sure that the system meets their expectations and requirements are met.

Functional Suitability

Specification

The functional suitability refers to the extent the system meets the clients' requests and requirements. The system should perform as intended by the clients and all features will support the needs of the user as required.

Quantification

By working closely with our client we intend to make sure we have a complete understanding of each subsystem and specific use case before any implementation begins. We have weekly or sometimes bi-weekly meetings with the client and discuss each subsystem in-depth to make sure there is no misunderstanding on our side and they are happy with the way we intend to implement.

Dependability

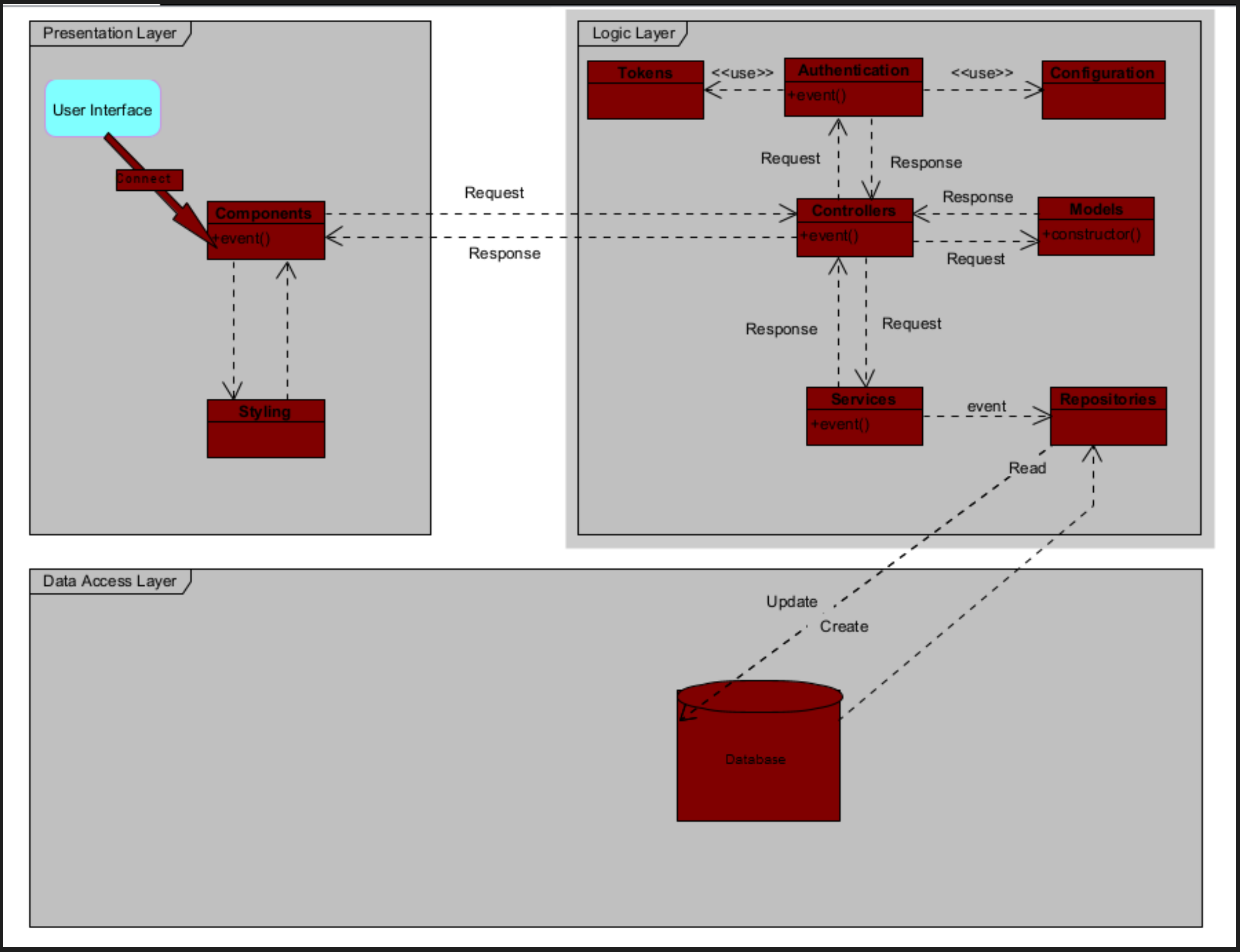
Specification

Lastly the system should be dependable. In systems engineering, dependability refers to the systems availability (discussed above), reliability, maintainability, durability (discussed above), safety and security (discussed above). Although we have already covered some of the topics that make up dependability, we thought it was important to note the system, at the end of the day, should be dependable and all the topics it refers to, even though it is not our most important quality requirement.

Architectural Design and Patterns

System Design

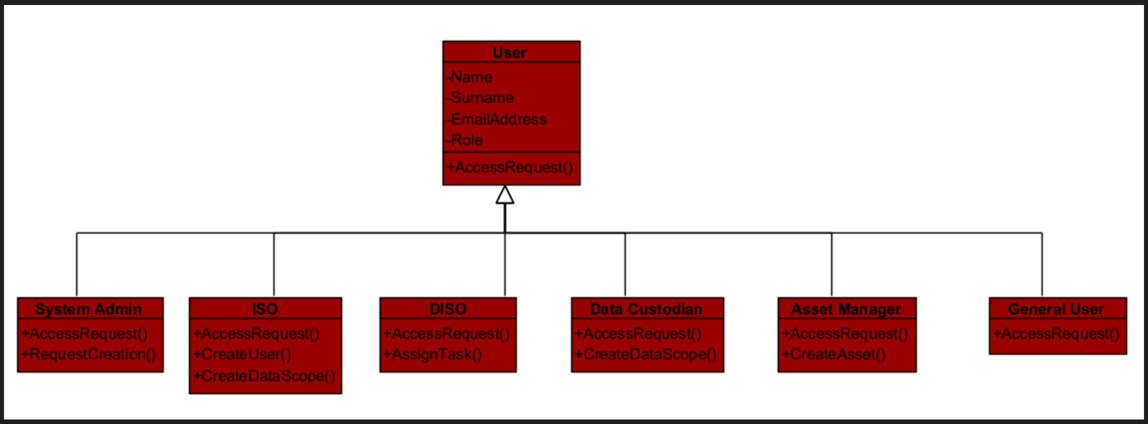
As stated earlier our system does have a multi-tier structure. The presentation layer communicates with the logic layer that in turn reads and writes to the data access layer where the database is located. As the system is Database-Centric, 90% of processes interact with the database either in the form of reads or writes. Events from the client side invoke the backend that communicates with the database, making the system very Event-driven in style. Lastly the system is Domain-Driven in architectural style too, meaning the models and functions are set up in a manner that the client has specified and will understand. Located below is a diagram of the system:



Design Patterns

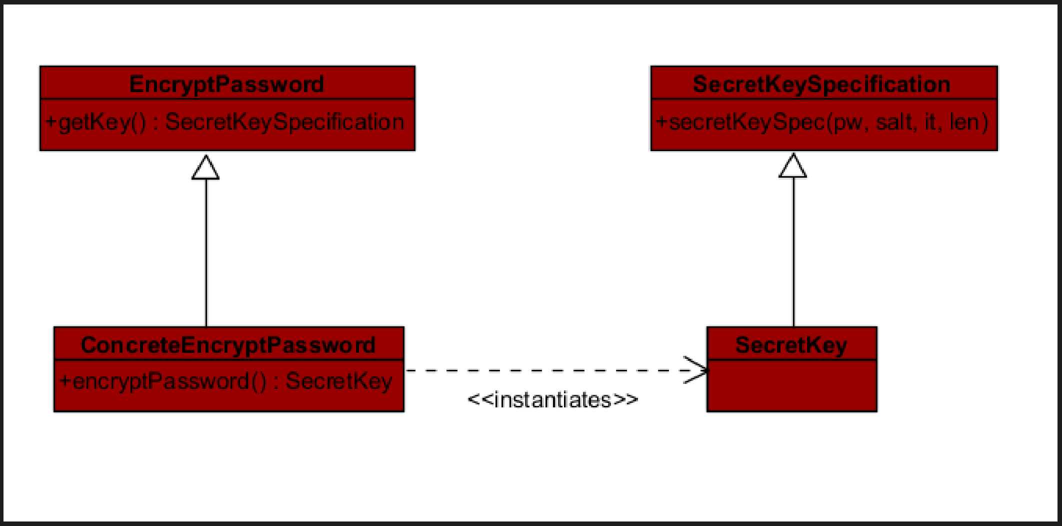
Template

The template design pattern was used to implement different types of users and assets. Although the internal structure of a user is the same, the way the user interacts with the system will change based on its roles. Assets in a similar fashion can be of different types and thus their intractability changes depending on their type.



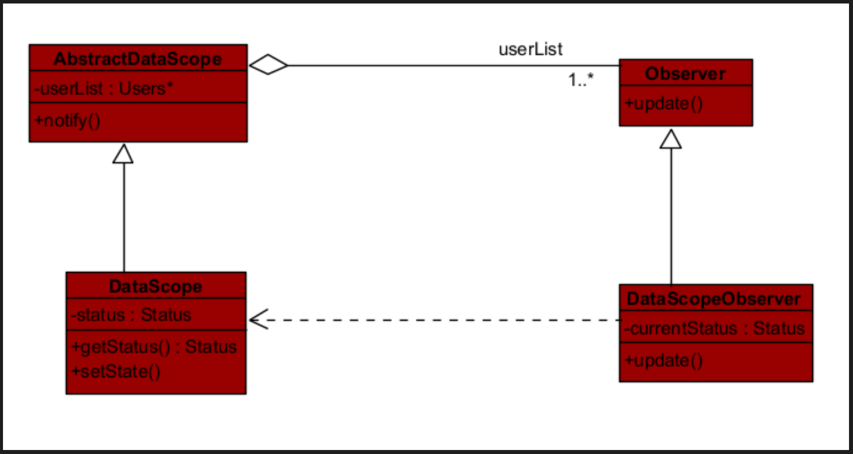
Factory

The factory design pattern is used in our password encryption and hashing. A secret key factory is created that takes in the original password in the form of a character array, the byte array of the salt, number of iterations and the desired key length. This then generates a key that is later used to encrypt the user's password.


EncryptPassword
+getKey() : SecretKeySpecification
SecretKeySpecification
+secretKeySpec(pw, salt, it, len)
ConcreteEncryptPassword
+encryptPassword() : SecretKey
SecretKey
<<instantiates>>

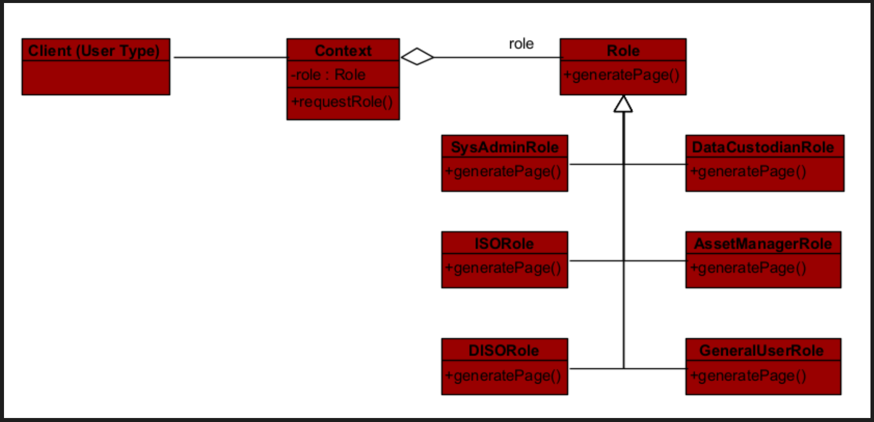
Observer

The Observer pattern is used in a few cases in the system. Mainly to notify all users that have access to a Data Scope, and to notify all users involved in tasks. The observer pattern will notify the relevant users when changes take place in a data scope or when another user is requesting access to a data scope. When tasks are assigned to users, that user will be notified and the assigner will also be notified when a task status changes or a task is submitted. The observer pattern makes this all possible.



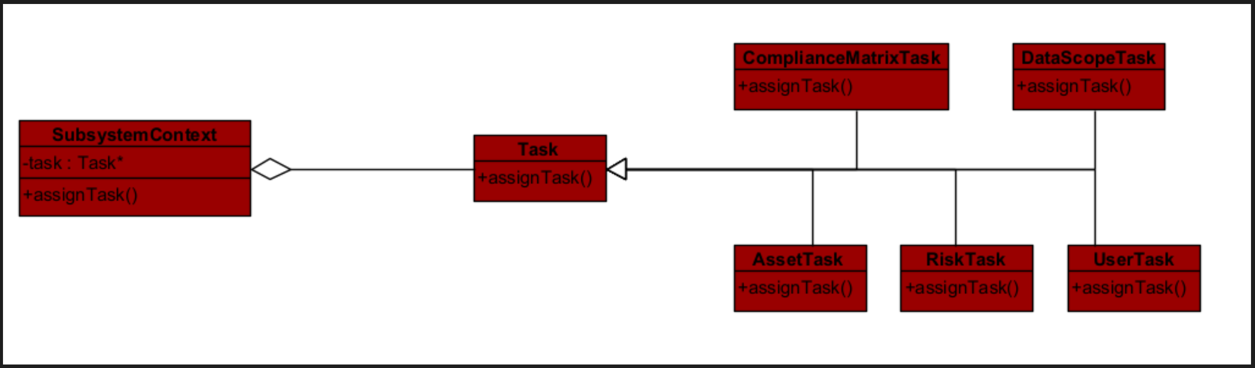
State

The state pattern is used to modify how the web pages generate based on the type of user that is using it. For example the ISO has far more functionality and privileges than the General User and the system needs to cater for this. Some pages will only generate for specific users and functionality on pages also changes based on the user role.



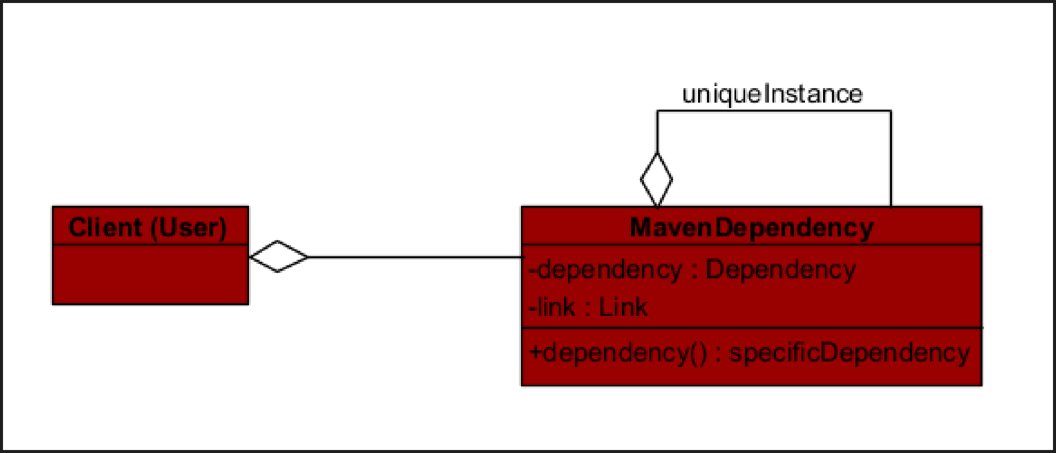
Strategy

We use the strategy pattern to define how our tasks are assigned and used in the compliance matrix subsystem. Based on specific inputs from the ISO on task creation the task may behave differently due to the subsystem it is also defined to.



Singleton

Singletons are used when a maven dependency is imported to the system. Creating it as a singleton ensures that only a single instance of the dependency is created and is the only instance of it referenced and used in the system. All dependencies are located in the pom.xml file in the Backend directory of the repository.



Architectural Constraints

In any project there will be constraints to the system that would need to be accounted for or solutions to be made to mitigate these constraints. In our system the main constraints have appeared due to the technologies required by the client which will be discussed below. On top the technological constraints we also had some requirements laid out by the client as follows:

- Open source
- Cloud based
- Multi-platform
- MVC framework
- Frontend - Javascript (React)
- Backend - Java (Spring Boot)
- Hosting - AWS

Below is discussed how our design choices help solve some of the constraints and how we approached each constraint to not hinder the systems' development.

Open Source

The client has specified that all additional technologies used on the project should be open source and easily available to access. This request was to ensure the longevity of the system and that no current technologies become restricted in the future that could have catastrophic effects on the system. The team has made sure any and all libraries and technologies used are open source and available for free use.

Cloud Based

The client asked for hosting and deployment to be cloud based so the system can be accessible from anywhere and not just in the workplace when connected to the company server. The system must be hosted on AWS (to be further discussed below) to ensure the system is available at all times from any location. The database will also be cloud based to ensure full access to the data at all times.

Multi-platform

The system's client side needs to work on all major web browsers (Chrome, Edge and Firefox). To ensure that this is possible we have made sure that none of the technologies implemented are proprietary to a single browser and that any features implemented are tested on different browsers to make sure there is consistency in the display on all browsers.

MVC Framework

One of the system requirements was for the project to follow an MVC framework. As stated above the system does follow a multi-tier architecture design with a heavy emphasis on the MVC architecture design. To ensure this requirement was met the directories for the frontend (View) and backend (Controller) are completely separate. This ensures better maintainability over the front and backends and keeps the design within the MVC framework. The database (Model) is cloud based on AWS RDS as well.

Frontend - Javascript (React)

It was required for us to develop the frontend using Javascript with React, a technology none of us were previously familiar with. The constraint here was simply learning to use a technology which was not too intensive as we already all had experience with Javascript.

Backend - Java (Spring Boot)

It was also required to develop the backend using Java with Spring Boot, another technology we were not familiar with. The constraint here again was learning the technology before development could take place, however we all know Java and which helped the process. Another constraint was for a few of the Spring dependencies, many of them had undergone major updates in the last year, and so there were few resources to help explain and implement the dependencies. There were a lot of outdated examples that were no use to us and did slow the backend development process.

Hosting - AWS

Another requirement was to make use of AWS for our hosting and deployment as well as the cloud database for the project. Once again learning a new technology was an initial constraint. The tier and paid systems for AWS was the biggest constraint we had in the system development. We had to make sure that we were not exceeding the free tier limits and thus incurring costs. To solve this we have made sure our system will fit within the limits to ensure no costs arise for the client. On the database side we have opted for initial development on a local database to test and make sure the system runs and then once the system is complete and ready for deployment we will link it to the AWS RDS database.

Technology Choices

Although our client did specify the technologies they want us to utilize we did look into other choices to make sure the specified requirements were the best for the system we were to develop. Below are the technologies we looked at and the final ones chosen. It should be noted that we did end up using the client required technologies as they were in fact the best choice for the system.

Frontend

React

React is a cross platform development tool that uses Javascript. Our team knows the javascript language and it is easy to use. It is also a very popular web development language so future development of the system would be seamless. A Pro for React is that runs smoothly on the user side due its use of native components, however a Con is that it connects to backend components via a bridge it can run slightly slower.

Flutter

Flutter is another cross platform development tool but it uses Dart, a language none of us are familiar with, nor is it very popular by most web developers and is a Con on our list. It does however have a hot reload feature that allows you to view changes made in real time and thus quicker development. It also tends to create larger components so your project becomes quite large in size and more space is required for hosting.

Ultimately we chose React as we are familiar with Javascript and it is very popular with much more support and resources to utilize.

Backend

Spring Boot

Spring Boot is developed in Java and makes developing web applications and microservices fast and easy through autoconfiguration, an opinionated approach to configuration and the ability to create standalone applications. It also allows for dependency injections that lets objects define their own dependencies that Spring will later inject into them to use their service. As it is coded in Java it was the top choice as we have all worked in Java extensively. Spring Boot also integrates nicely with React and the infrastructure between the tools is easy to set up and understand. It was also suggested by the client and we believe it is the best choice for backend development for this project.

Hosting

AWS

Amazon Web Services (AWS) is the world's largest cloud computing server provider. It is self hosting and provides many flexible solutions that can fit any customer needs. It provides reliable APIs to optimize your infrastructure as you see fit. It is also cheaper than Firebase and provides a pay-as-you-go option for the amount of space and storage you use. It is quite a complex service that provides a lot of different options that can be quite confusing to a new user and some time and research needs to be put into selecting the best option for your system.

Firebase

Firebase is a very popular mobile and web application to suit most hosting options. It is easy to implement and authentication can be set up quickly. It provides machine learning tools to train custom models. The servers are maintained and thus the client does not need to worry about the infrastructure as well. It also supports many cloud storage solutions. It does however have some constraints on the data storage as you cannot transmit more than one key query at a time and due to our system being a Database-centric design this was not ideal. You will also run into costs earlier than you would with AWS.

We ultimately went with our clients recommendations and chose AWS as our hosting option.