Strawberry Prediction System

Academic Year 2022-2023

Campus Geel, Kleinhoefstraat 4, BE-2440 Geel

H1: CodeFarm

Frederik Crauwels, Princewell Baffour Awuah, Derre Evers,

Jordy Boons, Bryant Suiskens, Mohamed Azish

1 TABLE OF CONTENTS

The table of contents provides an overview of our project plan.

2 INTRODUCTION

Outlining the expectations and contextualizing our project plan. This file holds all the necessary information for you to understand our thought process and how you can re-deploy the application in its entirety, as well as provide additional context regarding certain processes.

2.1 Context

The goal of this report is to provide you with a thorough overview and explanation of all the features provided in our proof of concept (PoC). This report will help you in utilizing the same code and deploy the entire application yourself. Each separate deployment will be included in this documentation in order to provide further information about each stack / deployment.

As of now we have three specific areas of implementation:

- **DevOps:** this includes everything related to **security, APIs** and **CI/CD.**
- **AI:** this includes everything related to **Artificial Intelligence.**
- **Application / web development:** this includes everything related to the **front-end of the website**.

2.2 Expectations from the report

At the end of this report you should be able to understand how each specific area of implementation works and functions within the overall picture of the architectural design. You should be able to fully re-deploy the application. It is also important to understand the basics of the application which is our starting point in the requirements chapter.

- **Requirements:** the requirements will guide you through our early thought process and starting position when development started. Here you can find a variety of documents: use-cases, ERD and PBS. They should give you a good understanding of how the end-product should work.
- **Proof of concept:** the proof of concept will guide you through each separate stack. This part will provide more explanation on how to use each stack separately specifically: setup, installation, coding guidance, usage,… At the end of this chapter you should have a thorough understanding of each separate component and how these work together via architectural drawings.
- **Conclusion:** at the end a conclusion will be provided outlining our overall end-result (PoC) and a brief guidance on how to move this project further with additional features, for example.
- **Bibliography:** the bibliography will provide you with additional sources used in the investigation and deployment of our PoC.

3 REQUIREMENTS

Investigative information providing insights in our thought process.

3.1 Use cases and use-case diagram

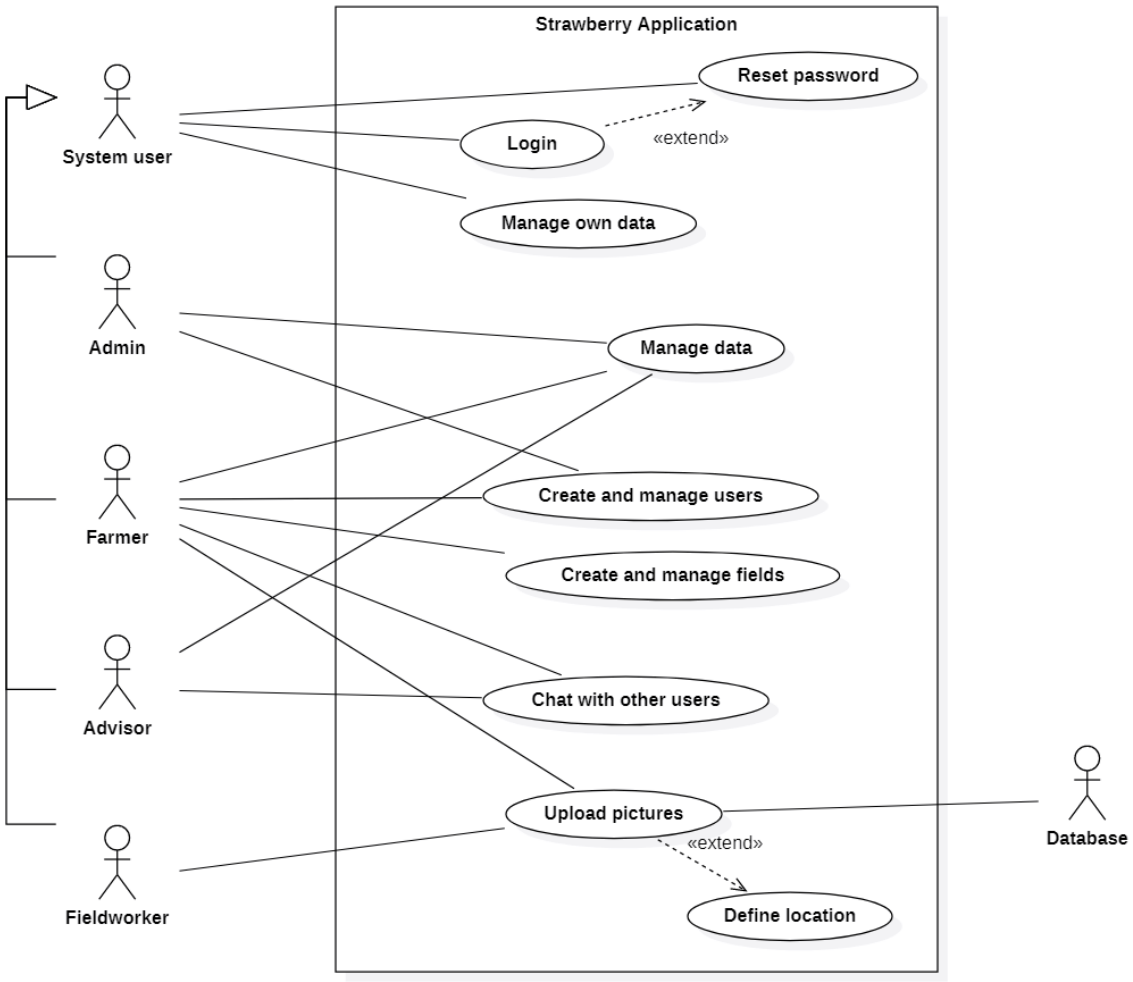
Use cases define a specific scenario in which the application user(s) will interact with a piece of software. This indicates how **useful an application** can be to an end-user. Every single interaction with our application is primarily analysed in order to know how different users will be handling our application.

The use-case diagram is a summary of every single use case we defined. All the users and use cases are visualised in one, single picture.

As we will be building two applications: **web application** and a **mobile application**, we have divided the uses cases amongst two application frameworks.

3.1.1 Web application

Below you can notice the **use-case diagram** for the **web application** summarizing every single use-case in one, single image:



3.1.1.1 System User

These use cases will focus on the **system user**.

Use case: Login

	Description
Actor	System User
Functionality	As a system user, I can login.
Precondition	/
Normal flow	The actor is presented with a username and password field. The actor fills in his credentials to login. When the credentials are correct the system shows the home screen.
Alternatives (not mandatory)	Wrong password: The system gives an error message. The actor can try again to login. Forgot password: The actor can reset his password.

Use case: Reset Password

	Description
Actor	System User
Functionality	As a system user, I can reset my password.
Precondition	An account has been created.
Normal flow	The actor says he want to reset his password. The system gives a form to reset a password. The actor can change his password.
Alternatives (not mandatory)	/

Use case: Manage own data

	Description
Actor	System User
Functionality	As a System User, I can manage my own data.
Precondition	The actor is logged in
Normal flow	The actor can change the filled-in specifications regarding their e-mail, username, …
Alternatives (not mandatory)	/

3.1.1.2 Admin

These use cases will focus on the **admin**.

Use case: Manage data

	Description
Actor	Admin
Functionality	As an admin, I can manage data.
Precondition	The actor is logged in.
Normal flow	The actor can edit or remove data (pictures, field data, strawberry data).
Alternatives (not mandatory)	/

Use case: Create and manage users

	Description
Actor	Farmer
Functionality	As a farmer, I can create and manage users
Precondition	Farmer is logged in.
Normal flow	The system shows an interface where the actor can create, edit and delete users.
Alternatives (not mandatory)	/

3.1.1.3 Farmer

These use cases will focus on the **farmer**.

Use case: Manage data

	Description
Actor	Farmer
Functionality	As a farmer, I can access data of my field.
Precondition	Farmer is logged in.
Normal flow	The actor can edit or remove data (pictures, field data, strawberry data).
Alternatives (not mandatory)	/

Use case: Create and manage users

	Description
Actor	Farmer
Functionality	As a farmer, I can create and manage users
Precondition	Farmer is logged in.
Normal flow	The system shows an interface where the actor can create, edit and delete users.
Alternatives (not mandatory)	/

Use case: Create and manage fields

	Description
Actor	Farmer
Functionality	As a farmer, I can create and manage fields
Precondition	Farmer is logged in.
Normal flow	The system presents the farmer with a list of their fields. The farmer can create a new field or update details of an existing field. The system presents the actor with an updated field list.
Alternatives (not mandatory)	/

Use case: Chat with other users

	Description
Actor	Farmer
Functionality	As a farmer, I can send and receive messages from other users
Precondition	Farmer is logged in.
Normal flow	The system presents the actor with a list of messages if there are any. The actor can select a message and respond to it. The actor can start a new chat by selecting a recipient.
Alternatives (not mandatory)	/

Use case: Upload pictures

	Description
Actor	Farmer
Functionality	As a farmer, I can upload pictures
Precondition	Farmer is logged in.
Normal flow	The system shows a list of picture types. The actor selects a type. The system shows the page where the actor can upload a picture with a list of fields. The actor uploads a picture and selects a field. The system uploads the data to the database.
Alternatives (not mandatory)	/

3.1.1.4 Field worker

These use cases will focus on the **field worker**.

Use case: Upload pictures

	Description
Actor	Field worker
Functionality	As a field worker, I can upload pictures.
Precondition	Farmer gave access to field worker (login).
Normal flow	The system shows a list of picture types. The actor selects a type. The system shows the page where the actor can upload a picture with a list of fields. The actor uploads a picture and selects a field. The system uploads the data to the database.
Alternatives (not mandatory)	Upload pictures directly through the mobile application as this will reduce the chance of having not sufficient (or correct) meta data.

3.1.1.5 Advisor

These use cases will focus on the **advisor**.

Use case: Chat with other users

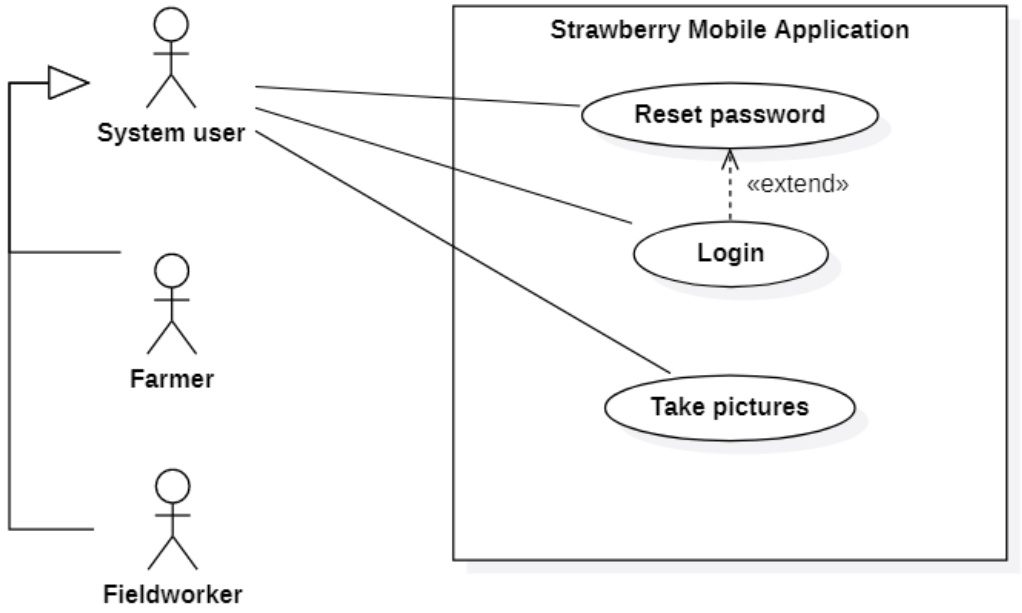
	Description
Actor	Advisor
Functionality	As an advisor, I can chat with other users.
Precondition	Advisor created their own account.
Normal flow	The system presents the actor with a list of messages if there are any. The actor can select a message and respond to it. The actor can start a new chat by selecting a recipient.
Alternatives (not mandatory)	/

Use case: Manage data

	Description
Actor	Advisor
Functionality	As an Advisor, I can access data of my field.
Precondition	Actor is logged in.
Normal flow	The actor can edit or remove data (pictures, field data, strawberry data).
Alternatives (not mandatory)	/

3.1.2 Mobile application

Below you can notice the **use-case diagram** for the **mobile application** summarizing every single use-case in one, single image:



3.1.2.1 System User

These use cases will focus on the **system user**.

Use case: Login

	Description
Actor	System User
Functionality	As a system user, I can login.
Precondition	/
Normal flow	The actor is presented with a username and password field. The actor fills in his credentials to login. When the credentials are correct the system shows the home screen.
Alternatives (not mandatory)	Wrong password: The system gives an error message. The actor can try again to login. Forgot password: The actor can reset his password.

Use case: Reset Password

	Description
Actor	System User
Functionality	As a system user, I can reset my password.
Precondition	An account has been created.
Normal flow	The actor says he want to reset his password. The system gives a form to reset a password. The actor can change his password.
Alternatives (not mandatory)	/

Use case: Manage own data

	Description
Actor	System User
Functionality	As a System User, I can manage my own data.
Precondition	The actor is logged in.
Normal flow	The actor can change the filled-in specifications regarding their e-mail, username,…
Alternatives (not mandatory)	/

3.1.2.2 Field Worker

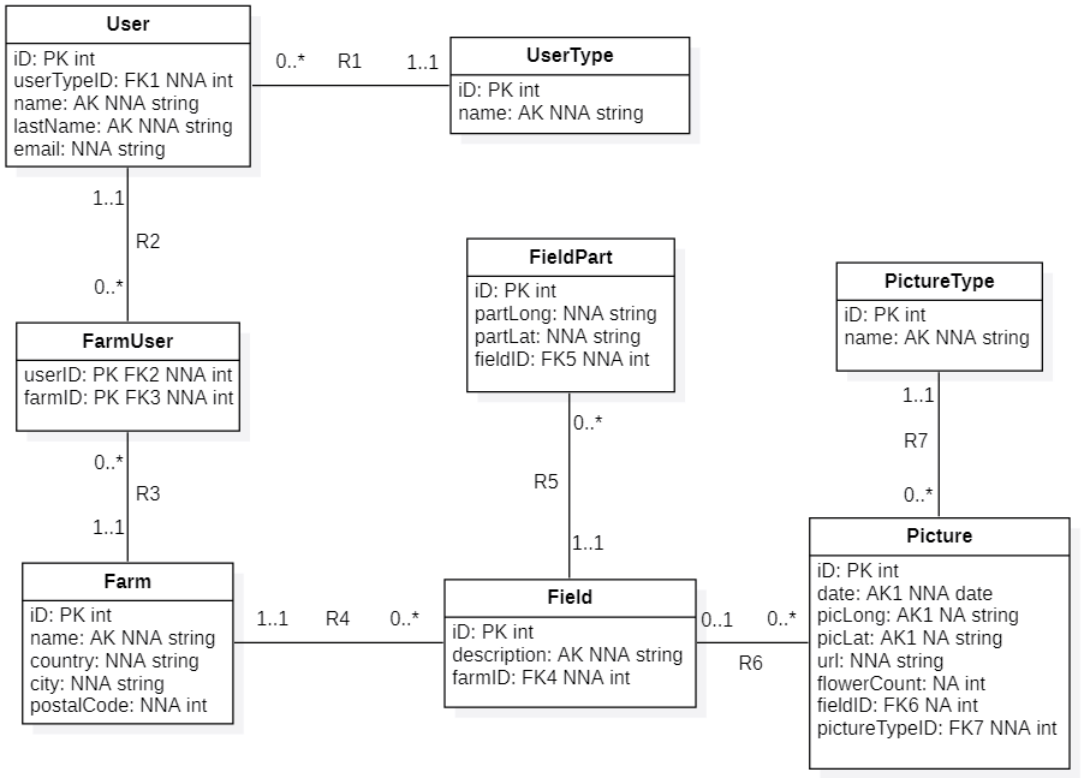
These use cases will focus on the **field worker**.

Use case: Upload pictures

	Description
Actor	Field worker
Functionality	As a field worker, I can upload pictures.
Precondition	Farmer gave access to field worker (login).
Normal flow	The system shows a list of picture types. The actor selects a type. The system shows the page where the actor can upload a picture with a list of fields. The actor uploads a picture and selects a field. The system uploads the data to the database.
Alternatives (not mandatory)	Upload pictures directly through the mobile application as this will reduce the chance of having not sufficient (or correct) meta data.

3.2 ERD: Entity Relationship Diagram

UML (Unified Modelling Language) is a modelling language between a variety of tables. This diagram shows you how our **database** will be drawn in a high-level overview. This diagram will therefore be used to **create our database tables**. This specific diagram is called an ERD: **E**ntity **R**elationship **D**iagram, providing a thorough overview of the data and tables we will be utilizing in the realization phase.



3.2.1 Must-haves

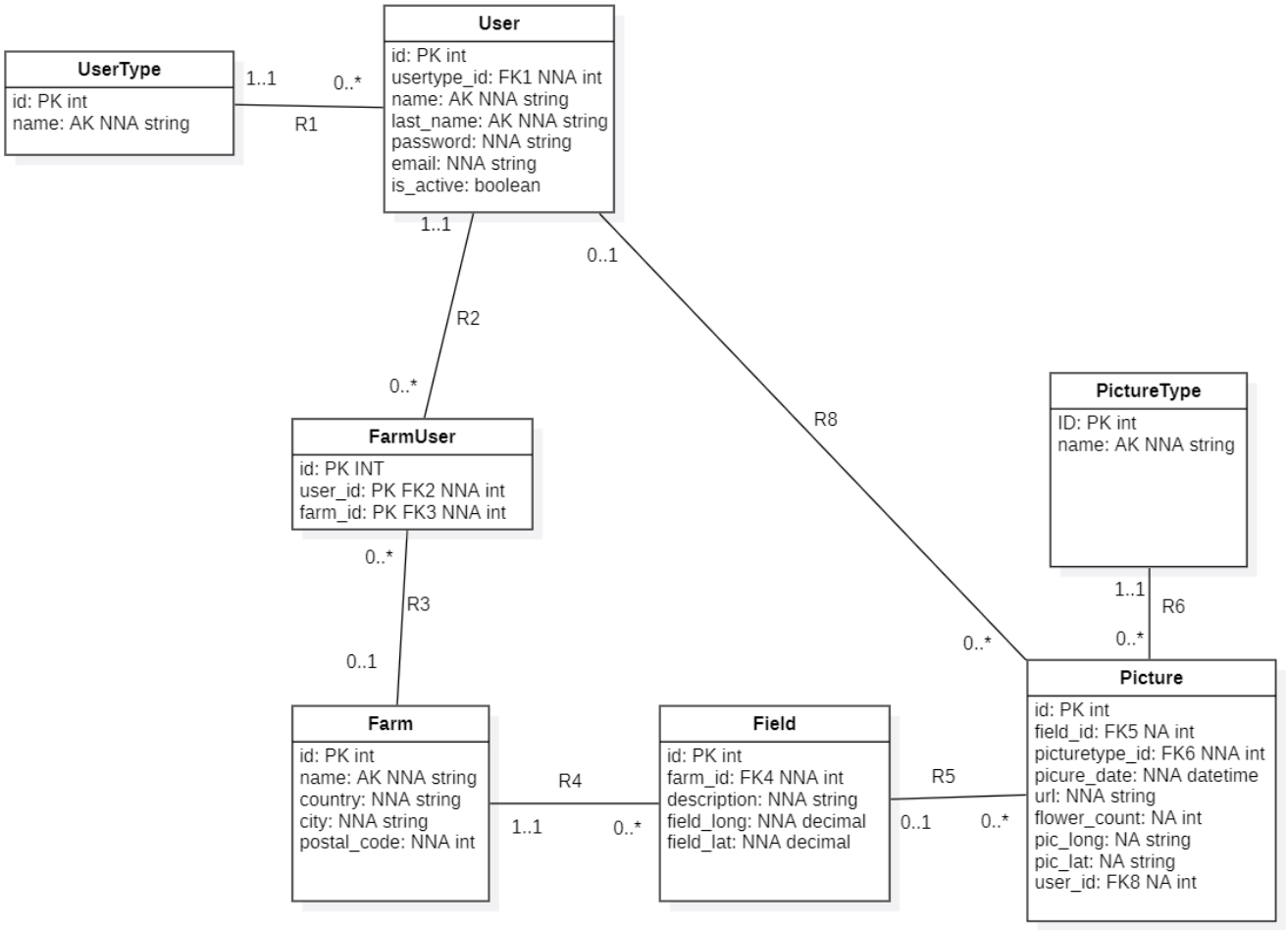
As we have defined clear must-do's and should do's, the primary tables include the following:

- **Farm:** the farm table will include general details about the farm. Where is it located? What is the name of this farm?
- **Field:** the field table will include specific details about the field. The name of the field can be taken up into the description. For example: "Field in the north".
- **FieldPart:** the FieldPart table will include important metrics about the location of the field according to longitude and altitude metrics. The EXACT location of a field. As there could be multiple points of interest to indicate the correct (entire) location of a field – multiple parts / field can be defined.
- **Picture:** the picture table will include all information about a picture. Location, AI data (flowerCount), date and URL (Amazon S3 bucket will have a specific URL for each picture, indicating the correct file / picture).
- **PictureType:** the PictureType table will include which type of picture you are uploading. Drone images or regular images?

3.2.2 Changes during development

As the above table describes our **investigative ERD model** – additional tables have been added and some tables have been changed.

- **Attributes:** some attributes have **changed** accordingly. Mostly it is naming convention changes, some have attributes added.
- **FieldPart:** while this could be used with the **satellite AI model** – in our current project this table had no use. The reason why it could be used by the satellite AI model is when analyzing the smaller satellite pictures – the data can be inputted one-by-one in this table. Since one big satellite picture is split up in multiple smaller pictures – a new table is absolutely required, which FieldPart could take up in the future.
- **Picture:** a new relationship with User has been established because we required a link towards the user id for each picture taken. This way logging can determine how many pictures a user has uploaded, for example.



Logging is not part of the final solution – but can be used for future implementation to log the user actions (who performed which action at a certain time? Upload pictures? Create or delete fields / farms / users?).

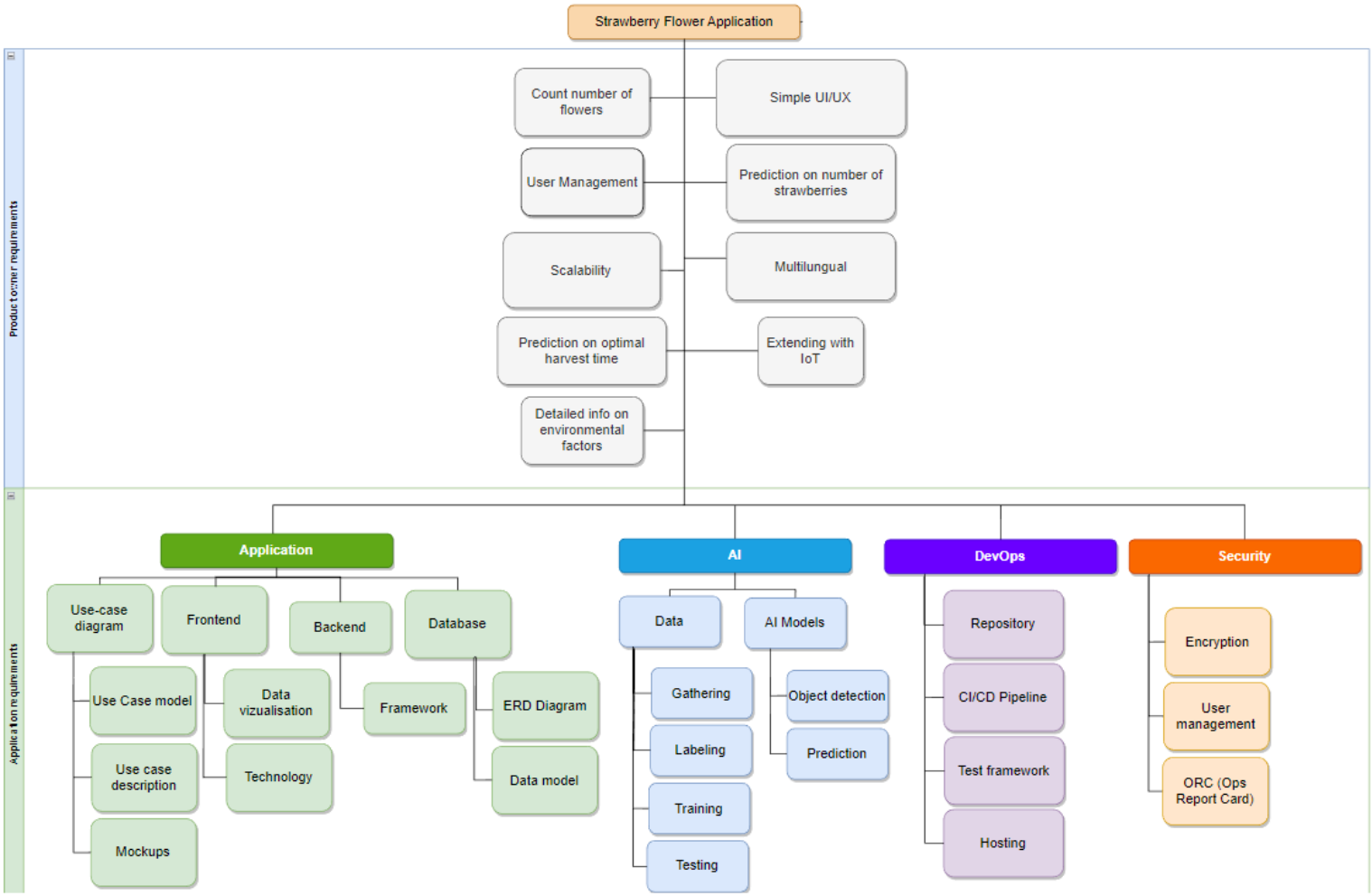
3.3 PBS: Product Breakdown Structure

The PBS offers an overview of all the specific requirements for our application / project. As we have used this PBS during multiple phases of the project by now, the below PBS is a modified version of the previous one.

Certain categories were changes (green area), as well as the product owner requirements (blue area).

All elements have been scaled-down to four theme's:

- **Application:** the focus of the application part is dedicated to creating use-case diagrams, mockups, frontend, backend and database diagrams.
- **AI:** the focus of the AI part is dedicated to AI modeling and making sure the AI can gather, label, train and test with data (pictures).
- **DevOps:** DevOps will focus on the back-end part of the application. Code repository, CI/CD pipeline, test framework and hosting the web application.
- **Security:** security will focus on keeping the application and data safe and secure. The focus theme's are encryption, user management and ORC.



In our final solution we have been able to fulfil at least **70%** of the requirements. Below are the requirements which were **not meant** and the **reason behind this:**

Product owner requirements:

- **Prediction on optimal harvest time:** as this was not a **must-have,** there was not enough time to look into this feature. It would definitely add tremendous benefits if implemented in the future.
- **Detailed info on environmental factors:** as this was not a **must-have,** there was not enough time to look into this feature. It would be **ideal** when **comparing results** if you have obtained further environmental factors: weather conditions, humidity, warmth,…
- **Extending with IoT:** while the primary goal was to provide a solution able to scan both regular pictures and satellite pictures, adding IoT possibilities was another requirement. This extension will allow the application to work indoors as location tracking is not easily done there – and could be automated. As the initial two requirements have taken up most time during the project – this feature did not make it into development.

Application requirements: DevOps & Security

- **Test framework:** as this was not a **must-have**, there was not enough time to look into this feature. Testing is ultimately split up in two parts:
  - **DTA environment:** while a test environment was provided as a separate development environment, testing the code itself automatically via a CI/CD, for example, was not implemented. Thus partially we have provided a solution for testing – but this is not automated testing within a CI/CD or code.
  - **Testing framework:** the primary reason why this feature **did not succeed**. There are no built-in test frameworks deployed. API testing or CI/CD code testing has not been implemented but could be an ideal feature to make sure the application is working sufficiently.
- **ORC (Ops Report Card):** report cards can help in determining if the overall application is secure. While some basic security implementations have been provided – advanced auditing has not been implemented.

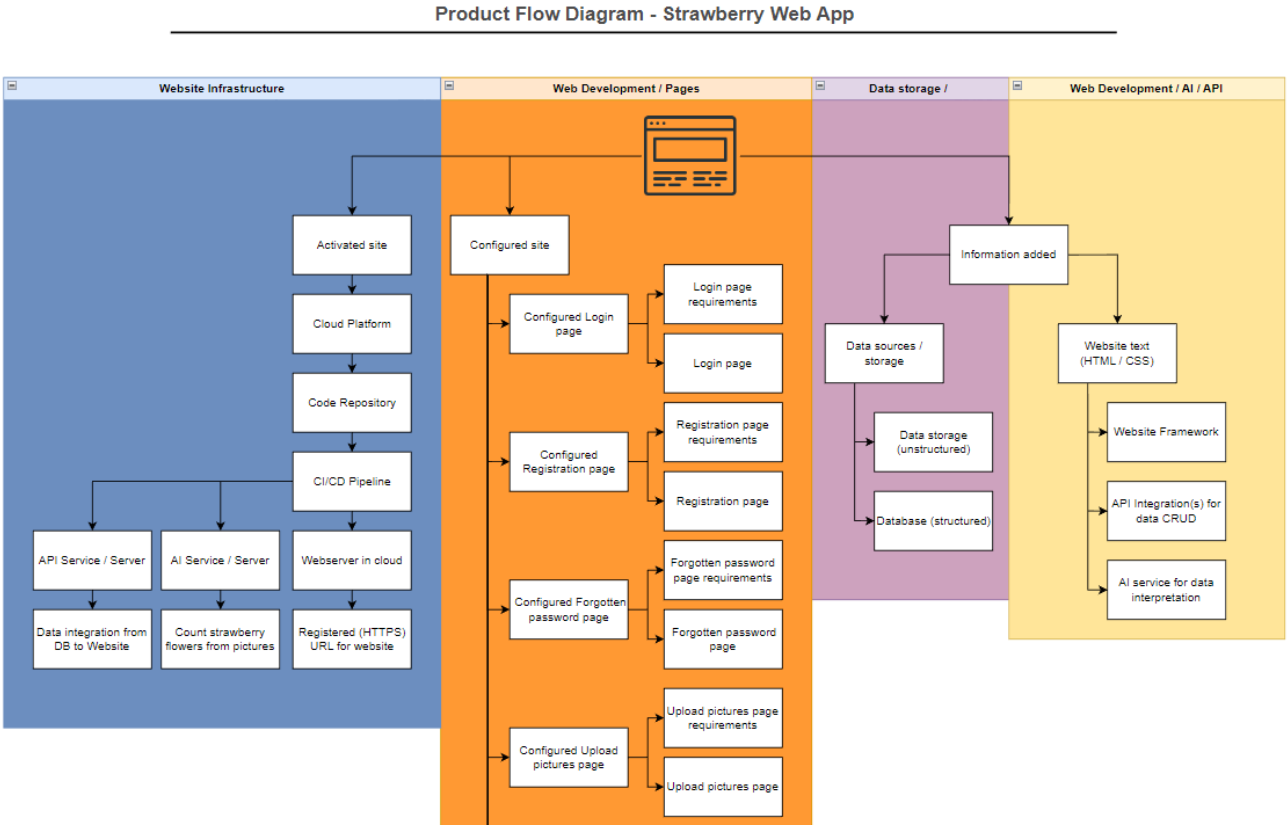
3.4 PFD: Product Flow Diagram

For each part of the application a PFD is provided. This **Product Flow Diagram** provides an overview of what is required for each product that is in development. For now, the primary focus is on **two products:**

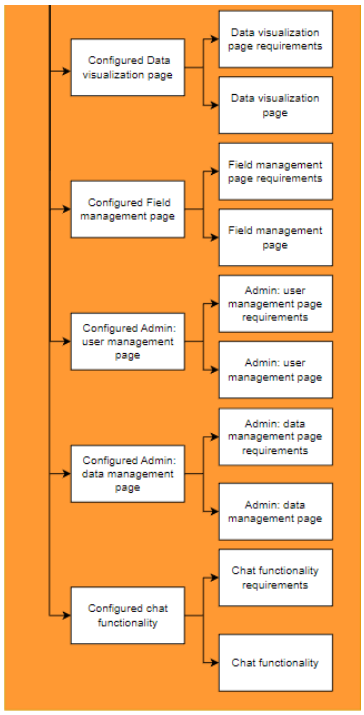
- **Website application:** a website application is visible via any modern day web browser. This web browser can be accessed via a variety of devices such as laptops or smartphones.
- **Mobile application:** a mobile application is visible via any smartphone. This application needs to be published to the android store or apple store, or can be exported as a .APK.

A product flow diagram is deferred from the **PRINCE2** framework which helps to achieve a better project management. The focus of PRINCE2 is to divide projects into controllable stages.

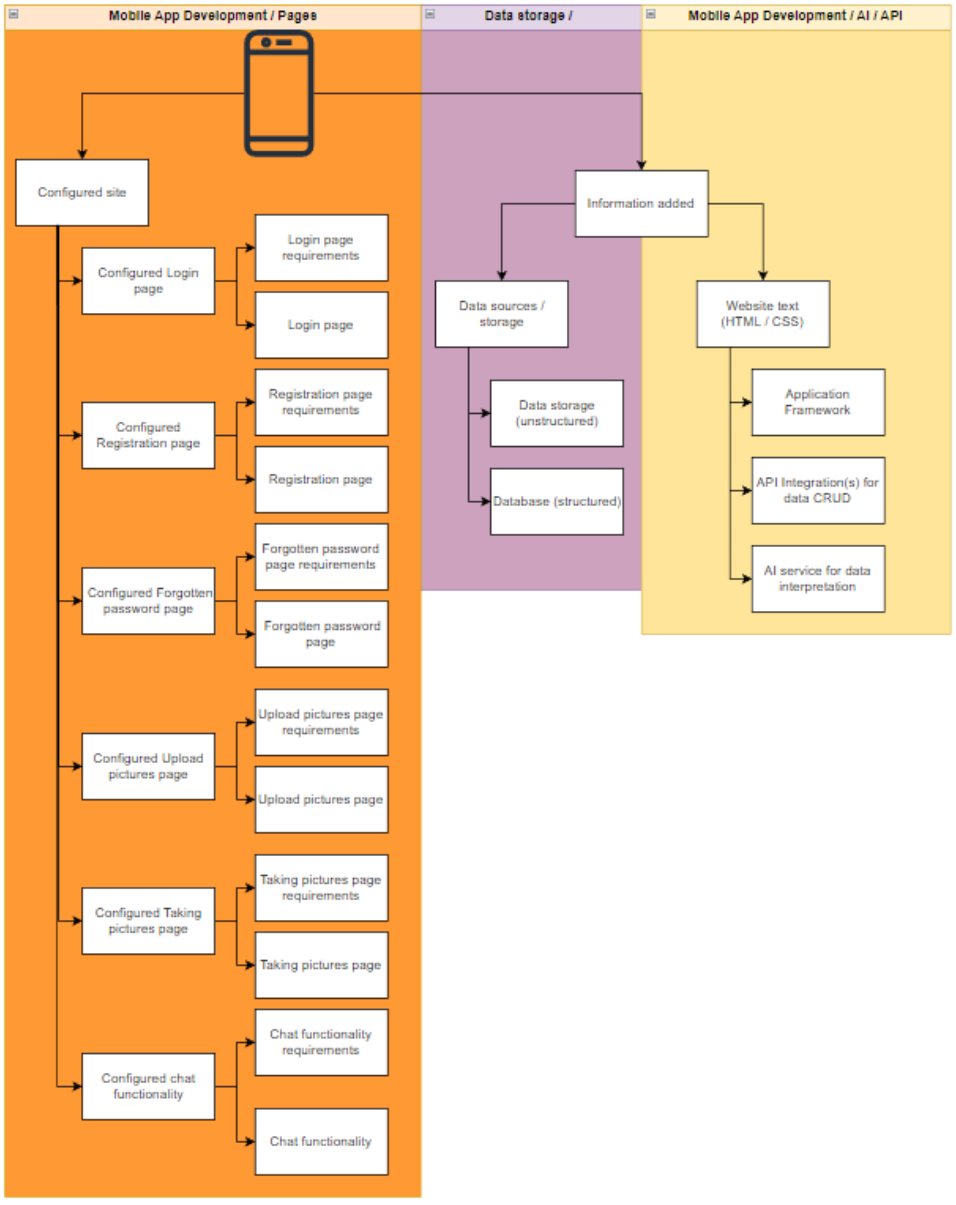
Therefore, two diagrams have been created with specific requirements for each application.



*On the next page you can find an additional part of the **web application PFD.** This part includes additional Web Development / Pages aspects.*



Product Flow Diagram - Strawberry Mobile App



3.4.1 Progressive Web Application (PWA)

Due to the need to have the application work in both a desktop and mobile environment, we have opted to go with a React-based PWA in order to ensure ease of maintenance and code parity of the two platforms. For this PWA we have outlined several requirements. The requirements are further divided into **must-haves** and **should-haves**.

3.4.1.1 Must-haves

The must-haves concentrate on creating a **MVP: Minimum Viable Product.**

This includes:

- **Infrastructure:** the infrastructure revolves around creating **servers** and **services** which make sure the web application can effectively run.
- **Web development:** web development revolves around creating **visual** representation(s) of the web application. The goal of this part is to achieve a variety of pages that can interact with the data source, APIs, AI service and infrastructure.
- **AI:** AI revolves around creating a service that can effectively count strawberry flowers from a picture which is being uploaded via the front-end webpage into a backend data store. This AI script will generate data which need to be stored in a database / data source.
- **API integration(s):** API integration(s) revolve around creating new data and managing existing data. APIs create an opportunity to build micro services which can interact with any kind of integration in the future. The current build focusses on integrating with a web application and mobile application. APIs can display data retrieved from a data source.
- **Data sources / storage:** the data sources / storage revolve around creating a storage opportunity for all data involved in this project. Primarily the focus lies on storing unstructured data and structured data;
    - **Structured data:** AI service provides data from pictures. This data needs to be stored in a database and later on be displayed on the web application.
    - **Unstructured data:** the primary source of structured data comes from unstructured data – pictures. The pictures need to be uploaded on the web application and stored for later use (AI or API).

Once the above parameters are met a MVP is effectively created.

Should-haves go beyond the scope of the MVP, creating an opportunity for a more secure web application and user permissions.

- **Infrastructure:** the infrastructure layer does not need additional requirements for should-haves such as user permissions. The additional factor that needs to be looked into is a WAF. A Web Application Firewall (WAF) is essentially the bread and butter for a modern day web application in the cloud. Without a WAF a web application is essentially defenceless in future cyberattacks and data breaches.
- **Web development:** the web development layer requires additional pages to be made which create an opportunity for end-users to register, login and change credentials.
- **API Integration(s):** the API layer creates an opportunity to interact with the data. This is called CRUD: Create, Read, Update and Delete. In order to create new users, or update credentials, API integration(s) are created to facilitate user permissions.

- **Data sources:** the data sources, especially the databases (structured data) need to be modified with user data. The as-is construction of the data warehouse of data storing solution can be used in this case.

3.4.1.2 Mobile application

The mobile application is essentially a trimmed-down version of the web application which focusses on **taking pictures** and **uploading pictures**. It is not the focus to work on additional parameters such as data visualization or expansive dashboarding, for example.

- **App development:** app development revolves around creating a mobile application that is suitable to achieve the must-haves for a mobile application MVP.
- **Data sources:** the data sources from the web application will be used to store pictures (unstructured data) and store additional data (structured data) created by the AI scripting, for example.

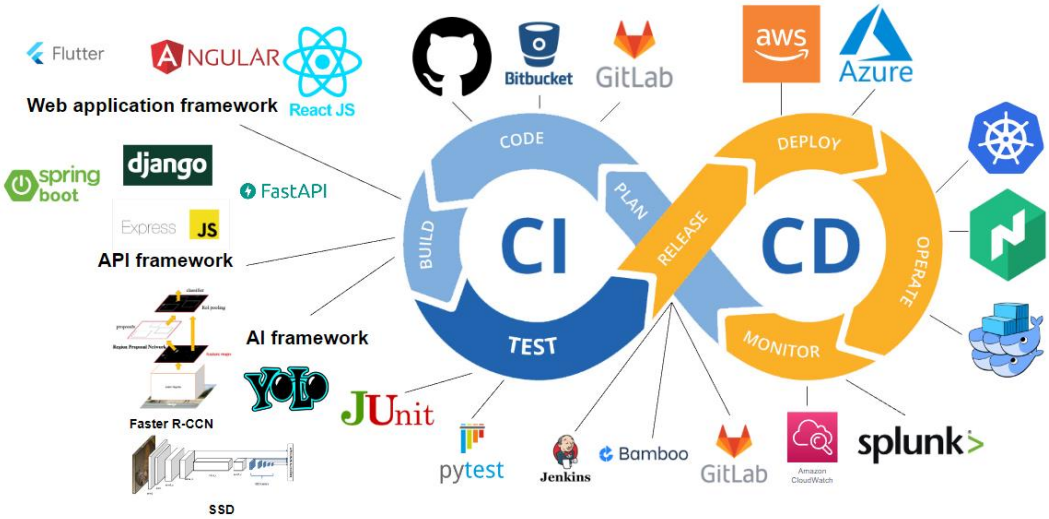
The minimum viable product for the mobile application revolves around creating a simple application that is able to create pictures, upload pictures and work as an easier way to include **location data** from within the application, opposed to providing strict measurements for the web application for uploading pictures.

3.5 Solution decision

Since we have determined the preferred solutions according to WDMs and specific scoring – we can finally determine our full architecture and solutions. In this chapter you will be able to read about our architecture and every component individually.

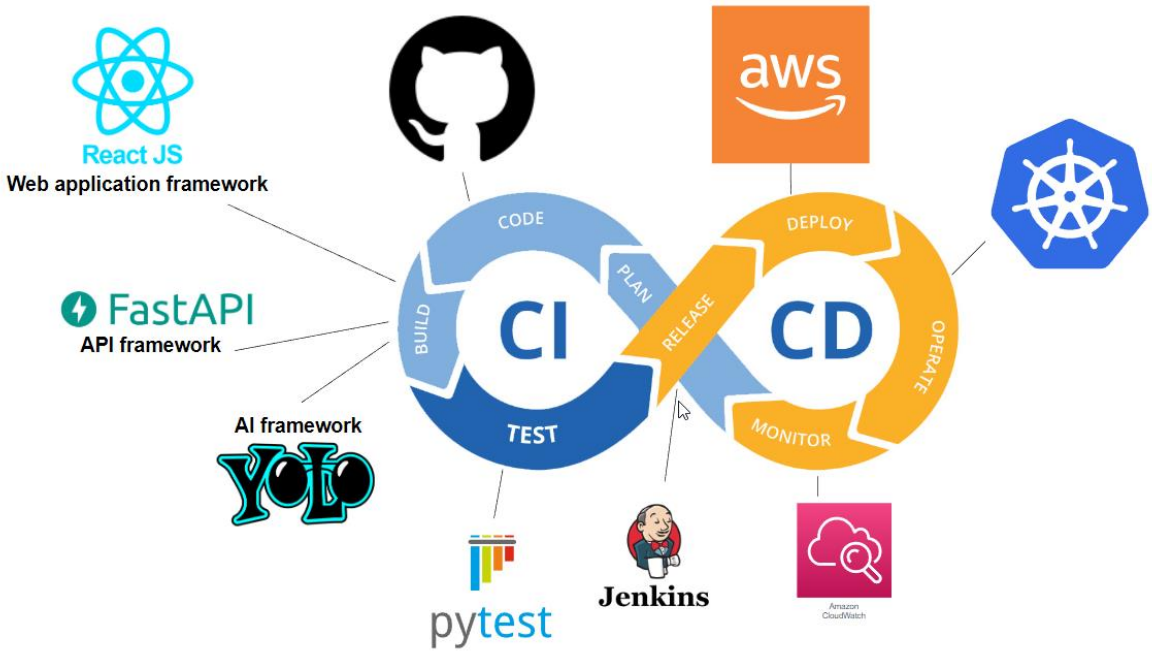
3.5.1 CI/CD

CI/CD stands for **continuous integration / continuous deployment**. It combines both aspects in a never-ending cycle in order to be able to integrate and deploy a final application. Since we have determined our solutions based on the final WDM scoring – most solutions can be summarized in one picture:



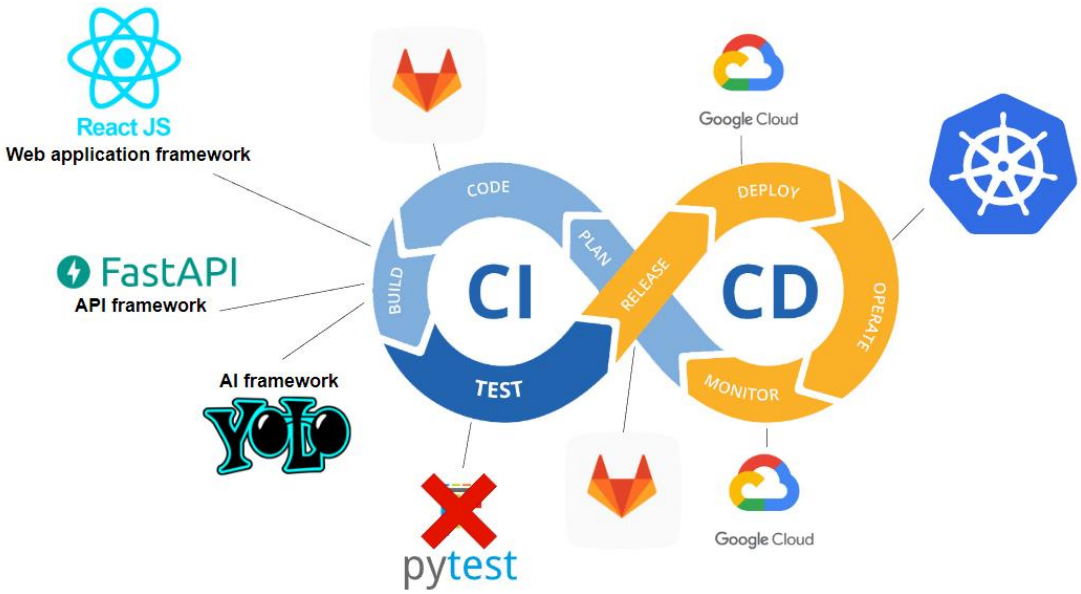
More details about certain pictures / tooling can be found in chapter 1.5 solution analysis.

Once we take a look at the final scoring – we can determine our final solutions which will be used during the next phase to build a complete application:



3.5.1.1 Realization phase changes

While we originally intended to implement the solution provided on the previous page – a significant amount of changes has been made with regards to DevOps.

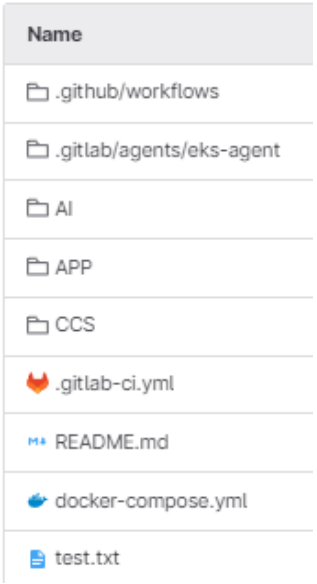


- **Code:** originally we intended to include the stack on GitHub. While his is perfect and mostly free – our release solution also changed. Since it can serve both as a code repository and CI/CD we decided to switch entirely to GitLab instead.
- **Release:** while Jenkins is a good solution and could be used as the final solution, GitLab would make it easier both from a code as a release perspective. This choice made the most logical sense since we can combine both in one.
- **Deploy:** AWS is a perfect solution – but the environment we were using was severely limited due to having a student account. This limited us in providing a thorough solution with the full capabilities of a cloud provider. When requesting an official account – this took too long, thus we have chosen to make a switch from Cloud Provider. GCP offers out-of-the-box full access accounts with 280$ of test credit.
- **Monitor:** similar to the last point – we changed cloud provider due to restrictions. This means monitor has also changed towards GCP, which is done internally by GCP logging.
- **Test:** while we anticipated to include a full CI/CD solution – we have not provided a testing framework such as Pytest. Due to time constraints testing was not on the agenda for the MVP (minimum viable product) and has therefore not been implemented further.

3.5.1.2 GitLab

Gitlab is our **code repository** and **CI/CD** solution. The files provided in the hand-over ZIP file are essentially all you need to start deploying the above provided solution. Even in this case you have two options for deploying the proof of concept.

- **Docker / okteto cloud:** the folder has a **docker-compose** file which can be used to deploy a staging environment. Essentially this is the complete build as is – just on a different cloud platform.
- **Kubernetes / google cloud platform:** the folder has **all the components** needed to deploy to google cloud platform. Specifically the **.gitlab-ci.yml** file has everything you need in order to deploy the solution without any manual interventions needed.



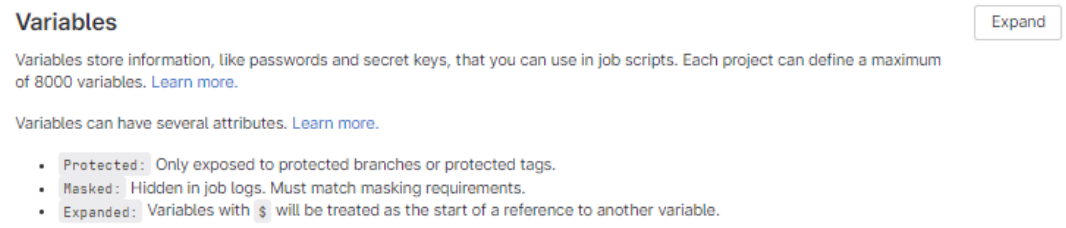
In Gitlab the only requirement you need to provide is the possibility for a **GitLab runner**. This can either be installed manually into your cloud solution – or locally.

More information on providing a personal GitLab runner can be found via the official documentation: https://docs.gitlab.com/runner/install/

When you do not want to provide a **GitLab runner** you can still use **shared runners** – which do the job just fine as well. You do have to take into consideration the **shared runners** have a **limited amount of run time available**. Once you exceed this threshold you will have to **pay for additional minutes**. *Therefore a personal GitLab runner is advised – but not required in the initial stage.*

The only requirement for an out-of-the-box working CI/CD solution to google cloud is the requirement of **variables**.

In GitLab you can set variables by selecting the **Settings** tab -> **CI/CD** -> **Variables** section.



Here you can configure all the required variables for this deploying. These variables need to be **customized** according to **your cloud environment**.

Create a new GitLab project, import all the files and add custom variables accordingly.

Required variables:

Type	↑ Key	Value	Options	Environments
Variable	db_name	*****	Expanded	All (default)
Variable	FLASK_APP_FASTAPI_URL	*****	Expanded	All (default)
Variable	FLASK_APP_FIREBASE_ID	*****	Expanded	All (default)
Variable	HOST	*****	Expanded	All (default)
Variable	KUBERNETESACCOUNT_TERA_PROJECT	*****	Expanded	All (default)
Variable	OKTETO_TOKEN	*****	Expanded	All (default)
Variable	OKTETO_USERNAME	*****	Expanded	All (default)
Variable	PASSWORD	*****	Expanded	All (default)
Variable	PORT	*****	Expanded	All (default)
Variable	REACT_APP_APP_ID	*****	Expanded	All (default)

When you insert a variable you need to **format it in this way:**

<KEY_NAME>=<VALUE>

Example: FLASK_APP_FASTAPI_URL=https://api.vitofruitcounter.org/

For every single variable defined here you need to provide the **exact same naming convention** as described in the two screenshots. These specific **key names** are coded within the **.gitlab-ci.yml** file.

Variable	REACT_APP_FASTAPI_URL	*****	Expanded	All (default)
Variable	REACT_APP_FIREBASE_ID	*****	Expanded	All (default)
Variable	REACT_APP_FIREBASE_KEY	*****	Expanded	All (default)
Variable	REACT_APP_FLASKAPI_URL	*****	Expanded	All (default)
Variable	REACT_APP_GOOGLEMAPS_KEY	*****	Expanded	All (default)
Variable	REACT_APP_MEASUREMENT_ID	*****	Expanded	All (default)
Variable	REACT_APP_MESSAGING_SENDER_ID	*****	Expanded	All (default)
Variable	SERVICE_ACCOUNT_KEY	*****	Expanded	All (default)
Variable	SERVICEACCOUNT_TERRA_PROJECT	*****	Expanded	All (default)
Variable	USERNAME	*****	Expanded	All (default)

Once the above variables have been defined you simply push new code, or create a new CI/CD action and you deploying will be rolling out on GCP. Thanks to the **terraform automation** you do not have to provide more settings.

3.5.1.3 Staging environment – docker – okteto

In order to deploy the **staging environment** you simply need to create a **okteto account** and you are already ready to go. The provided **docker-compose** file will automatically deploy the stack from GitLab to okteto cloud.

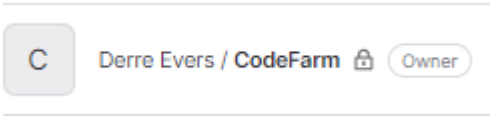
Once you have created your okteto account you need to create a **new dev environment** and add the **following parameters:**

https://<username>:<access_token>@gitlab.com/<username>/<repository_name>.git

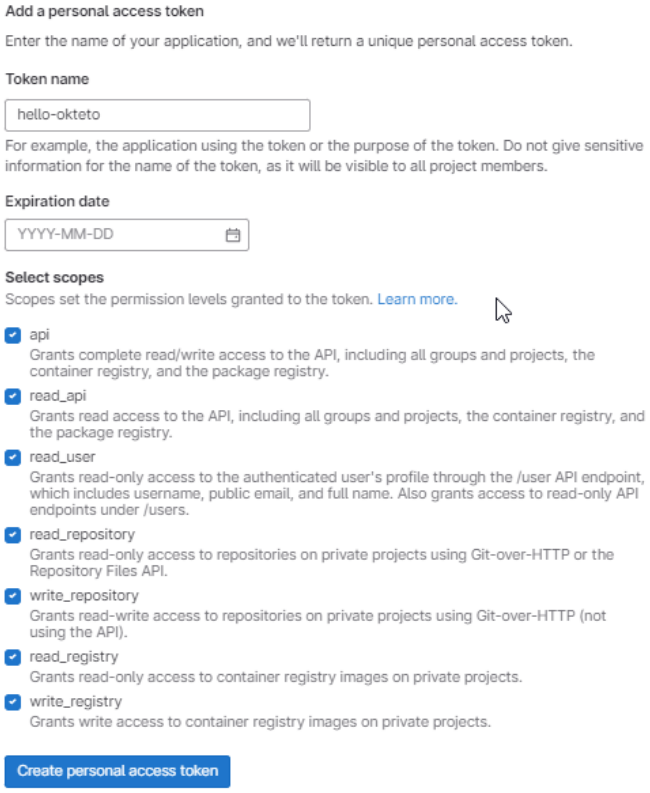
An example from our project could be the following:

https://<username>:<access_token>@gitlab.com/derre-evers/CodeFarm.git

In which our **project** had the following **username** and **repository name**:

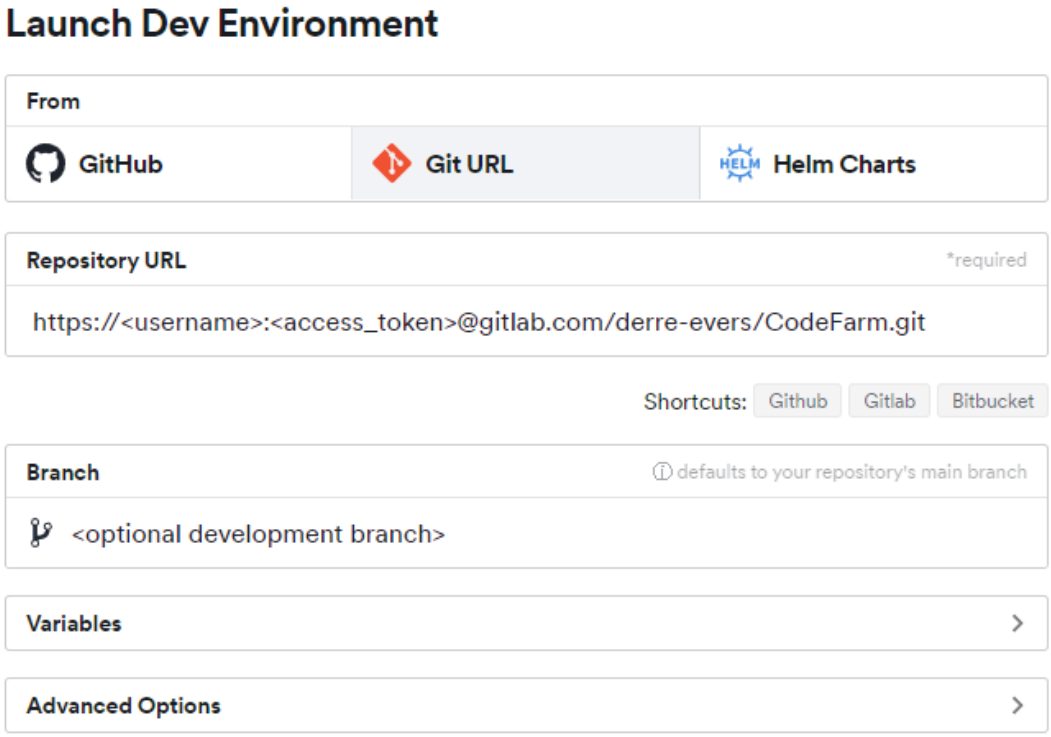


An **access token** needs to be provided in order to be able to push and pull from GitLab. Simply revert back to your **user account settings** by clicking on the user icon in the **top right corner** -> **edit profile**. Now click on the **left side** of your screen on **Access Tokens** and simply create a **new access token.**



Select **all scopes** to guarantee for enough access rights to deploy.

Now simply go to the **okteto portal** and **launch a new dev environment**:



You can specify a **custom development branch** as well, if you are using another branch.

Now you deployment will automatically **deploy the entire GitLab stack** as if it was a production stack towards Okteto cloud. *Do mind again: make sure to use custom variables in order for the applications to work properly. This is why a custom development branch is advised.*

3.5.1.4 Terraform

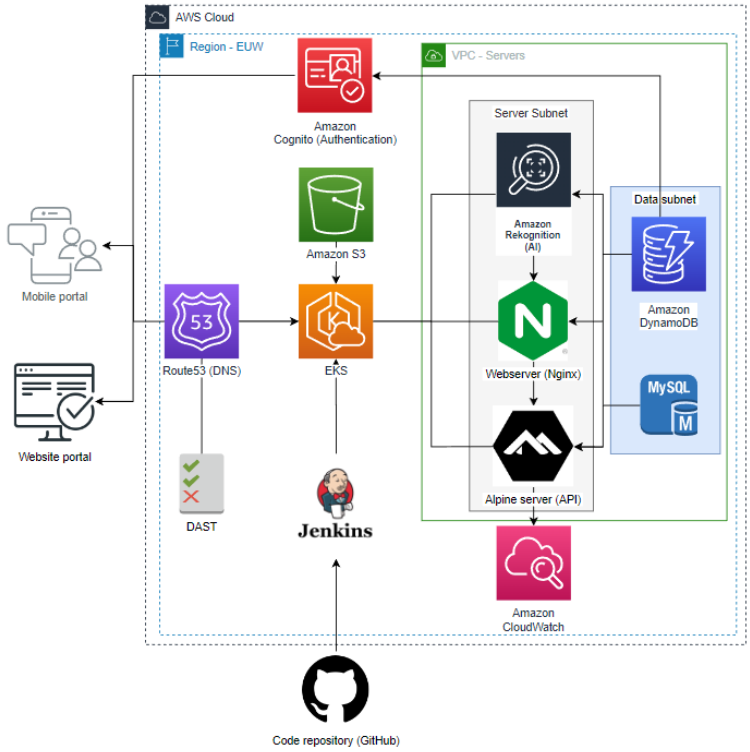
Terraform is used to automated our **infrastructure** at the lowest possible stage. All required terraform files have been provided in the folder and should only be **imported directly into GitLab** in order for them to work.

No additional configuration aside variables is required in order for terraform to do its thing.

3.5.2 Architecture

Not all solutions are mentioned in the CI/CD chapter. Since a variety of solutions are also loosely connected – via AWS, for example, other solution(s) will be visualised in our architectural overview. Most of the proposed **frameworks** will also have to be managed on a **server** (in the cloud). These specific software components are very important to consider.

The architecture can be summarised as the overall DevOps solution which will be built in **AWS,** our **deploy** solution:



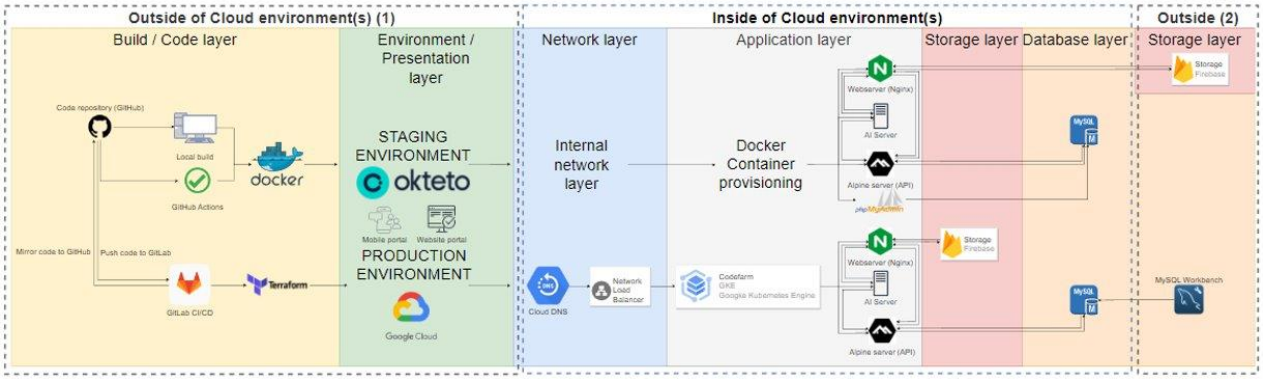
Additional factors in-scope for our project are:

- **Route53:** Route53 is an Amazon AWS solution providing DNS routing in the cloud.
- **EKS:** Elastic Cloud Kubernetes (EKS) is essentially **Kubernetes** in the cloud (CI/CD: Operate).
- **Amazon S3:** S3 buckets provide a storage solution for **unstructured data** (pictures / images) and a handy **URL** for better integration(s).
- **Amazon Cognito:** Cognito provides **user management** / **authentication** via Amazon AWS.
- **Amazon Rekognition:** Rekognition is an **AI solution** provided via Amazon AWS. It is essentially the doorstep for our Ai integration.
- **Webserver – Nginx:** Nginx is a webserver solution providing a software layer to deploy our **React JS** framework to (CI/CD: build).
- **Alpine server – API:** Alpine is a specific OS providing a software layer to deploy our **API** framework **FastAPI** (CI/CD: build).
- **Amazon DynamoDB:** DynamoDB is a NoSQL database offering a single table of data stored as key value pairs for simple data structures. This can be used, for example, for the chat functionality.
- **MySQL / Amazon RDS:** Amazon RDS offers a relational database model (MySQL) required for our UML diagram (our defined tables).
- **DAST:** Dynamic Application Security Testing is an additional security measurement providing automated security during / after the deployment.

3.5.2.1 Realization phase changes

While we originally intended to deploy to **AWS** – during the development phase we have made significant changes and have added a completely new environment: a DTA environment.

Our original scope has changed tremendously and has a variety of additional solutions added. You can find the original image in the folder provided.



A number of changes have been made:

- **Cloud provider:** while originally we intended to go for AWS – due to restrictions we have changed cloud provider. The new solution is built on top of Google Cloud Platform (GCP) and Okteto Cloud, which is our **staging environment**.
- **Staging / testing environment:** in theory this could be another proof of concept. It is deployed in the cloud and running / working as such. By providing both a staging environment and production environment we doubled our attempts at displaying the capabilities of our proof of concept. Since staging in an actual environment (not just localhost deployments) is much more handy, the staging environment was the perfect solution to solve this problem. **Production** is made with **Kubernetes** – and is a tad harder to test, while **staging** is made with **docker** – making it easier for testing. In the end you simply change certain **parameters** in the code (URLs) once deploying to production.
- **Storage solution:** in the CI/CD chapter we haven't even looked into storage solutions yet. Originally this was intended for AWS – but as we changed cloud provider, so changed the storage solution. In the new deployment we utilize **FireBase** – which is a Google solution for a variety of options. In our case it is provided as a **file storage solution**. The combination of GCP and FireBase makes it easier as well: budgeting can be completely integrated into GCP opposed to have an entirely different platform / service provider.
- **Terraform addition:** terraform makes sure not only our **machines** are built automatically – also our **environment as a whole**. This means even deploying the entire stack to GCP doesn't have to be done manually anymore. By providing this possibility – we essentially provide a full deployment of every application and the underlying environment. *Please do note while we want to achieve full automation – a very limited scope of aspects is still manual.*

Essentially the solution itself has not changed a lot – primarily the new DTA addition, storage solution and full automated deployment are major features added to the proof of concept.

3.5.3 Google cloud platform

Google Cloud Platform (GCP) is our cloud provider – and as described in a previous chapter – you only need to create a **new account** in order to be able to deploy to GCP. Do make sure you create custom variables!

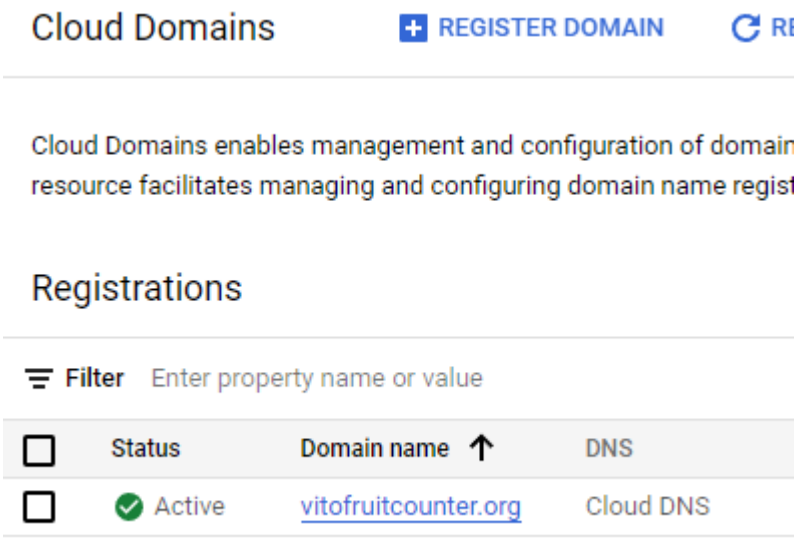
There are two specific manual requirements:

- **Domain:** you need to have a domain.
- **Database:** you need to create a database and whitelist the correct asset accordingly.
- **FireBase:** you need to have a FireBase account with a storage solution.
- **Service account:** you need to create a service account. For testing purposes you need to provide the owner role. In production you should work with **principle of least privilege**. You can provide specific service roles for every specific job.
- **Deploy token:** you need to obtain a deploy token from GitLab which is a pre-defined variable.
- **Load balancer IP:** you need to create additional DNS entries.

3.5.3.1 Domain

Since you already have the possibility to purchase a **custom domain within Google Cloud Platform** – this becomes very easy.

Simply search for **Cloud Domains** and **create a new domain**. This will automatically be subtracted from your **GCP budget**.



Once this is set up you need to change additional **DNS configuration**. Search for **Cloud DNS** and simply adjust the parameters accordingly:

DNS name	Type	TTL (seconds)	Routing policy
ai.vitofruitcounter.com.	A	300	Default
api.vitofruitcounter.com.	A	300	Default
vitofruitcounter.com.	SOA	21600	Default
vitofruitcounter.com.	NS	21600	Default
vitofruitcounter.com.	A	300	Default
www.ai.vitofruitcounter.com.	CNAME	300	Default
www.api.vitofruitcounter.com.	CNAME	300	Default
www.vitofruitcounter.com.	CNAME	300	Default

- **A records:** A records are records that link **directly to your load balancer**. This will essentially transform the **IP address** to a **domain name**. Look for the tab **load balancing**, save the **ip address** and create custom A records for your domain and the load balancer IP address. The load balancer is **automatically created** from your deployment.

Load balancer:

ab57b431e0c9540c988cc2125135d964

Frontend

Protocol	IP version	IP:Port	Network Tier
TCP	IPv4	34.140.235.184:80-443	Premium

A record:

DNS name	ai.vitofruitcounter.com.
Type	A
TTL(seconds)	300

Routing data

Filter Filter datas

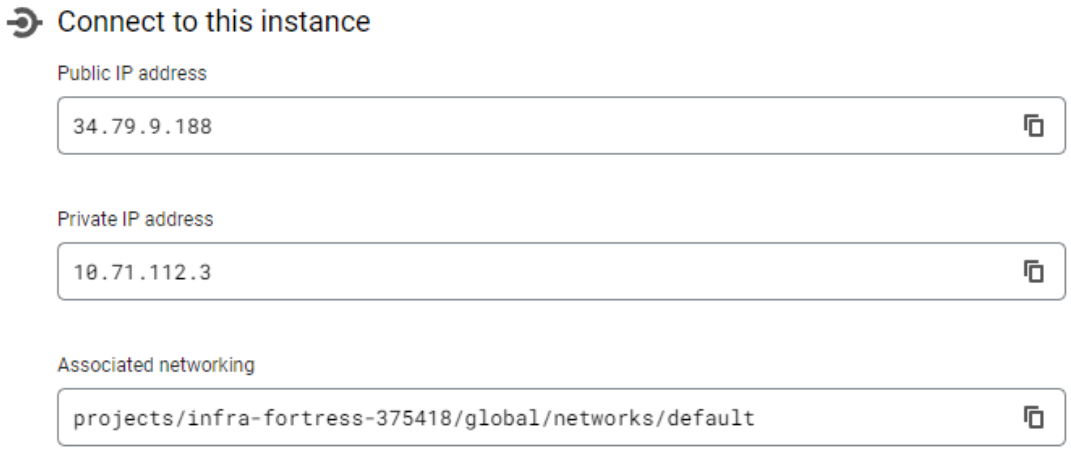
Data
34.140.235.184

- **CNAME records:** CNAME records are required in order to obtain **more custom domain name** aside from your original domain name. simply add **www** to the front of your website in order to redirect towards your original domain without www. This is essentially what a CNAME record does.

3.5.3.2 Database

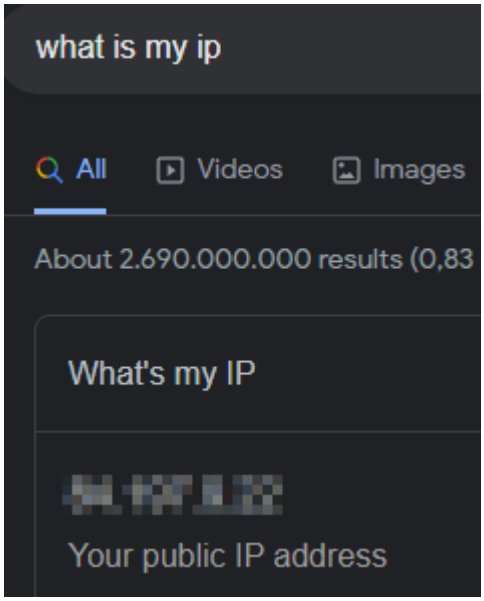
The database creation is also a manual process. *Simply adjust the GitLab variables in order for your services to connect to this SQL server*.

This **public ip address** can be used by your **Kubernetes stack** in order to **connect to the database**. Add this variable in GitLab!

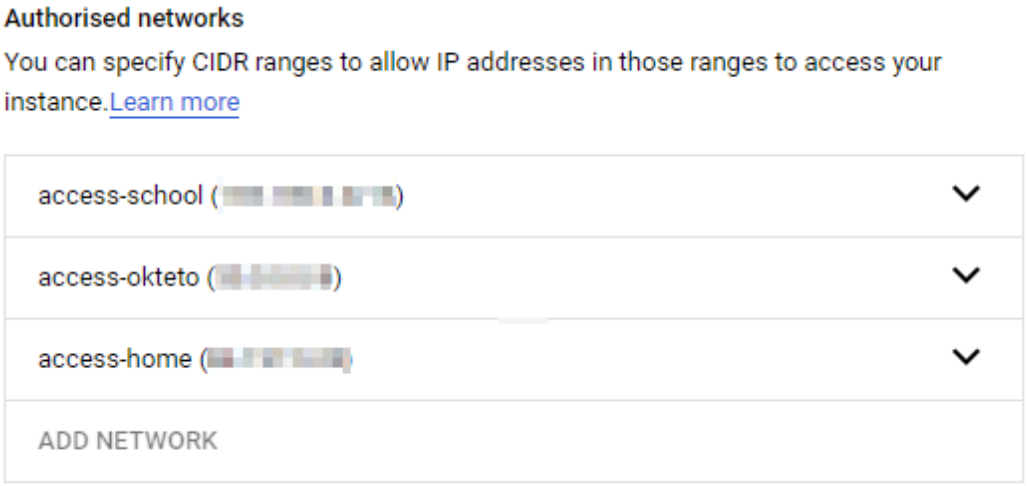


Once this is done simply **connect to the database** via any preferred **database connection tool**. The preferred tool in our case is **MySQL Workbench**.

In order to connect from your **local computer** you also need to **whitelist your ip address**. Simply look for your **public ip address** by googling **what is my ip:**



Now **whitelist this ip address** in your **GCP database**. Click on **Connections** while in your GCP database instance and **adjust the authorized network**:



In order to allow your **deployment** to **communicate with the database** it is important to **whitelist** your **load balancer IP**. Please revert to the **domain** and **load balancer IP** sections for more information about obtaining this IP. Once you have obtained this IP you simply need to **whitelist** this IP address similar to your own IP address.

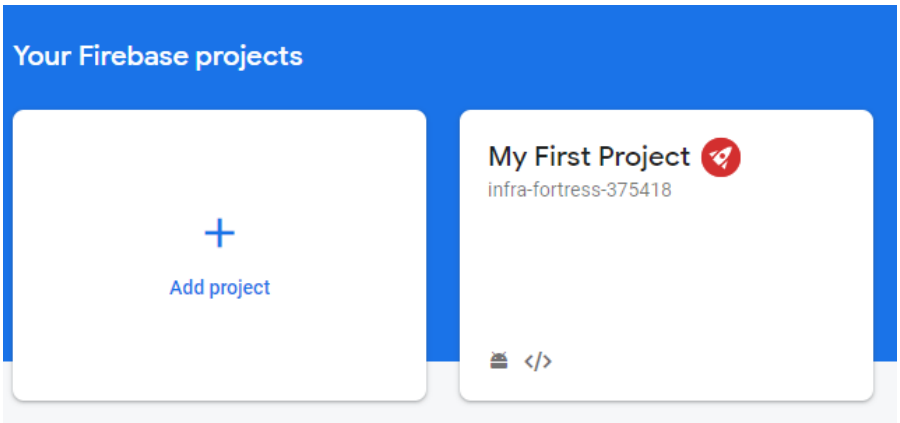
*Use the **add network** button in order to create a new IP whitelisting. Add your public IP address or a custom range in order to be able to connect to your GCP database.*

Once this is provided you can successfully connect to your GCP database via MySQL workbench.

In your **workbench** you only need to run the **automated script** provided within the files (.sql) and **run this SQL file** for your database. The SQL script is **provided within the map structure** and can be recognized by the .sql extension.

3.5.3.3 Firebase

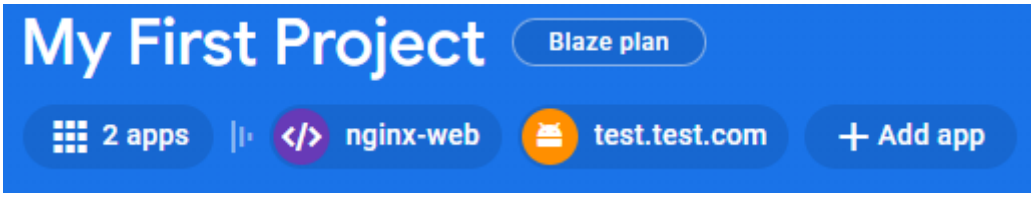
Firebase is our **picture storage solution**. The only requirement you need is to **create a new FireBase project** and create a **storage container**.



It is important to note you best use an **authorized google cloud account** which is linked to your GCP cloud. The main account used in google cloud

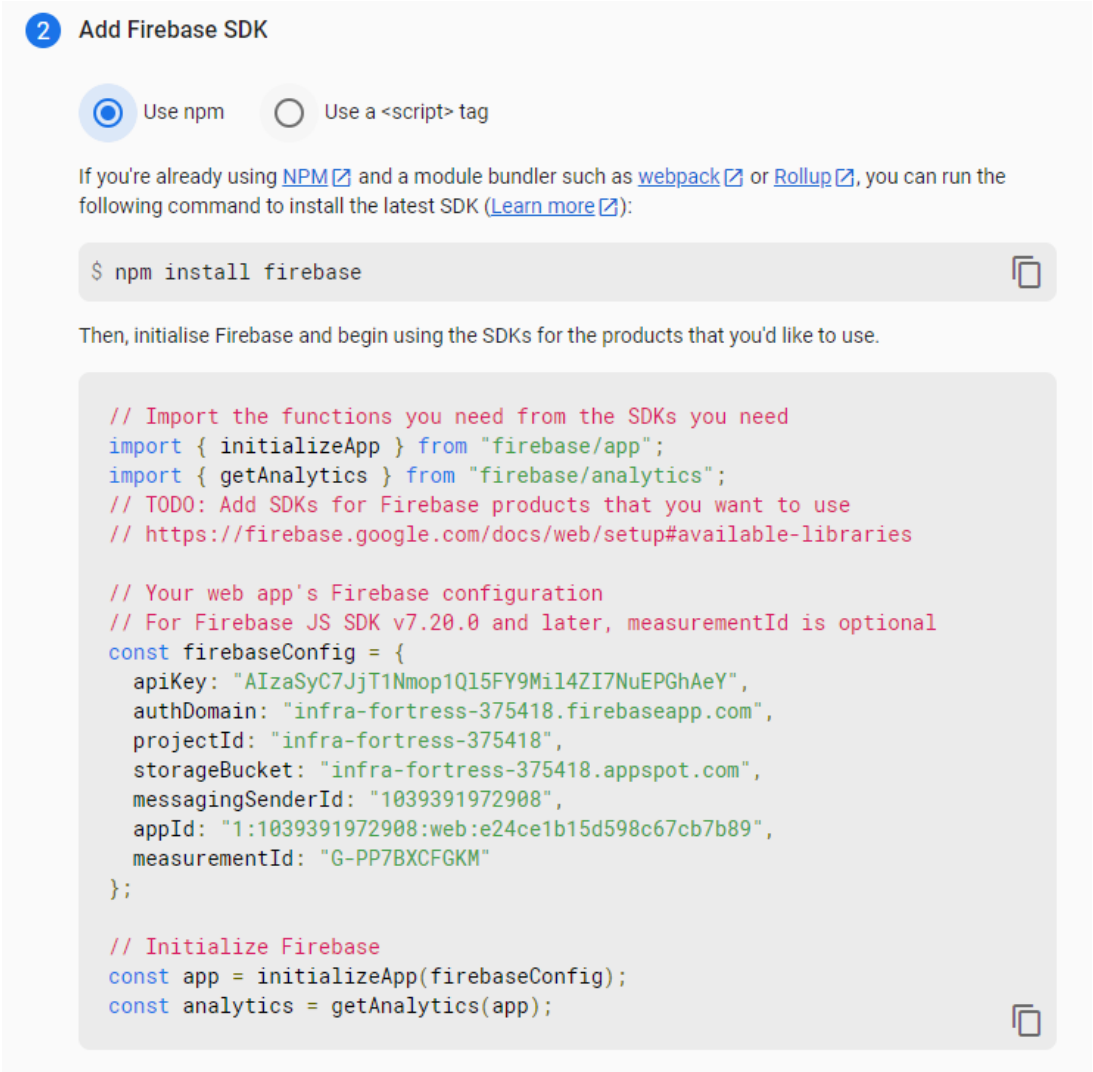
platform would ideally be your FireBase account as well. Since this is a Google product – you can sign in with this account and set up a project accordingly.

Link **FireBase** to your **Google Cloud Platform** by creating a **new app**.



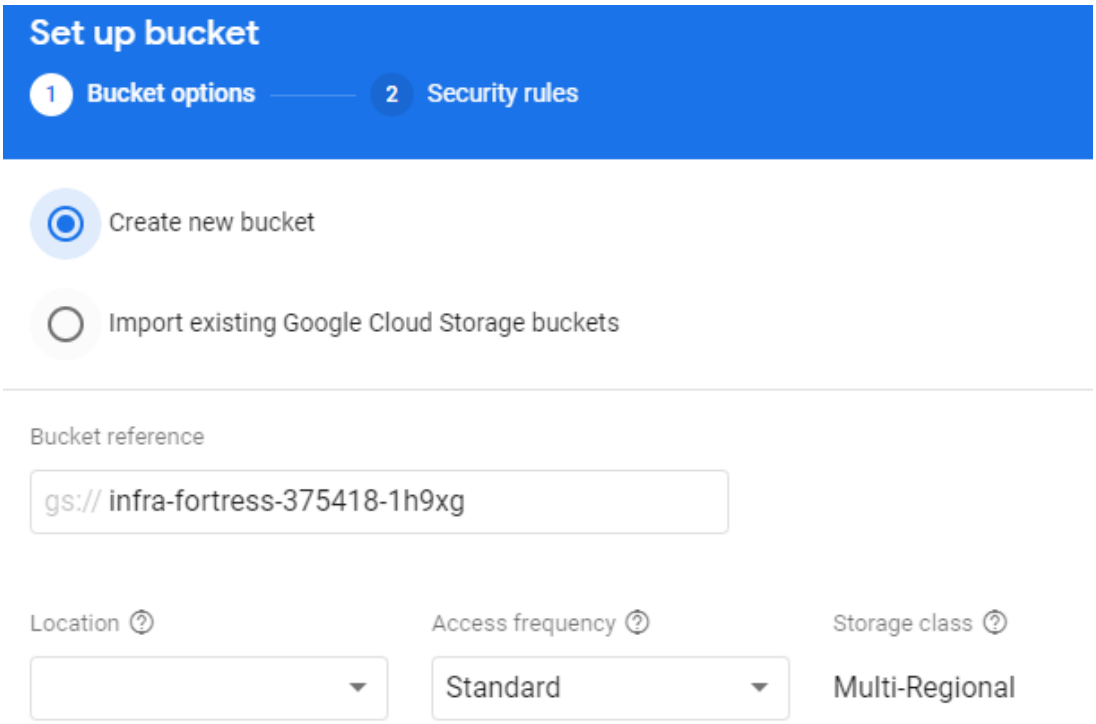
Now click on the **Storage** tab in order to create a storage container.

This will provide you with all the configuration required in order to setup your storage solution in order for applications to use it.



Since these steps already have been taken – you do not need to do this anymore. This is just additional information on how you would link GCP and FireBase.

Once this has been done – you can create a **storage container** as well.



Follow the steps and create a new storage container.

Again – since this **has already been provided** – you do not need to follow this specific step.

If you do create a new custom FireBase account and setup – simply change the GitLab environment variables for the webserver!

The webserver is the only connection towards the FireBase storage solution.

One last aspect you need to consider is adjusting the **rules** of your storage container. A simply set of rules is provided and can be toggled **on or off**:

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if false;
    }
  }
}
```

Simply change the **false** state to **true. CAUTION:** this will allow **EVERYONE** to access your storage container (read: the entire world). It is best to create **custom rules** for **your solution** specified.

3.5.3.4 Service account

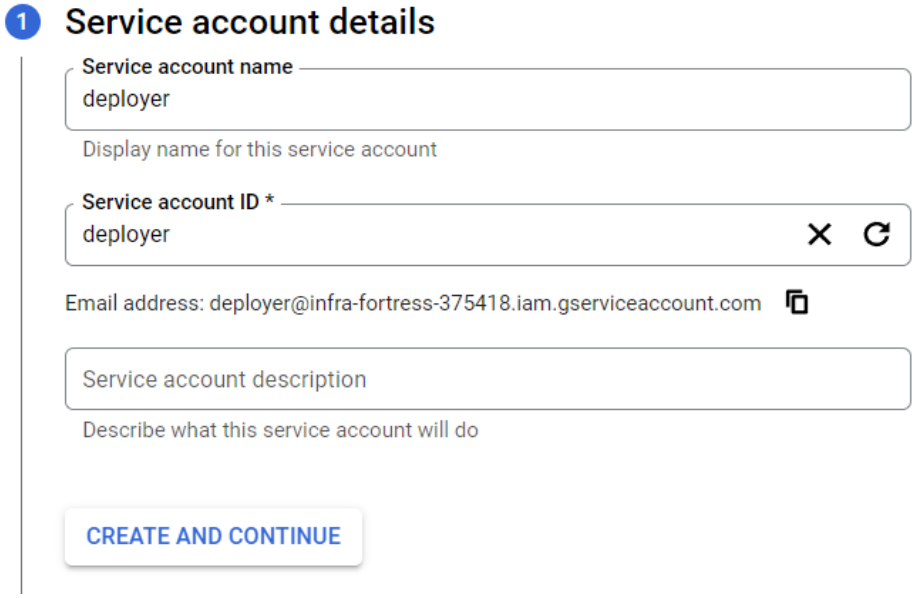
In order to be able to build towards the cloud platform you need to provision **sufficient rights** to the CI/CD pipeline to **access your cloud provider**.

In order to accomplish this we will provide an example of how to do this in your **google cloud platform**.

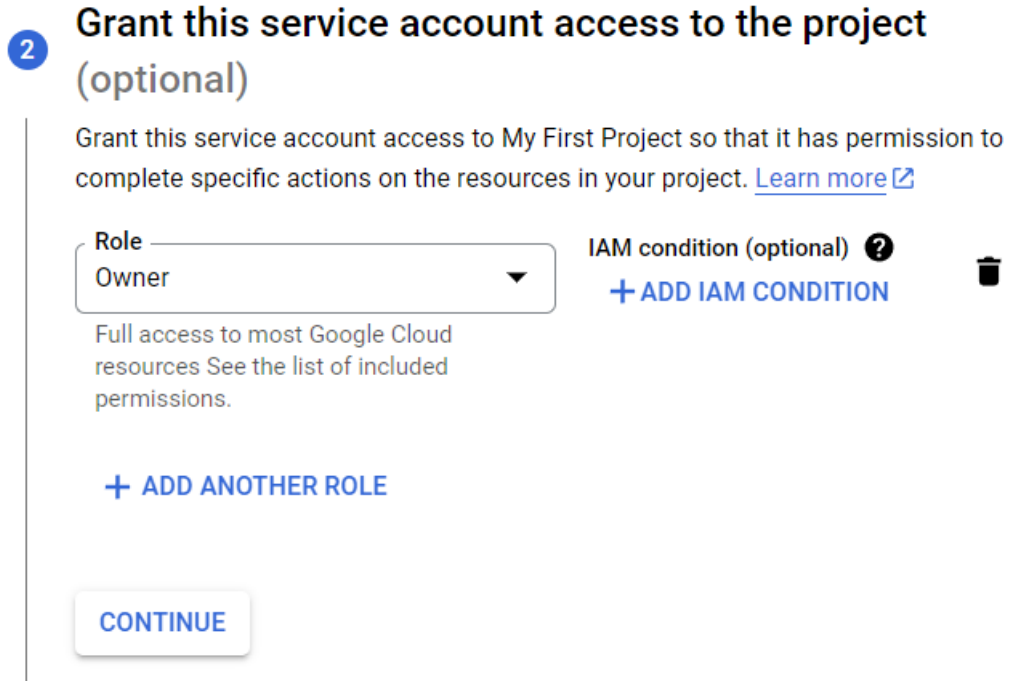
You must create a service account with the **owner** role. Please take in consideration this is possibly not the **most secure** way of using such service account. As explained before – you should create specific service roles for each separate job and apply the **least access privilege**.

Under the tab **IAM and admin** you can find a section **Service Account**.

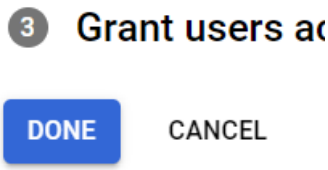
1. **Create service account:** click on the blue button "+ **create service account**". *For example – this SVC will be called "deployer" because it will deploy your GitLab resources to the cloud.*



2. **Grant service roles:** now you need to give your service account **access rights**. In our example we use the **owner** role – which has full admin access.

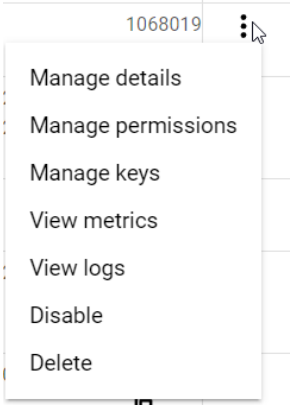


In the third step you don't have to perform anything – thus you can click on **DONE**.

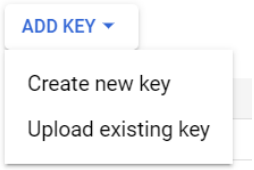


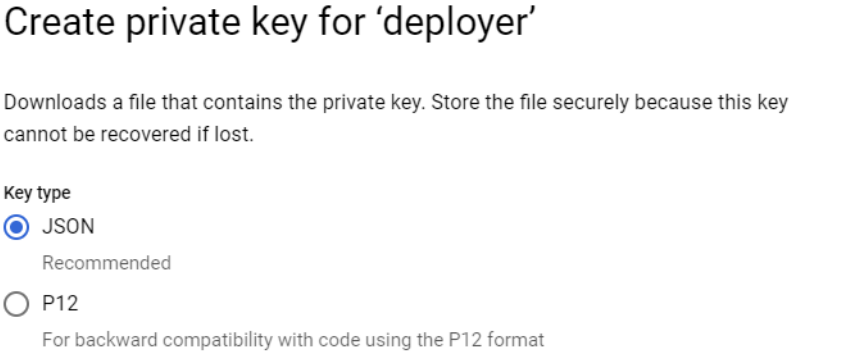
Now your **service account** has been created.

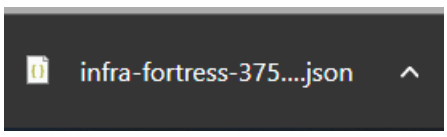
At this stage you still require an **access key**. In order to create this access key you simply click on the **three dots** at the end of the service account and select **Manage Keys**.



Here you will create a **new key**. Click on the blue button and select **create new key**.





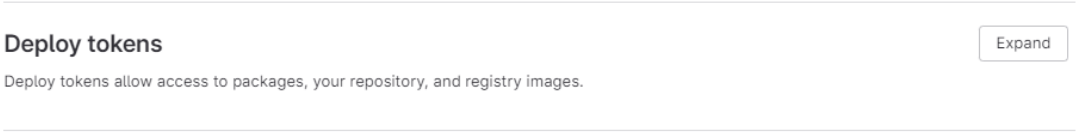


Select **JSON** and click **Create**. Now GCP will create a **JSON download file.** This file you need to **import in the designated variable** in order for your CI/CD to have access to your cloud environment. Insert the entire contents of this JSON file into the variable and you should have access.

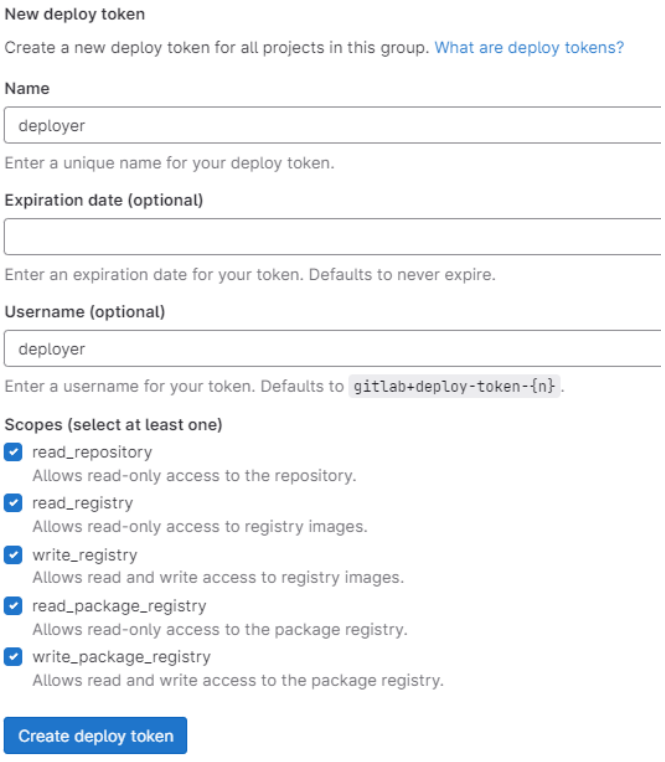
3.5.3.5 Deploy token

A **deploy token** is a specific token used to **deploy**, as the name suggests. This token is required in order to **use the .gitlab-ci.yml** file. This is effectively your deployment file. Either way you can still use **variables** if you choose to.

Click on the **Settings** tab on your left and select **Repository**. Scroll down and **expand Deploy tokens**.



Create a new deploy token and **make sure to capture the deploy token**. The Username is **required** in this case since we are using this value as well.



This deploy token can be used in order to perform the CI/CD pipeline actions. *Currently this is hardcoded in the .gitlab-ci.yml file:*

 # # - kubectl create secret docker-registry regcred --docker-server=${CI_REGISTRY} --docker-username=<Username> --docker-password=<deploy token>

Once you adjust the parameters you are able to deploy via GitLab.

3.5.3.6 Load balancer IP

Last but not least you will have to provide **DNS records** of your **load balancer IP**. Revert to the **domain section** for an overview of **all the A records** required. *An example of A records you require:*

DNS name ↑	Type	TTL (seconds)	Routing policy
ai.vitofruitcounter.com.	A	300	Default
api.vitofruitcounter.com.	A	300	Default
vitofruitcounter.com.	SOA	21600	Default
vitofruitcounter.com.	NS	21600	Default
vitofruitcounter.com.	A	300	Default
www.ai.vitofruitcounter.com.	CNAME	300	Default
www.api.vitofruitcounter.com.	CNAME	300	Default
www.vitofruitcounter.com.	CNAME	300	Default

In the **domain** section this is more thoroughly explained.

3.5.3.7 Kubernetes files

The project itself is built with **Kubernetes files**. These files make sure the infrastructure is automatically generated in a Kubernetes cluster. In order to be able to obtain a **valid domain name**, it is required you change certain files accordingly.

In the **CCS** folder, a variety of files and folders can be found:

- **AI-Server:** this folder holds all the deployment files required for the AI server. *While it doesn't use the Dockerfile – as this is also present within the general AI server found in the map at the root, we still included it here.* The deployment file here is called **yolo.yaml.**
- **API-Server:** this folder holds all the deployment files required for the API server. The deployment file is called **ApiServerDeployment.yaml.** The **project** folder holds the **production API files** required for deployment, as well as the dockerfile.
- **Certificate-mgmt:** Since we are using **Let's Encrypt** for our deployment security and encryption, here you will find **all the required cert-manager** files required by cert-manager. Cert-manager is automatically deployed and provisions certificates with these files.
    - **Production:** these .yaml files hold the **production certificate.** This will provide you with a proper certificate on production assets.
    - **Staging:** these .yaml files hold the **staging certificate.** This can be used if you wish to deploy a staging environment in a GCP cluster. In our current setup we have provisioned a staging environment within Okteto Cloud. At this stage of the project you can easily deploy a **second cluster**, aligning with the staging certificate, to provide a proper staging setup within GCP.
- **Staging:** this folder holds all the **staging assets** required for deployment. Mostly revolving around the database and a test API setup. For development. Since the API will use a different database – the

staging database – it required a different **database.py** setup (and especially different .env variables).

- **Terraform:** this folder holds all the required files for our **terraform automation.** All the terraform files can be recognized by their .tf extension.
- **Webserver:** this folder holds all the required files for deploying our **webserver**, including our **ingress rules.** The files actually used here are:
    - **Ingress.yaml:** this file holds the **ingress ruling** for **all the applications.** This will be the only file that needs changing for your deployment.
    - **Deployment.yaml:** this file holds the **deployment** of the **webserver** in our Kubernetes cluster.
    - **Service.yaml:** this file holds the **service** of the **webserver** in our Kubernetes cluster. *A service is required to expose the application. In the other application folders this part is included within the single .yaml file.*

The **most important** file you need to **change** is the **ingress.yaml** file, found in the Webserver folder.

```
spec:
  tls:
    - hosts:
      - vitofruitcounter.org
      - api.vitofruitcounter.org
      - ai.vitofruitcounter.org
      secretName: ssl-cert-production
  rules:
  - host: vitofruitcounter.org
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: hello-nginx-https
            port:
              name: http

  - host: api.vitofruitcounter.org
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: hello-api-server
            port:
              name: api

  - host: ai.vitofruitcounter.org
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: yolo-service
            port:
              name: ai

```

A domain is generated in a previous chapter. Once this is done – you simply change the **host parameters** within this file. Every single **URL** should be **changed** to **your domain** accordingly before deployment.

Once this has been achieved – ingress will automatically create new rules linking your domain names to your load balancer. If you have provided the correct DNS records in your Cloud DNS (A records), the outside world should now be able to open these host URLs correctly.

Do take in consideration the Let's Encrypt can take a little while to apply it's production certificate.

This file could also be used to create another **staging environment**, for example. Simply change the **secretName** and certmanager.k8s.io/cluster-issuer: "letsencrypt-production" to **staging** where **production** is called.

Now two clusters can be spawned: one production cluster and one staging cluster.

The other files in the folder were used for test deployments and are here for documentation purposes, as well as self-signed certificates.

3.5.3.8 Docker compose

Last but not least you can find a **docker compose** file in the **root** of the project. This file can effectively be used in the exact same matter as the Kubernetes cluster – making testing in Okteto Cloud a tad easier.

```
services:

  db:
    image: freds00n/database-codefarm:latest
    container_name: mysql-codefarm
    environment:
      - MYSQL_DATABASE=codefarm
      - MYSQL_ROOT_PASSWORD=abc123
    ports:
      - "3306:3306"
    restart: always
    expose:
      - "3306"
#    env_file:
#        - .env

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    depends_on:
      - db
    container_name: my-php-myadmin-codefarm
    ports:
      - "8082:80"
    restart: always

  api-service:
    depends_on:
      - db
    links:
      - db
    image: freds00n/apiserver-codefarm:latest
    container_name: apiserver
    ports:
      - "8000:8000"
    restart: always
#    env_file:
#        - .env
```

If you wish to perform a Docker deployment – or a deployment in Okteto Cloud (which is free – but has limited resources!), you simply adjust this file accordingly.

Change the **image** variables to contain **your own docker hub images**. The current mentioned **images** are provisioned by **one of the CCS students** and should be the **latest up-to-date** files according to the project deadline.

Once new development has started on this project – new images need to be created, and the **image** names should be changed. The only image that should not be changed is your **phpMyAdmin** image – since this is just a general deployment used to **inspect your database table(s).**

Other important files, such as the database files, can be found within the **staging** folder in the **CCS** folder.

At this stage you simply need to create your **own docker files** which can be used to deploy a second variant of our project. Simply use the below commands in order to generate new docker images for the deployments:

Always login first: docker login -u $DOCKER_USER -p $DOCKER_PASSWORD

Webserver examples (use the root location):

- docker build --no-cache -t freds00n/codefarm-webserver:latest -f CCS/Webserver/Dockerfile .
- docker push freds00n/codefarm-webserver

AI server examples (use the root location):

- docker build -t freds00n/ai-codefarm:latest -f AI/Dockerfile ./AI
- docker push freds00n/ai-codefarm:latest

API server examples (use the root location – staging setup will be used here):

- docker build --no-cache -t freds00n/apiserver-codefarm:latest ./CCS/Staging/api-server
- docker push freds00n/apiserver-codefarm:latest

Database examples (which includes an **automated .sql script** – which is also used in our **production database**) use the **database staging folder**:

- docker build --no-cache -t freds00n/database-codefarm:latest .
- docker push freds00n/database-codefarm:latest

Apply the above commands with your **own username** (not including <freds00n>) to push images to your **docker hub**.

3.5.3.9 Budget

In order to have a realistic view on the possible **costs** of **your deployment** – a overview will be given here.

A **basic deployment** as described above with a **minimal set of capabilities** (minimum RAM and CPUs) for your **Kubernetes cluster** will result in approximately **20 €** consumption **each week**:

Service	Cost	Discounts	Promotions and others	↓ Subtotal
Cloud Domains	€11.26	€0.00	-€11.26	€0.00
App Engine	€0.00	€0.00	€0.00	€0.00
Kubernetes Engine	€25.55	-€25.55	€0.00	€0.00
Cloud Storage	€0.01	€0.00	-€0.01	€0.00
Cloud Logging	€0.00	€0.00	€0.00	€0.00
Cloud DNS	€0.22	€0.00	-€0.22	€0.00
Networking	€3.82	-€3.74	-€0.09	€0.00
Cloud SQL	€18.07	€0.00	-€18.07	€0.00
Compute Engine	€20.64	€0.00	-€20.64	€0.00

This includes **two stacks** with **two continuously running clusters**. SQL databases are **manually** started each time you require them. All other aspects will **always be running** (DNS, for example).

Project	Project ID	Project number	Cost	Discounts	Promotions and others
[Charges not specific to a project]	–		€11.04	€0.00	-€11.04
terraform-testproject	terraform-testproject-375708	796993379129	€16.21	-€2.39	-€13.82
My First Project	infra-fortress-375418	1039391972908	€81.78	-€41.70	-€40.08

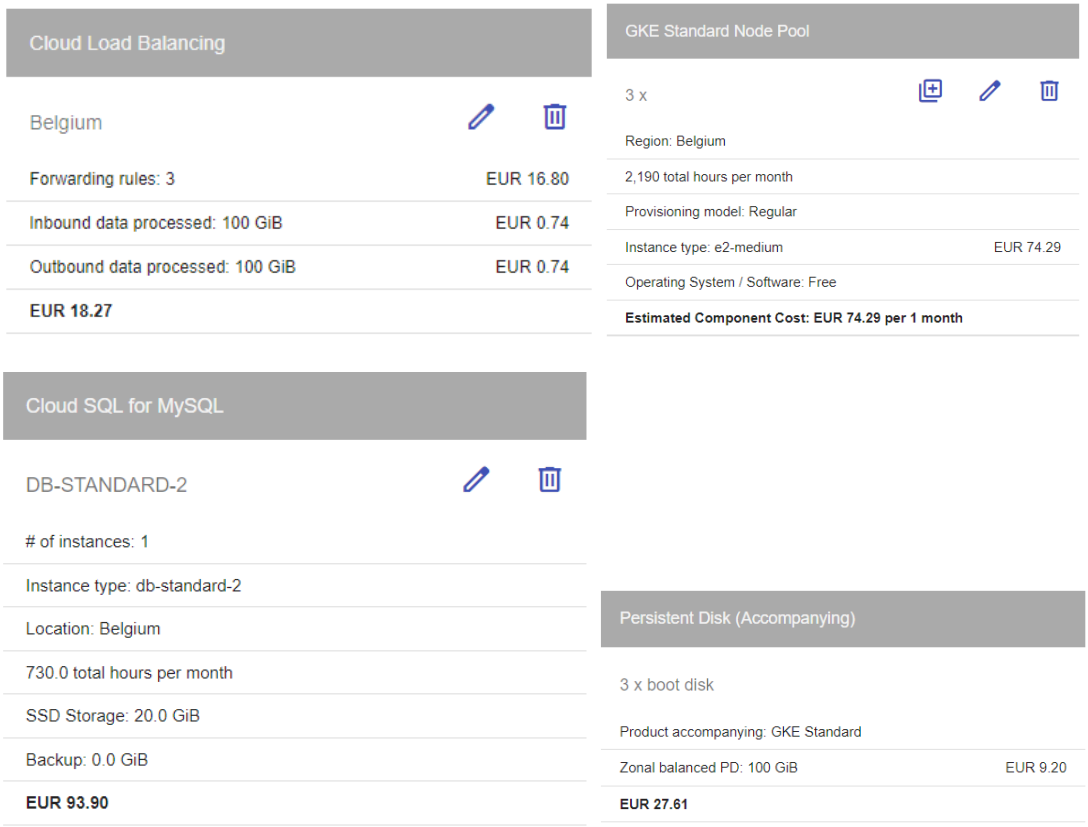
Thus far – running from **1 February** until **18 February** you will have a total bill of **64,94 €** approximately. By combining both stacks you can obtain a realistic view on how much the current stack would cost. **Discounts** are automatically applied by GCP due to selecting certain settings in your deployment. **Promotions and others** is the reduction obtained by **Google Cloud free budget** provided at the start of this account. Thus this is the normal operation cost in our case if we did not have free budget provided.

Approximately this would result in +- **100 € running costs** each month. Do take in consideration: SQL database is mostly **off** while **all other services** are **continuously running**. This would not give a realistic image of the true costs – thus we have created a simulation.

Four aspects need to be taken into consideration:

- **Cloud load balancing:** load balancing can also add costs due to traffic coming into your network and going out your network.
- **GKE Standard Node pool:** this is your Kubernetes cluster – essentially all your services and machines running.
- **Cloud storage:** this is your storage solution. While this is a very simple calculation – do keep in mind **FireBase** can add additional costs.
- **Cloud SQL for MySQL:** this is your database solution.

For all these aspects we have generated the following approximate costs:



Cloud Load Balancing	
Belgium	
Forwarding rules: 3	EUR 16.80
Inbound data processed: 100 GiB	EUR 0.74
Outbound data processed: 100 GiB	EUR 0.74
EUR 18.27	

GKE Standard Node Pool	
3 x	
Region: Belgium	
2,190 total hours per month	
Provisioning model: Regular	
Instance type: e2-medium	EUR 74.29
Operating System / Software: Free	
Estimated Component Cost: EUR 74.29 per 1 month	

Cloud SQL for MySQL	
DB-STANDARD-2	
# of instances: 1	
Instance type: db-standard-2	
Location: Belgium	
730.0 total hours per month	
SSD Storage: 20.0 GiB	
Backup: 0.0 GiB	
EUR 93.90	

Persistent Disk (Accompanying)	
3 x boot disk	
Product accompanying: GKE Standard	
Zonal balanced PD: 100 GiB	EUR 9.20
EUR 27.61	

Total Estimated Cost: EUR 214.07 per 1 month

The above calculation takes into consideration **all services are running continuously (24/7).** Which brings a total of **214,07 €** expenditure each month.

- **GKE Standard Node Pool:** by far the #1 cost. This calculation essentially provides you with **two pools** that can be used by the applications. If, for example, the load in one pool is becoming too much – the other pool can be used and essentially gives you **double the resources**. This can always be **optimized** – is your load lower? Change the node pool settings and reduce costs if required.
- **Cloud SQL for MySQL:** a cloud instance does cost a lot of money – but is essentially 24/7 available. You have at least 100 GB of storage and sufficient resources to process incoming traffic. Do take into consideration you do **not have to manage this database at all**. This is the reason why it is also **HA: High Available**. The database requires **updates** and can cause your DB to be unavailable. Since it is setup as HA – you will **never** experience downtime and the DB is essentially updated and maintained for you. You only need to control the data!

Persistent storage relies on the fact your GKE standard node pool needs to have storage available. Since each pool can run on 50 GB of storage – this should be sufficient for each node.

3.5.4 AI Models

The Smartphone model

3.5.4.1 Dataset preparation process

We collected 340 high-resolution images of strawberry flowers from the greenhouse located on Thomas More campus in Geel. The images were captured using a smartphone camera with a resolution of 12 megapixels. Before labelling, we performed quality checks to ensure that the images were suitable for training the YOLOv8 model. As a result, we removed a number of images that were out of focus, blurred, or had poor lighting conditions. After the quality checks, we were left with a total of 315 high-quality images for labelling.

To label the images, we used the open-source annotation tool, MakeSense.ai. Two annotators manually labelled the images, drawing bounding boxes around the strawberry flowers and assigning them to the "strawberry" class. We also applied random image augmentations, such as rotation, flipping, and resizing, to increase the diversity of the dataset and improve the model's performance. The final dataset consisted of 315 labelled images and was split into training (70%), validation (15%), and testing (15%) sets.

3.5.4.2 The model architecture

We made several modifications and customizations to the YOLOv8 model architecture and parameters to improve its performance on our strawberry flower detection task. Specifically, we used the following modifications:

We increased the input image size to 608x608 to capture more details and improve the detection accuracy.

We changed the anchor boxes to better match the size and aspect ratio of strawberry flowers.

We adjusted the number of filters and layers in the backbone network to improve the feature representation and reduce computation time.

We trained the model using the Adam optimizer with a learning rate of 0.0001 and a batch size of 64, for a total of 100 epochs.

Overall, these modifications and customizations helped to improve the performance of the YOLOv8 model on our strawberry flower detection task, achieving an average precision (AP) of 0.93 on the validation set. Keep a record of the validation process, including any metrics used to evaluate the performance of the model, such as mAP (mean average precision).

3.5.4.3 Challenges encountered

During the training process of the YOLOv8 model, we encountered the following issues and challenges:

Limited amount of labelled data: We only had 315 labelled images for training the model, which could lead to overfitting and poor generalization to new images. To address this issue, we applied data augmentation techniques to increase the diversity of the dataset and improve the model's ability to generalize to new images.

GPU memory limitations: The YOLOv8 model is a large and complex model, and training it on a single GPU with limited memory can be challenging. We

addressed this issue by adjusting the batch size and reducing the image resolution to fit the available GPU memory.

Overall, by applying these techniques and addressing the issues we encountered, we were able to train a high-performing YOLOv8 model for our strawberry flower detection task. The model achieved an average precision (AP) of 0.93 on the validation set and performed well on new, unseen images.

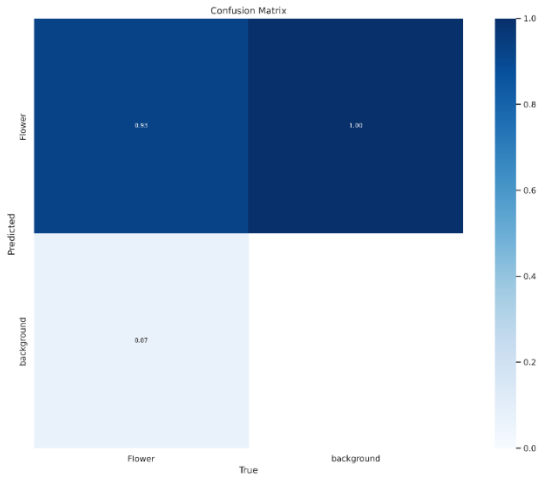
3.5.4.4 Code snippets

For this training, we used the YOLOv8l object detection model and a large pre-trained label to achieve high accuracy in detecting objects. However, it is important to note that there is a trade-off between speed and accuracy when using larger models and labels. The training was run for 100 epochs on the specified dataset.

```
!pip install ultralytics

%cd ultralytics
%pip install -qr requirements.txt

drive.mount('/content/gdrive/', force_remount=True)

from ultralytics import YOLO
model = YOLO("yolov8m.pt")  # load a pretrained YOLOv8n model
model.train(data="custom_data.yaml", epochs=100)  # train the model
```

We used the Ultralytics YOLO library to train a custom object detection model using the YOLOv8m architecture. The model was trained on a custom dataset specified in the custom_data.yaml file for 100 epochs. The drive.mount() command was used to mount Google Drive to the Colab environment to use the custom dataset. The model was loaded with the YOLO() function and then trained with the train() function.

3.5.4.5 Performance evaluation

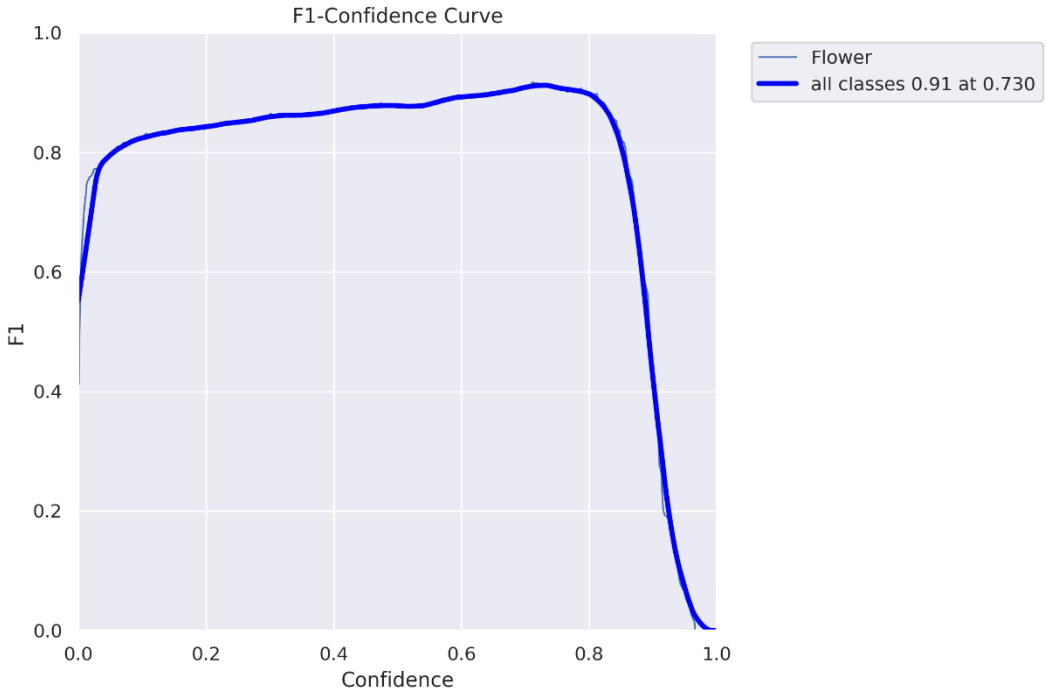


A confusion matrix is often used to describe the performance of a model on a set of data for which the true values are known.

In our case, the model predicted that there is a flower with a probability of 0.93, and background with a probability of 0.07, while the actual object in the image is also a flower. This is referred to as a true positive (TP) in the confusion matrix.

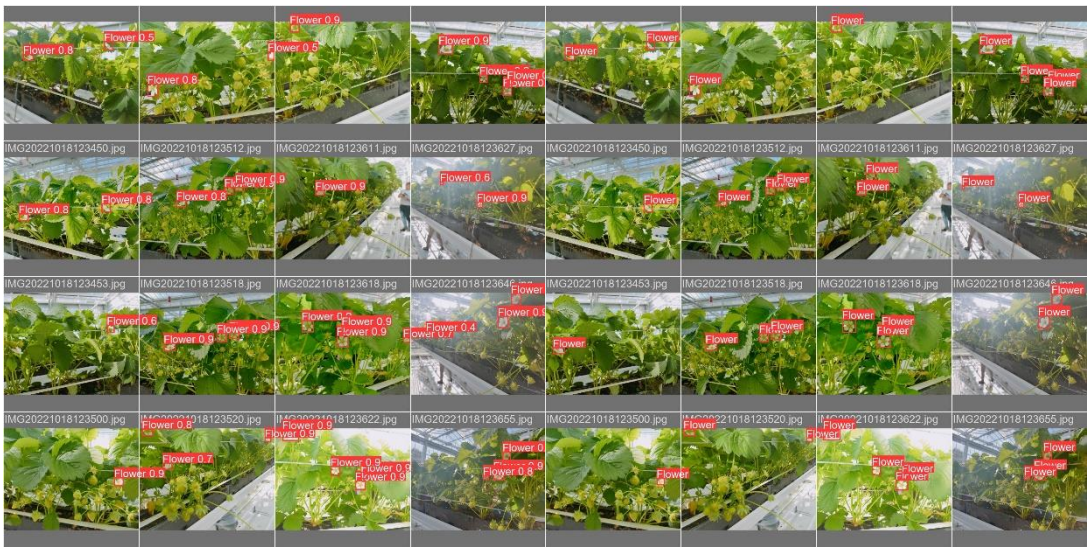
The second scenario from the graph above is when the model predicted that there is a flower with a probability of 1.0, but the actual object in the image is the background. This is referred to as a false positive (FP) in the confusion matrix.

The confusion matrix allows us to evaluate the performance of a model by comparing its predictions to the actual true values. In this case, the model appears to perform well in correctly identifying the presence of flowers, but it also has a high rate of falsely identifying backgrounds as flowers. By analyzing the confusion matrix, we can identify areas for improvement in the model's performance and adjust the model accordingly.



The F1 confidence curve is a graphical representation of the relationship between the model's F1 score and the confidence threshold used to make predictions. The F1 score is a measure of the model's accuracy that takes both precision and recall into account. By varying the confidence threshold, you can adjust the trade-off between false positives and false negatives, and the F1 confidence curve allows you to visualize the impact of this trade-off on the overall accuracy of the model.

Our smartphone model had 0.91 accuracy at a confidence threshold of 0.730. This means that at a confidence threshold of 0.730, the model has a good balance of precision and recall for all classes, indicating a high level of overall accuracy.



The Satellite / Drone model

3.5.4.6 Dataset preparation process

We labelled 34 images using makesense.ai and label studio. Due to the tiny size of the flowers that need to be detected, the images were sliced into smaller 15 x 15 images. This allowed us to generate a larger dataset of 6100 images for training, 1400 images for validation, and 225 images for testing.

3.5.4.7 The model architecture

We made several modifications and customizations to the YOLOv8 model architecture and parameters to improve its performance on our strawberry flower detection task for the drone images. Specifically, we used the following modifications:

We increased the input image size to 1024x1024 to capture more details and improve the detection accuracy.

We changed the anchor boxes to better match the size and aspect ratio of strawberry flowers in drone images.

We trained the model using the Adam optimizer with a learning rate of 0.0001 and a batch size of 32, for a total of 100 epochs.

Overall, these modifications and customizations helped to improve the performance of the YOLOv8 model on our strawberry flower detection task for the drone images, achieving an average precision (AP) of 0.94 on the validation set. We kept a record of the validation process, including the metrics used to evaluate the performance of the model, such as mAP (mean average precision).

3.5.4.8 Challenges encountered

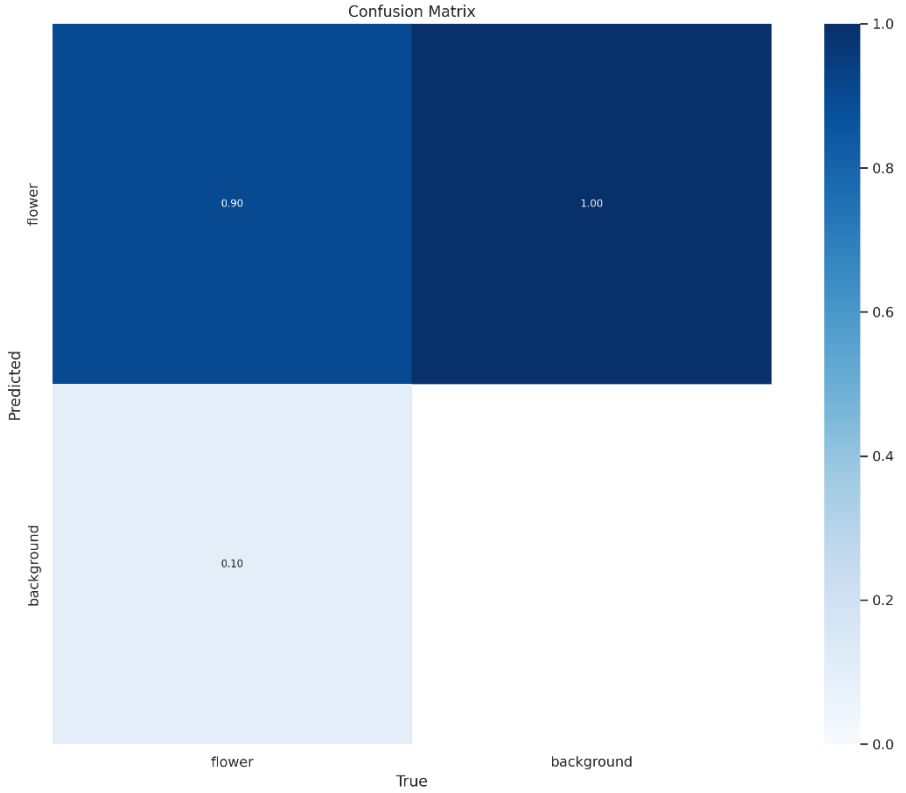
Training the YOLOv8 model on drone images posed some challenges. First, the small size of the objects of interest made labeling a difficult and time-consuming task. Additionally, we faced GPU memory limitations, which required us to carefully manage the batch size and the size of the input images. Despite these challenges, we were able to successfully train a model that achieved high accuracy on the detection task

3.5.4.9 Code snippets

```
!yolo task=detect mode=train model=yolov8l.pt data={dataset.location}/data.yaml epochs=100
```

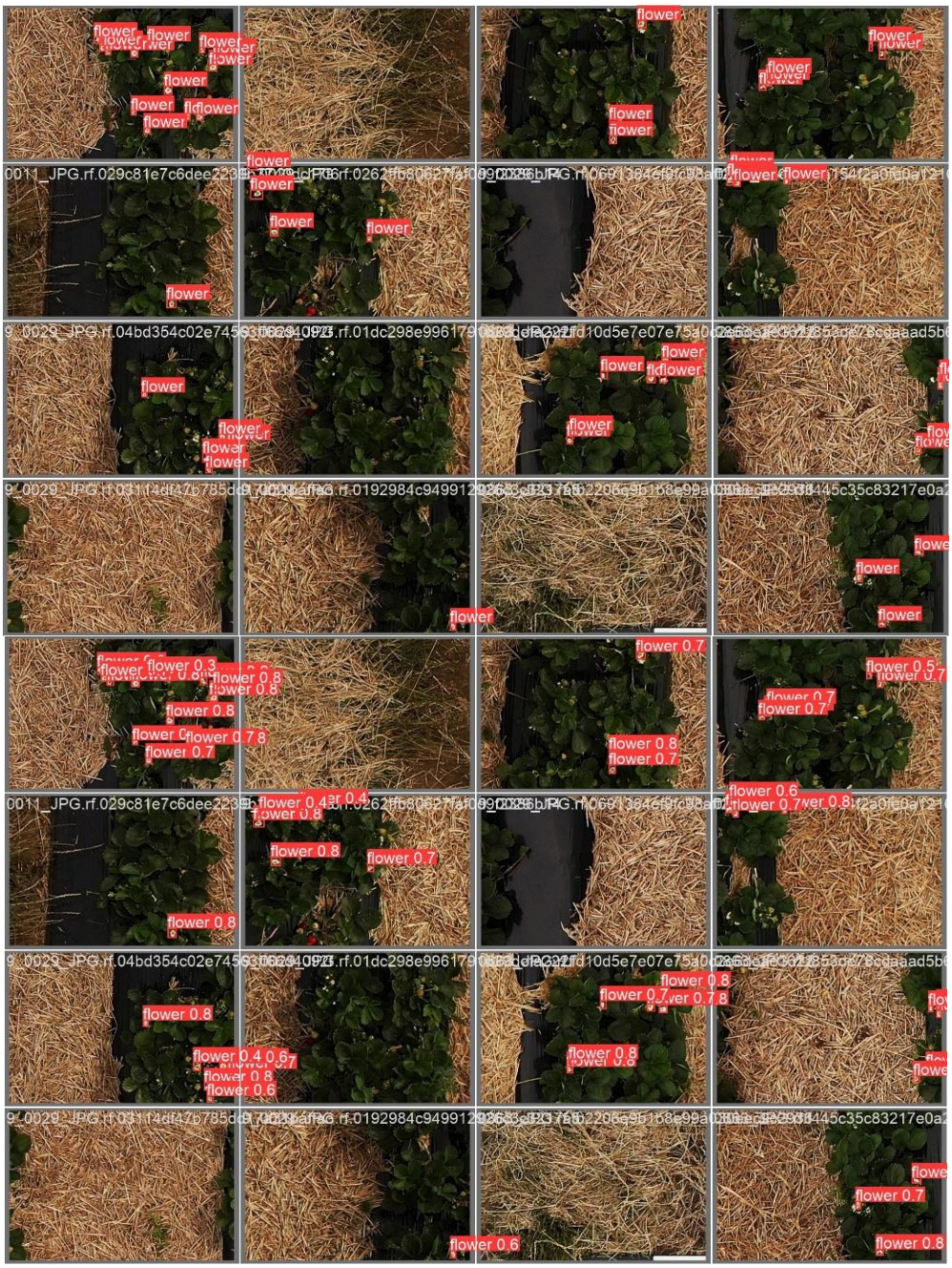
For this training, we used the YOLOv8l object detection model and a large pre-trained label to achieve high accuracy in detecting objects. However, it is important to note that there is a trade-off between speed and accuracy when using larger models and labels. The training was run for 100 epochs on the specified dataset.

3.5.4.10 Performance evaluation



A confusion matrix for a YOLO model shows how well the model is able to correctly identify and classify objects in an image. In this scenario, the confusion matrix indicates that the model was able to accurately identify a true flower 90% of the time (true positive) while incorrectly predicting a flower 10% of the time when there was none (false positive). Additionally, the model was able to accurately identify a true background 100% of the time (true negative) and did not predict any flowers when there were none (true negative).

Overall, the confusion matrix suggests that the model is performing well in identifying true flowers, but may need to be fine-tuned to reduce the false positive rate in order to improve its overall accuracy. We performed a number of actions like including more background images which ultimately increased the accuracy of the model.



3.5.4.11 Conclusion

The training of the YOLOv8 model for strawberry flower detection was successful, and we achieved high accuracy metrics on the validation set. The final average precision (AP) of the model on the validation set was 0.93 for the smartphone mode and 0.922 for the satellite / drone model, indicating that the model was able to accurately detect strawberry flowers in images.

We also observed that the model's performance improved with additional training epochs and a larger batch size, as well as the use of mixed precision training. The use of data augmentation and focal loss also helped to improve the model's ability to generalize to new, unseen images.

Based on these results, we can conclude that using appropriate training techniques and hyper parameters is important to achieve high accuracy metrics in object detection tasks.

3.5.5 Application

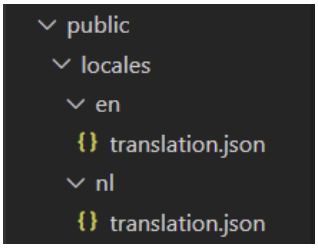
3.5.5.1 Codebase

For this project we created a mobile application and a web application. We chose React as our code base because that way we only had to make one project for both applications. By making this decision we saved a lot of time because we only had to write one piece of code to achieve the same goal. This work method is called a Progressive Web App(PWA).

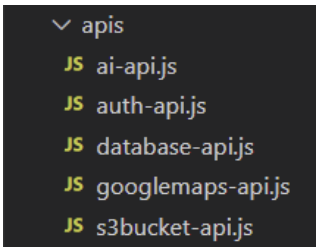
3.5.5.2 Challenges encountered

For the possibility of taking pictures on the mobile application we had to implement a npm library. Not all libraries were compatible with some other libraries we used, so we had to do a lot of trial and error with a couple of camera libraries. Another challenge with using the camera was that all the libraries automatically convert the taken images to a base64 string. Due to this we had to make some workarounds to send a proper jpg image to the AI server.

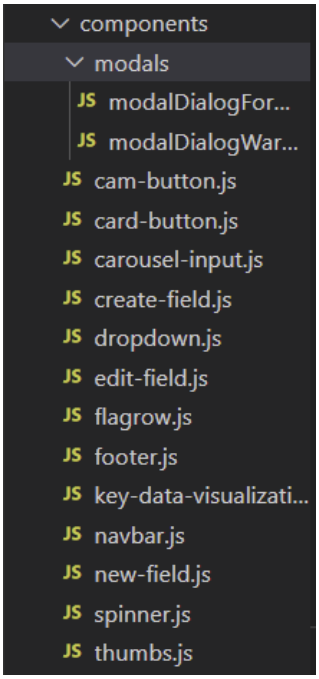
3.5.5.3 File structure



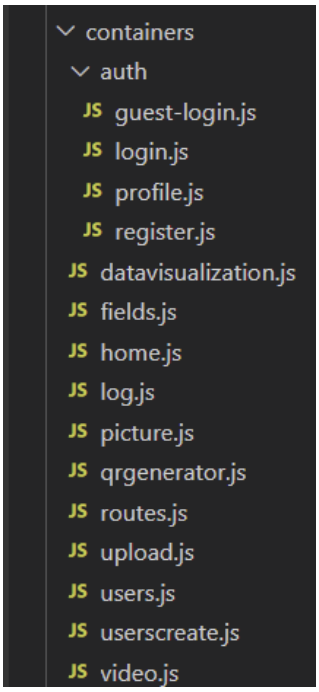
In these files you can find the text for the application in Dutch and English, if you want to add an extra language or text you can add that here.



In this folder you can find the files which connects to the different API's we've used. We chose to make a separate file for each API to have a clear overview of the different API calls.



In the components folder we stored all smaller parts of a page. This is easier because that way we can re-use a part if you need it on different pages. This way you don't need to configure every part over and over again.



In this container folder we programmed the files for the main pages of the application. On this pages we implemented the components so the user can access them.

3.5.5.4 Use and adapt the project

To use and make changes locally in the project you have to follow the next steps.

1. Download the project folder onto your device
2. Open a terminal in this folder
3. Run the command 'npm install' to install all the used libraries in the project on your device
4. Run the command 'npm start' to run the application so you can review and test the changes you've made

4 CONCLUSION

Building multiple services, in a multi-disciplinary team, posed a challenging Project 4.0 deliverable. The deliverable contains a multitude of servers consistent of artificial intelligence, progressive web application, APIs and a Kubernetes infrastructure. All of these services are eventually combined into one overall solution: the ability to be able to recognize (strawberry) flowers from a picture. Even at this stage the assignment was expanded with drone / satellite pictures.

Once separate services were built and ready to be integrated, a CI/CD pipeline creates a fully automated way for integrations. Furthermore the automation was expanded with Terraform, providing a solution to automatically generate most aspects of the base layer of the final project. Combining both a CI/CD pipeline and Terraform creates a Kubernetes cluster that is nearly fully automated and scalable.

Once the above functions have been provided – the addition of expanding APIs and the webserver were our next achievements. The project effectively holds a user management system and a field creation option. As there are different types of users – only a farmer and admin are allowed to view all building blocks of the project. Other users, such as the worker, can only upload pictures, enhancing security further. Each user is provided with a login and the ability to create a new account. Data visualization creates a thorough overview of each farm's data regarding counted flowers. The addition of including a heatmap in the field overview creates a possibility for a farmer to recognize where most of the flowers on their field(s) are. The ability to create and manage multiple fields is also possible.

All of the above features create a fully functional application which is our final Project 4.0 achievement. A lot of the components mentioned required everyone to work on different fields, in their field of expertise and beyond.

In the end we learned a lot from our own part, and especially from each other. Working on features outside of our comfort zone made us resilient to a rapid expanding IT industry. An essential skill for the future. Alongside learning we got to know each other and bonded in order to achieve the desired end result.

This report is the final representation of our Project 4.0 achievement and we hope it brings more ideas and inspiration to the new owners of the project files and instructions listed in this folder.

5 BIBLIOGRAPHY

Interested in looking into our sources? Here you will be able to find even more information about the sources used during our investigative phase.

5 Image Annotation Tools to Get Your Labeling Project Started. (2022, October 25). Datagen. https://datagen.tech/guides/image-annotation/image-annotation-tool/

500: We've Run Into An Issue | Mailchimp. (n.d.). Copyright (C) Mailchimp. All Rights Reserved. https://mailchi.mp/0a6a8b0603aa/hlcog8m7ub

Ali, A. (2022, November 4). *14 Container Orchestration Tools for DevOps*. Geekflare. https://geekflare.com/container-orchestration-software/

Amazon CloudWatch vs Splunk Enterprise. (n.d.). TrustRadius. https://www.trustradius.com/compare-products/amazon-cloudwatch-vs-splunk-enterprise

Application and Infrastructure Monitoring – Amazon CloudWatch – Amazon Web Services. (n.d.). Amazon Web Services, Inc. https://aws.amazon.com/cloudwatch/

Author, B. (n.d.). *6 Reasons Why You Should Choose Cloud Hosting*. https://www.baass.com/blog/6-reasons-why-you-should-choose-cloud-hosting

Badkar, G. H. A. A. (2022, July 1). *Popular Test Automation Frameworks: How to Decide*. BrowserStack. https://www.browserstack.com/guide/best-test-automation-frameworks

Bandyopadhyay, H. (2022, October 6). *YOLO: Real-Time Object Detection Explained*. V7. https://www.v7labs.com/blog/yolo-object-detection

Brice, S. (2021, December 16). *Choosing the Right Model for Object Detection - Samuel Brice*. Medium. https://samdbrice.medium.com/dcda0c8f6f70

Carmichael, C. (2022, December 5). *The 9 Best Cloud Hosting Providers 2022 | Ranked and Reviewed*. Website Builder Expert. https://www.websitebuilderexpert.com/web-hosting/cloud-hosts/

Carrero, L. (2022, September 16). *Which are the most used web servers?* Stackscale. https://www.stackscale.com/blog/top-web-servers/

Chowdhury, A. R. (2022, April 23). *Best Python Testing Frameworks - arnab roy chowdhury*. Medium. https://medium.com/@arnabroyy/best-python-testing-frameworks-bb7ab1b3d366

DhiWise. (n.d.). *Explore the top testing libraries and tools for React.js app.* https://www.dhiwise.com/post/react-testing-libraries-and-tools-of-2022

Etukudo, E. (2022, February 25). *Best options for self-hosting Create React App*. LogRocket Blog. https://blog.logrocket.com/best-options-self-hosting-create-react-app/

Keylabs. (2022, September 8). *Compare The Nine Best Image Annotation Tools in 2022*. Keylabs Blog Features the Latest News and Updates. https://keylabs.ai/blog/reviewing-the-top-9-image-annotation-tools-in-2022/

Kubernetes Documentation. (n.d.). Kubernetes. https://kubernetes.io/docs/home/

Maayan, G. D. (2022, June 20). *AWS vs. Azure Cost Comparison [2022]*. Codemotion Magazine. https://www.codemotion.com/magazine/devops/cloud/aws-vs-azure-cost-comparison/

Mesquita, D. (2022, March 11). *Introduction to Object Detection Model Evaluation - Towards Data Science*. Medium. https://towardsdatascience.com/introduction-to-object-detection-model-evaluation-3a789220a9bf

Michael Crilly. (2021, November 17). *Is HashiCorp Nomad worth learning?* YouTube. https://www.youtube.com/watch?v=j6CZ5nJ1O8Q

Nerd, T. (2022, August 18). *The Top Ten Web Frameworks for creating REST APIs -Backend Development*. PyCodeMates. https://www.pycodemates.com/2022/08/top-ten-web-frameworks-for-creating-rest-apis.html

Pratama, A. S. (2021, December 27). *5 Best Free Image Annotation Tools - Data Folks Indonesia*. Medium. https://medium.com/data-folks-indonesia/5-best-free-image-annotation-tools-80919a4e49a8

ProjectPro. (2022, June 6). *AWS vs Azure-Who is the big winner in the cloud war?* https://www.projectpro.io/article/aws-vs-azure-who-is-the-big-winner-in-the-cloud-war/401

What are the best web frameworks to create a web REST API? (n.d.). slant.co. https://www.slant.co/topics/1397/%7Ebest-web-frameworks-to-create-a-web-rest-api

What are the best web servers? (n.d.). slant.co. https://www.slant.co/topics/764/%7Ebest-web-servers