SOFTWARE SHARKS

System Requirements & Design Document

for

NINSHIKI

Version 1.0.0 - Demo #5 Final

A document providing an overview of the Ninshiki system, a specification of the system requirements, and the architectural design of the system.

Compiled By

The Software Sharks Team

Coetzer MJ
Bester TJ
Orrie O
Lew J
Matodzi MC

Department of Computer Science
The University of Pretoria

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

For

Bramhope International School of Innovation

COS 301 Team: Software Sharks

Department of Computer Science, University of Pretoria

https://github.com/OrishaOrrie/SoftwareSharks

This Systems Requirements and Design document was drafted under the supervision of involved lecturers according to the assessment guidelines of the final year Computer Science module: COS 301 - Software Engineering, presented by the Department of Computer Science in the faculty of Engineering, Built Environment and Information Technology at the University of Pretoria during the first and second semester of the year 2018.

First release, September 2018

Contents

[1] Complete Domain Model Diagram Link

[2] Mobile Application Deployment Diagram Link
[3] Dashboard Application Deployment Diagram Link
[4] Google Cloud Platform Deployment Diagram Link
[5] System Interaction Deployment Diagram Link
[6] Complete Deployment Diagram Link

1 Introduction

The purpose of this document is to provide a detailed description of the requirements for the Ninshiki image recognition system. It will explain the purpose, features and architectural design of the system, as well as system constraints. This document is intended to be proposed to University of Pretoria lecturers and the project owner, as well as a reference for developing the first version of the system.

2 System Overview

The system's main requirement and primary purpose is to enable a user to take an image of an object, compare the image to a predefined model, and have the application respond with a labeled classification of the object.

The system is made up of two core components, namely, the web application and the mobile application. From this point on in the document, the web application and mobile application will be referred to as "the application", because both of these applications serve the same purpose and have similar interfaces with both the user and the back-end server.

The user will make use of the application device (the device that is running the application) to capture an image of an object with a Webcam or mobile camera device. Then, using the designed interface, the user will be able to push the image to the application. Once the application receives the image, the image gets processed by the image recognition model. After processing, the model outputs a prediction of the classification of the object. A list of the predictions is then created and returned to the user.

2.1 Purpose

The objective of the project is to develop an interface that is capable of using image recognition technology to accurately identify a specific object within an image and then return a response with the object's classification in the context of the client's warehouse inventory.

2.2 Project Scope

Ninshiki is an image recognition app specifically intended to identify engineering supplies in a warehouse in order to help employees of a warehouse correctly stock the supplies. The system will be able to recognise images, learn from those images, allow users to request a quote as well as help employees with the counting of supplies. The system will be both web and mobile based. The website can be accessed through any browser while the app will only be available to download on a phone which runs on the Android operating system.

The objective of the system is to help employees who are not familiar with engineering supplies to correctly stock the warehouse and add the supplies in the correct supply bins. This will benefit the warehouse as all supplies will be properly organised and employees will not waste time trying to figure out what the object is and where it belongs.

2.3 Definitions, Acronyms and Abbreviations

- BISI：Bramhope International School Of Innovation
- R#：Requirements number
- UC#：Use Case number

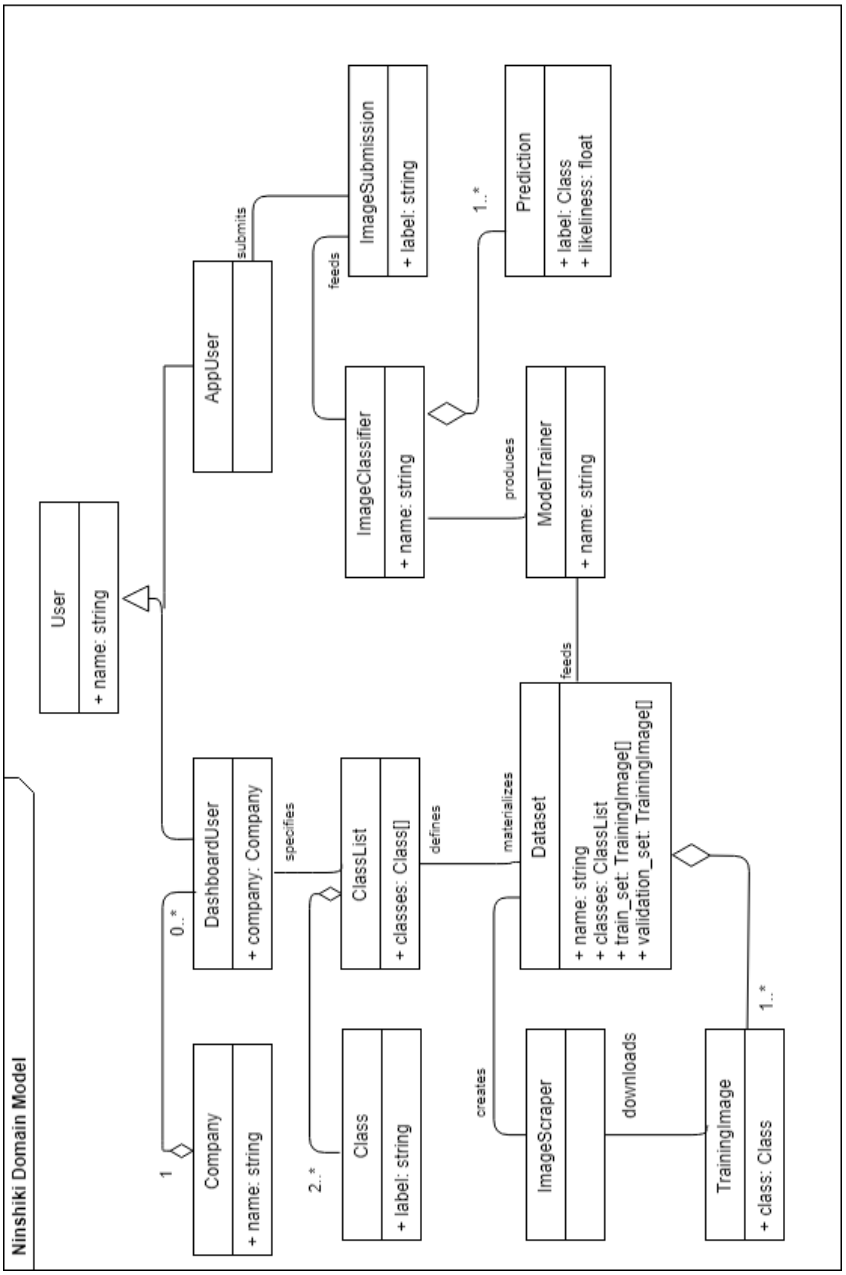
2.4 References

IEEE Software Engineering Standards Committee, "IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications", October 20, 1998.

2.5 UML Domain Model[7]

The domain model describes and models the project's real world entities. The entities can be discussed from the perspectives of the two kinds of users. The dashboard user is part of a company, and creates a dataset by specifying a list of classes (the categories which are to be predicted). This dataset is materialized by the image scraper, which downloads a few hundred images per class. This dataset is then input into the model trainer, which uses machine learning and a convolutional neural network to build an image classifier. The second type of user - the application user - submits an image to the image classifier, which then returns a list of predicted classes.
The domain model can be found on the next page.

[7] Complete Domain Model Diagram Link


Ninshiki Domain Model
User
+ name: string
AppUser
submits
ImageSubmission
+ label: string
feeds
Prediction
+ label: Class
+ likeliness: float
1..*
ImageClassifier
+ name: string
produces
ModelTrainer
+ name: string
feeds
DashboardUser
+ company: Company
0..*
specifies
ClassList
+ classes: Class[]
defines
materializes
Dataset
+ name: string
+ classes: ClassList
+ train_set: TrainingImage[]
+ validation_set: TrainingImage[]
Company
+ name: string
1
Class
+ label: string
2..*
creates
ImageScraper
downloads
TrainingImage
+ class: Class
1..*

3 Functional Requirements

- R1: A user should be able to capture or select a picture on a Mobile or Web application
  - R1.1: A user should be able to crop the chosen picture.
- R2: A user should be able to handle HTTP connections through a Mobile or Web application
  - R2.1: A user should be able to submit an image from a Mobile or Web application.
  - R2.2: A user should be able to receive and view information regarding a product matching their picture using image recognition on a Mobile or Web application
- R3: A user should be able to view existing catalogues from a server on a Mobile or Web application
- R4: A user should be able to process inventory through weight analysis on a Mobile or Web application
- R5: A user should be able to contact a company on a Mobile or Web application
- R6: A user should be able to create and train a model based on a dataset on a Web application.
  - R6.1: A user should be able to build a dataset by just specifying category names on a Web application.
  - R6.2: A user should be able to provide images to build a dataset on a Web application

3.1 Users

1. App User
   This user group makes up the largest portion of users of the system. These users are not technically inclined, or have no established knowledge of MRO items. The system helps these users identify items so that they can either handle them correctly in a warehouse environment, or look to purchase the items from the Bramhope online store.
2. Dashboard User
   This user group is made up of developers who are technically inclined enough to build their own application system, but are not

willing to build their own image classifier. The system aids these users by allowing them to simply define a list of classes for their own needs and have the system build an image classifier for them.

3.2 Subsystems

3.2.1 Web Application

This subsystem is an online version of the mobile application. (See section 3.2.2) The main aim of this application is to allow users that are unable to make use of the Mobile application. For example an iPhone user or someone who has a version of Android not supported by our application.

3.2.2 Mobile Application

This subsystem is used as a user interface that allows users to capture images and then submit them to be identified. They are then presented with probable results. It also allows users to do weight analysis for stock take. Lastly it allows users to contact the company.

3.2.3 Server

This is our back-end system. This is where we keep the prediction models, as well as the python scripts used for the Dashboard API (see section 3.2.4).

3.2.4 Dashboard API

This is a dashboard that allows users to define categories for a model. There is then an option to upload images or use an image scraper to create a dataset of images. It then uses this dataset to create a prediction model that the user can utilize.

3.3 Specific Requirements

3.3.1 Use Cases

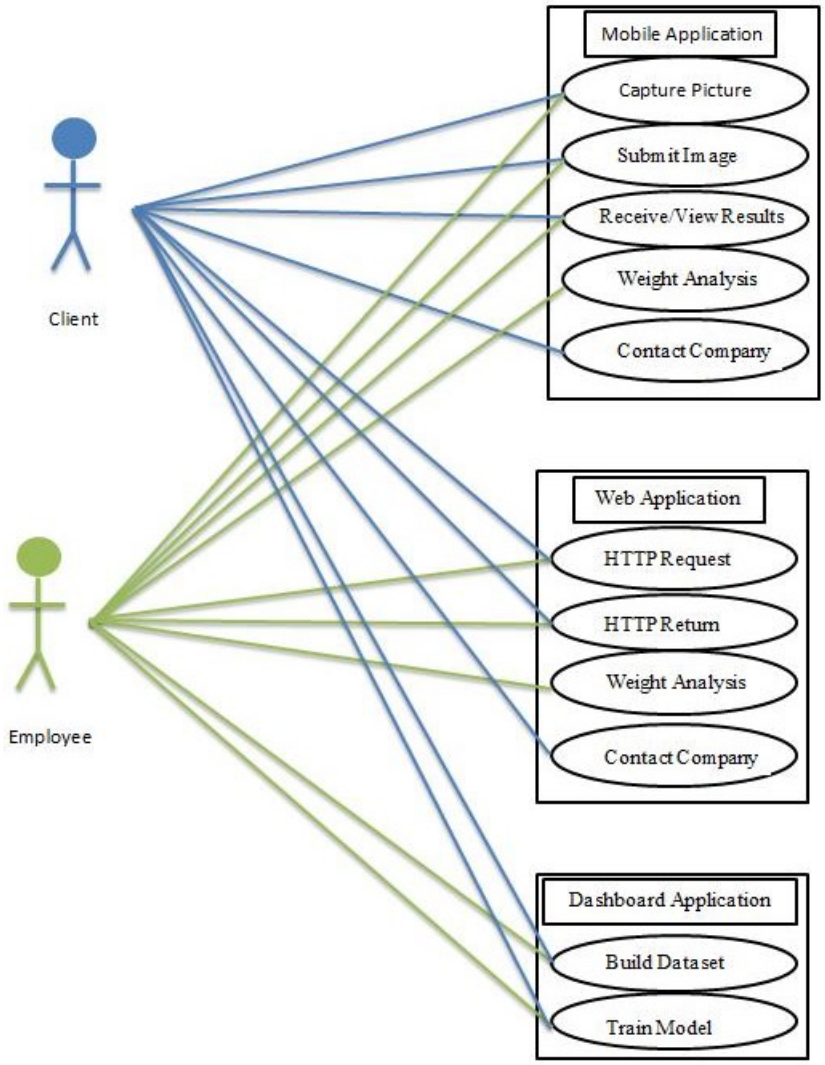
- UC1: Capture Image (Actor: User, System: Mobile/Web)
- UC2: Submit Image (Actor: User, System: Mobile/Web)
- UC3: Receive/View Results (Actor: User, System: Mobile/Web)
- UC4: Weight Analysis (Actor: User, System: Mobile/Web)
- UC5: Contact Company (Actor: Client, System: Mobile/Web)

- UC6. Build Dataset (Actor: User, System: Dashboard Web)
- UC7. Train Model (Actor: User, System: Dashboard Web/Server)

3.3.2 Traceability Matrix

Requirements	Priority	UC1	U2	U3	U4	U5	U6	U7
R1	2	X						
R1.1	5							
R2	1		X	X			X	X
R2.1	1		X	X				
R2.2	1			X				
R3	3			X				
R4	2				X			
R5	4					X		
R6	4							X
R6.1	4						X	X
R6.2	4							X
UC Priority		2	1	1	2	3	4	4

3.3.3 Use Case Diagram



4 Non-Functional Requirements

4.1 Performance Requirements

Performance of the system is one of the critical aspects of the project solution, since many of the use cases are enhanced by speed and accuracy of functionality. This section describes the performance requirements in terms of response time of the app subsystems, and the throughput of the server. Then, the performance of the image classifier will be described in terms of training and test prediction accuracy.

4.1.1 App Performance

The benchmarks used can be found in Google RAIL guidelines here:
https://developers.google.com/web/fundamentals/performance/rail

Response Time of the app is measured in terms of the time in which the first meaningful paint is done. This indicates how long it takes for the primary content of a page to load. The expectation of the app is to have a response time lower than 5000 ms. The web app achieves this with a response time of 1080 ms, largely thanks to the code minification techniques used by Angular and Webpack.

Server Throughput is measured in terms of how many invocations can be handled by the server within a period of time. The expectation of the server is to allow for at least 10,000 invocations per month. This requirement is met, since the Firebase server allows for up to 125,000 invocations per month. This was a big reason why Firebase was used, as well as for the next requirement.

Server Operation Throughput is measured in terms of how many operations can be handled by the server specifically for serving the image prediction model to the client side. The expectation for the server is to allow for at least 250,000 download operations per month. This requirement is met, since the Firebase storage bucket allows for up to 50,000 download operations per day.

Model Load Time is measured in terms of how long it takes for the image prediction model to be loaded into memory, starting from opening the app. The expectation is that the model should be loaded within 10 seconds. This requirement is met on average, although results depend on the Internet connection.

Image Prediction Time is measured in terms of how long it takes for a prediction to be made on an image, starting from when the Predict button is clicked. The expectation is that a prediction should be presented within 5 seconds. This requirement is met most of the time, with exceptions occurring mostly on the first prediction of a new session. This performance is largely as a result of the change of the image prediction model. Details on this can be found in the Testing Policy.

4.1.2 Image Prediction Performance

The performance of the actual image classifier is one of the biggest factors of the system's success. The following requirements are the minimum for any image classification task. Various machine learning techniques were used and tests conducted in order to build a model that met the requirements. More in-depth details about this can be found in the Testing Policy.

A Training Accuracy of at least 90% and a Validation Accuracy of at least 70% should be achieved. This indicates that a model can successfully be fit on a given dataset. A top-1 Test Accuracy of at least 70% and a top-5 accuracy of at least 90% indicates that the classifier has a good generalization rate, i.e. is good at predicting images that it has not seen before.

4.2 Quality Requirements

Quality requirements refer to various attributes that add value to the system and contribute to the project's success. These attributes include availability, scalability, recoverability, and reliability.

4.2.1 Availability

This attribute refers to the degree to which the system is operational and accessible for use. This can be measured in terms of the availability of the web and mobile apps, and the operating time of the backend server. Each of these availability requirements are met with the use of Google's Firebase, as it provides reliable hosting capabilities, always-accessible storage buckets, and high-throughput cloud functions, nearly 100% of the time. Firebase eliminates the need to separate the hosting of the web app, the online storage of the models, and the server functions, by providing a high-quality, enterprise-standard, serverless cloud platform.

4.2.2 Scalability

This attribute refers to the degree to which the system can grow in usage while still being able to function optimally. This requirement can be measured in terms of the throughput and storage space of the server and cloud storage buckets. Once again, Firebase helps to meet these requirements by offering an enterprise-standard service that allows for scalable usage of both cloud functions and cloud storage buckets, albeit at a premium.

4.2.3 Recoverability

This attribute refers to the measure of preparedness in the case of system failure and the ability for a system to recover from a fail state. The failure of the website hosting and the cloud functionality is dependent on and recovered from by Firebase, which uses all of Google's resources to ensure that there is a high degree of recoverability. The use of Travis Continuous Integration also assists in the recoverability of a newly updated build, as it can prevent builds containing errors from being deployed to Firebase.

4.2.4 Reliability

This attribute refers to the degree to which the system can fulfill its purpose and meet user requirements for a certain period of time. This can be measured in terms of mean time between failure. Once again, both Firebase and Travis-CI are used to ensure a high degree of reliability.

4.3 Security Requirements

The security of the system is related to vulnerabilities that may expose the system to threats and attacks which may cause it to be in an unwanted state. Security involves creating countermeasures to these vulnerabilities to prevent threats or mitigate the effect of attacks. Security requirements are important in preserving the confidentiality, availability, and integrity of the system assets.

In terms of finding vulnerabilities and creating countermeasures within the internal structure of the system, unit testing is performed to ensure that no software flaws can be exploited.

In terms of external vulnerabilities, the following will be done to protect system assets:

- Internal penetration testing is performed to ensure that no external vulnerabilities can be exploited.
- The entire system will be backed-up online, to ensure availability.
- The dashboard subsystem will use Firebase's OAuth to ensure user authentication and data confidentiality and integrity.
- The user applications will not share submitted image data, due to the predictions being done on-device.
- The user applications will not expose user data when submitting an email query or feedback response, due to usage of the safer POST HTTP method (as opposed to GET).

4.4 Interface Requirements

Interface requirements cover a broad spectrum of the usability of the system, as well as its hardware and software interfaces and their respective capabilities. Since the largest subsystems of the project are the user applications, as well as the user-oriented approach of the Agile methodology, usability is a crucial factor in the project's success. In terms of the dashboard subsystem (image classifier builder), being able to interface with various kinds of software is important.

4.4.1 User Interface

The effectiveness of the user interface refers not only to the look and feel of the interface, but also the ease of use, interaction, error handling, and documentation of the interface. Since a large portion of the user base will be warehouse workers, many of whom are not technically inclined, app usability is an important requirement. Two methods are employed to ensure that this requirement is met: obtaining user feedback and obtaining information from a UX consultant. Details on these two methods can be found in the appropriate section of the Testing Policy.

4.4.2 Accessibility

A quality attribute that adds value to the system is its accessibility. This increases ease of use for target users and allows for more users to use the system. The Web Content Accessibility Guidelines (WCAG) are used to help meet this requirement, more of which can be found here: https://developers.google.com/web/fundamentals/accessibility/

5 System Architecture

5.1 System Type

The system was identified to be a transformational system, because it consists of information-processing activities to transform input to output. Specifically, the input is the user's image of an object, the information processing is the image recognition model processing the image, and the output is the object classification made by the image recognition model. Furthermore, there is little interaction between the user and the system while the server processes the image. The system is also stateless. All the aforementioned factors helped determine that the system was a transformational system.

5.2 Overall System Architecture

The system as a whole can be seen as two core subsystems, the application and the server. The two primary activities that are performed by the system is the application sending the server an image, and the server responding to the application with an image classification. For this reason, the overall system architectural style was identified to be a Client Server architectural style, with the server being the " component and the several instances of users using the application to upload images being the ". Further reasoning for this is that the back-end server is a designated subsystem that provides services to several clients or application users. The server is located remotely from the clients, and the server has no knowledge of the client details; it only knows that it must send a response to the client after it has received a request. The clients, or the application users, send a request (the image) to the server in expectation of a response (the classification)

5.3 Interfaces

1. **User Interfaces** The entire approach to setting up user interfaces revolves around the concept of modularity. Each aspect of the UI is designed in such a way that it can be loaded dynamically and independently from one another. Allowing for optimal code reuse and maintainability. It stems from a native MVC approach adapted from Angular and Ionic respectively. This also allows for "lazy loading" which only displays the aspects of the interface that are currently visible to the user.

   Web Interface:

   The web application utilizes a systematic and clean user interface designed to guide the user towards the intended function-

ality.

Functionality:

- UC1: Capture Image
- UC2: Submit Image
- UC3: Receive/View Results
- UC4: Weight Analysis
- UC5: Contact Company

Mobile Interface:

The mobile application utilizes a systematic and clean user interface designed to guide the user towards the intended functionality.

Functionality:

- UC1: Capture Image
- UC2: Submit Image
- UC3: Receive/View Results
- UC4: Weight Analysis
- UC5: Contact Company

Dashboard Interface:

The dashboard application utilizes a systematic and clean user interface designed to guide the user towards the intended functionality.

Functionality:

- UC6: Build Dataset
- UC7: Train Model

2. **Hardware Interfaces**
3. **Software Interfaces**

5.4 Architectural styles

5.4.1 Web Application

Type: Interactive

The user initiates and interacts with the system to jointly accomplish the image prediction functional requirements

Architectural Style: N-Tier

Based off of Angular MVC Setup.
View ⇒ Displayed web page - Point of interaction with user
Model ⇒ Predefined data structures representing view templates and data sets
Controller ⇒ The typescript classes

5.4.2 Mobile Application

Type: Interactive
The user initiates and interacts with the system to jointly accomplish the image prediction functional requirements

Architectural Style: N-Tier
Based off of Ionic MVC Setup.
View ⇒ Displayed web page - Point of interaction with user
Model ⇒ Predefined data structures representing view templates and data sets
Controller ⇒ The typescript classes

5.4.3 Dashboard Application

5.4.4 Google Cloud Platform

Firebase Hosting

Type: Client Server
Sub-System provides an interface that allows a client to request and successfully receive a web page from the server.

Architectural Style: Client-Server

Firebase Functions

Type: Event Driven
Sub-System exhibits state-dependant reactive behaviour by waiting on a feedback trigger to enable an email to be sent on behalf of the user.

Architectural Style: Event Driven System Architecture

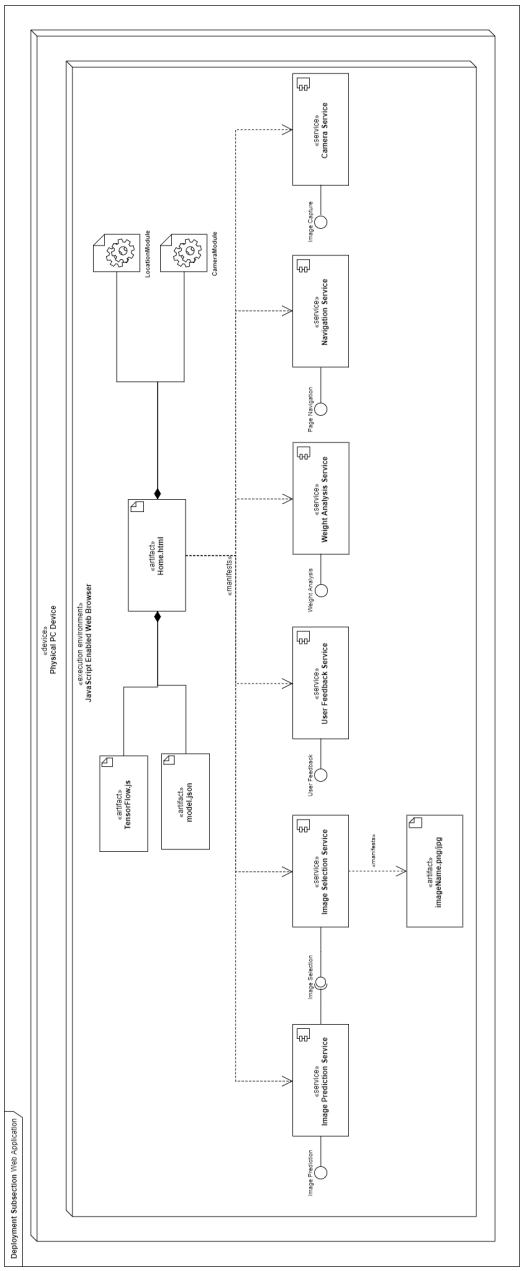
Firebase Storage

Type: Object Persistence
Provides object persistence and sharing between applications as well as shielding business objects from changes in database implementation.

Architectural Style: Persistence Framework

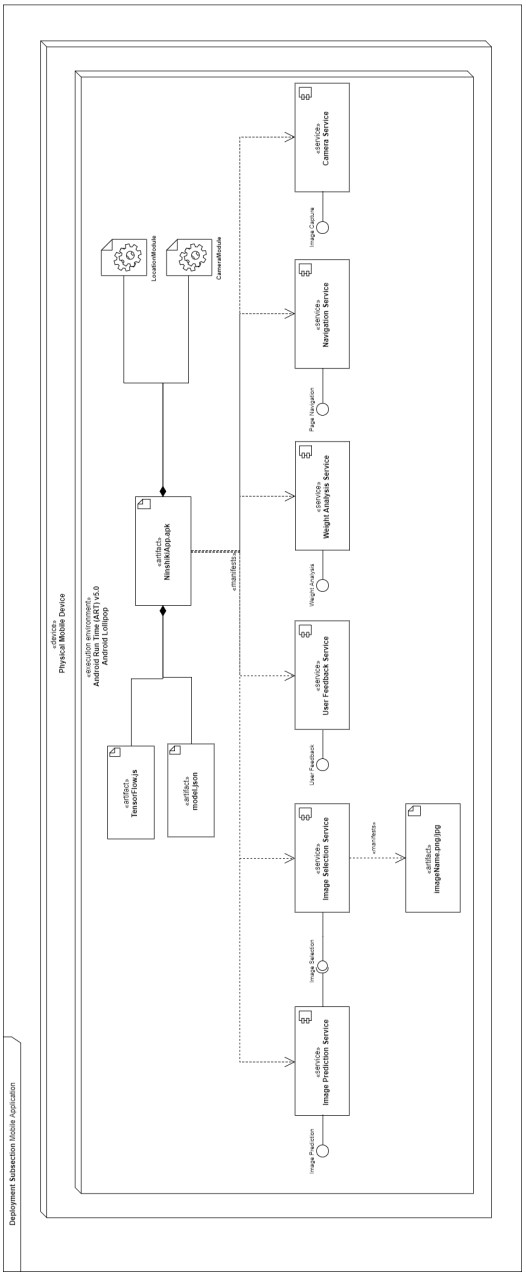
5.5 System Configuration

5.5.1 Subsystem: Web Application [8]



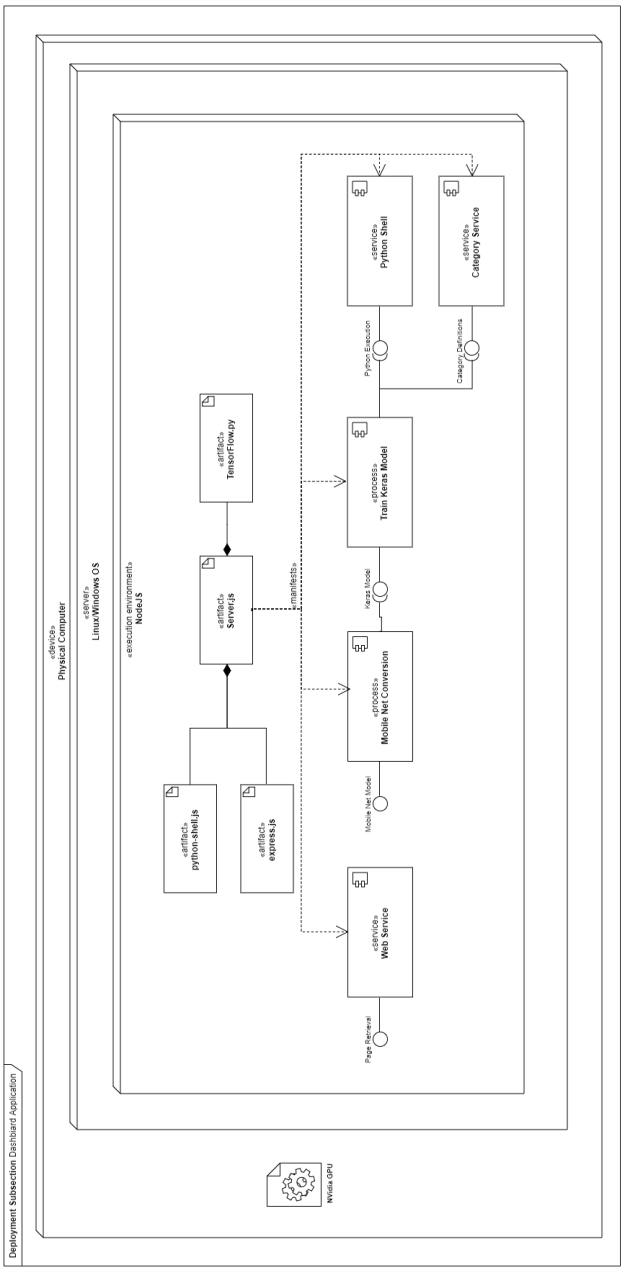
[8] Web Application Deployment Diagram Link

5.5.2 Subsystem: Mobile Application [9]



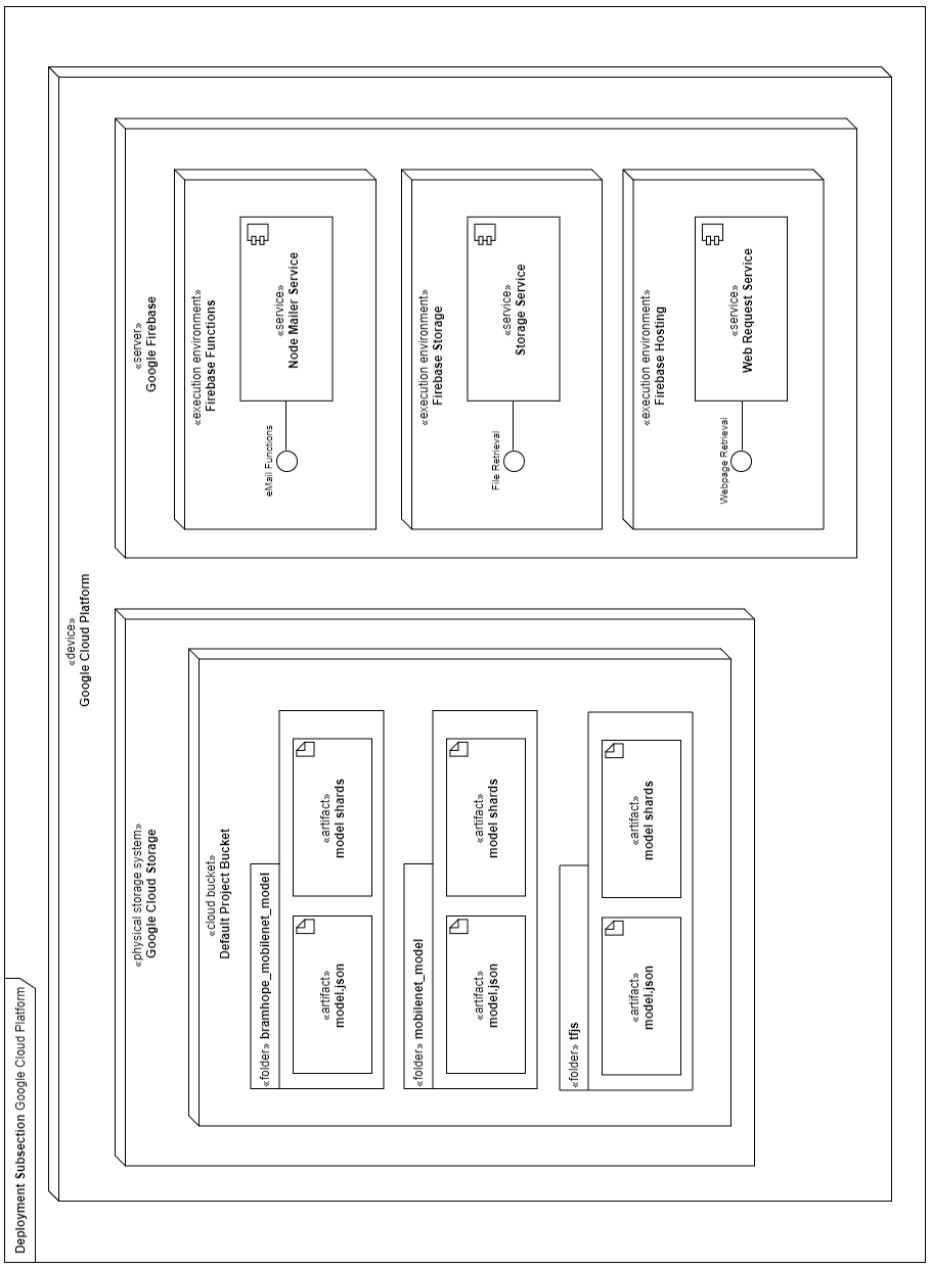
[9] Mobile Application Deployment Diagram Link

5.5.3 Subsystem: Dashboard Application [10]



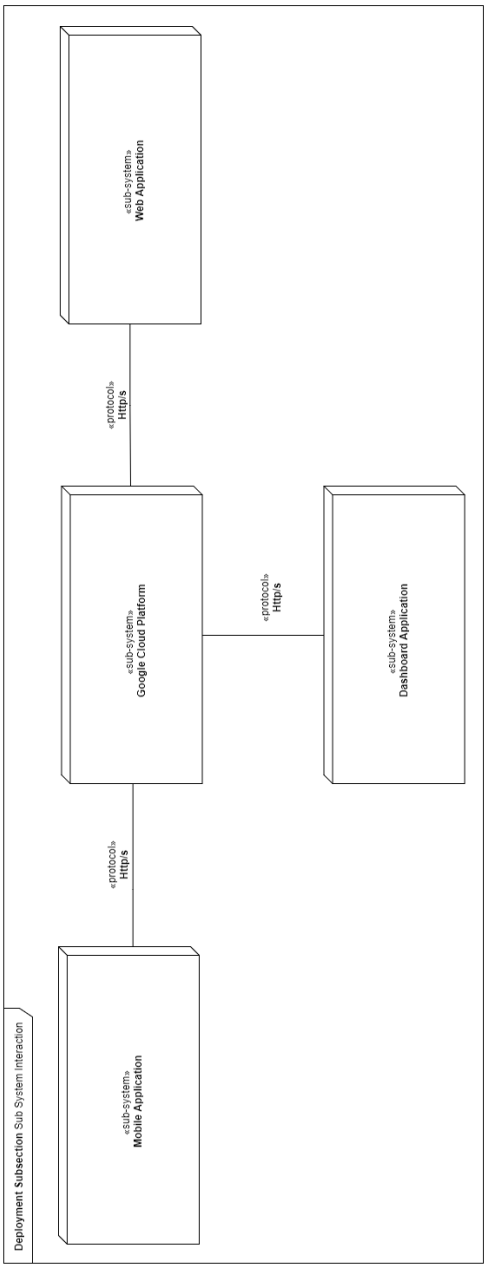
[10] Dashboard Application Deployment Diagram Link

5.5.4 Subsystem: Google Cloud Platform [11]



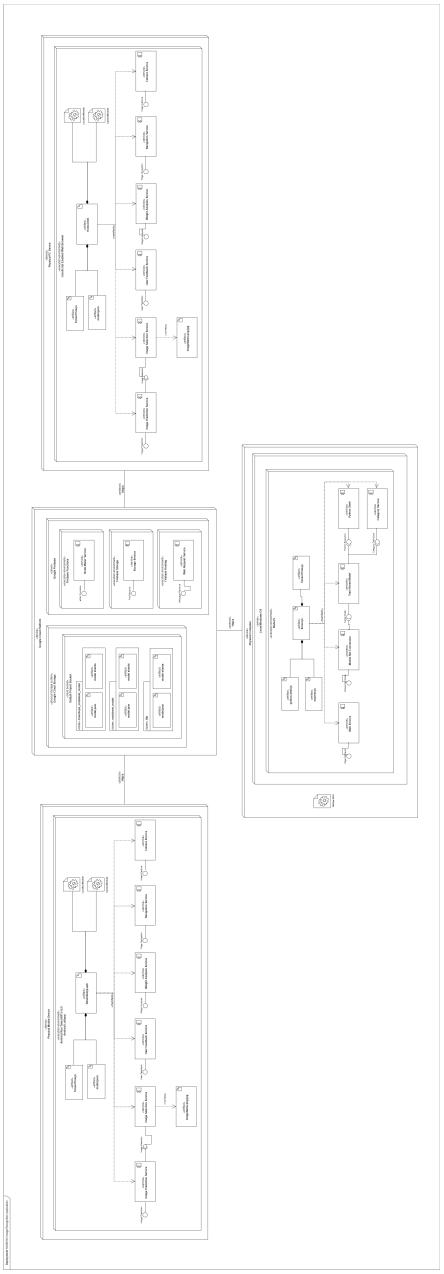
[11] Google Cloud Platform Deployment Diagram Link

5.6 System Interaction [12]



[12] System Interaction Deployment Diagram Link

5.7 Complete Deployment Diagram [13]



[13] Complete Deployment Diagram Link