Software Sharks
Department of Computer Science
University of Pretoria
Lynwood Road
Hatfield, Pretoria
0002
012 420 4111

Architectural Specification

for

BISI - Image APP

Version 2.0

Prepared by:

Mark Coetzer	u14044537
Orisha Orrie	u13025199
Tobias Bester	u14041368
Len Bekker	u11026953
Mukundi Matodzi	u16091265
Jonathan Lew	u13318765

On:

10th April 2018

Revision History

Revision Number	Revision Date	Summary of Changes
001	10-04-2018	Created initial document

Table of Contents

1. Introduction

1.1 Document Overview

This document will first introduce the purpose and abbreviations used in this document (in this section). Then, an overview of the system and the system's type will be identified and discussed. Following this, the system's overall architecture will be made clear with the use of a diagram. Finally, the subsystems will be identified along with the architectural style of each subsystem, each accompanied by a diagram.

1.2 Document Purpose

The purpose of this document is to give an indication of the system's overall architectural style (along with that of the subsystems), so that it can be used as an artifact for conceptualizing, constructing, managing, and evolving the system under development.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition

2. System Overview

2.1 System Description

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

The system is made up of three core components, namely, the web application, the mobile application, and the back-end server. 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 upload their image to the back-end server.

Once the server 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. This classification is labeled, returned by the server to the application, and displayed to the user.

2.2 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.

3. System Architecture

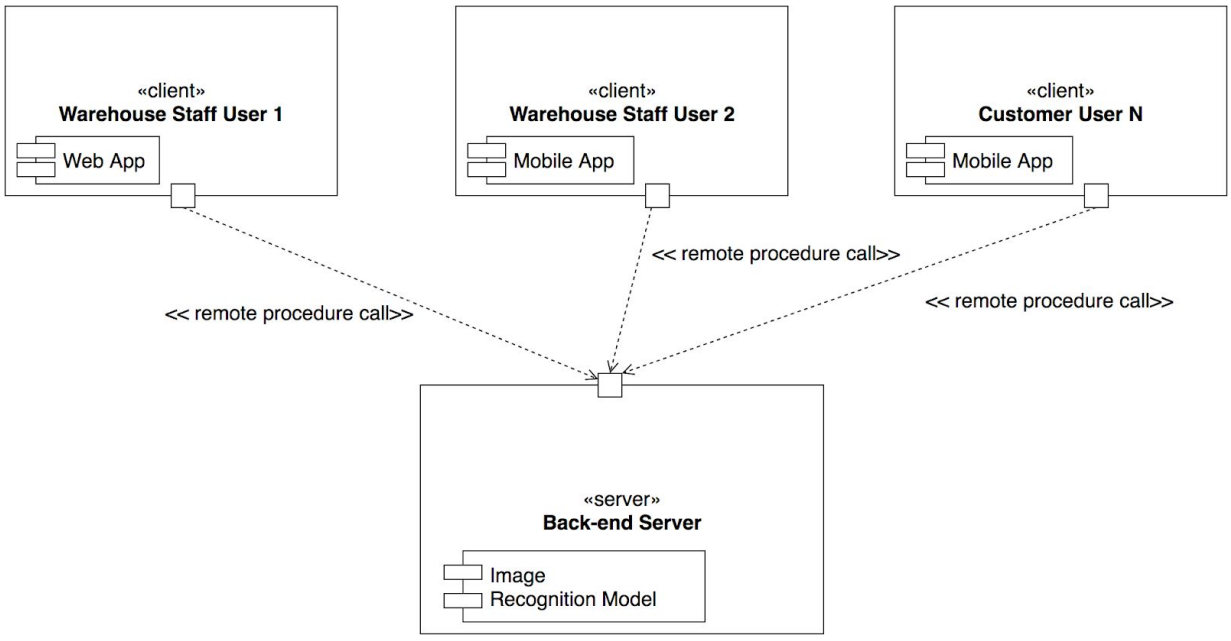
3.1 Overall System Architecture

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 “Server” component and the several instances of users using the application to upload images being the “Clients”.

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).

A structural diagram of the architecture is shown in Figure 3.1.

Figure 3.1



3.1 Subsystem Architecture

The two core subsystems of the overall system are the application and the server, as previously stated. Although both of these subsystems play a role in the overall architecture as a client and server, on their own, they are whole new systems and as such, are more than “just” client and server components.

The two subsystems and their respective architectural styles will be discussed further.

3.1.2 Server Subsystem

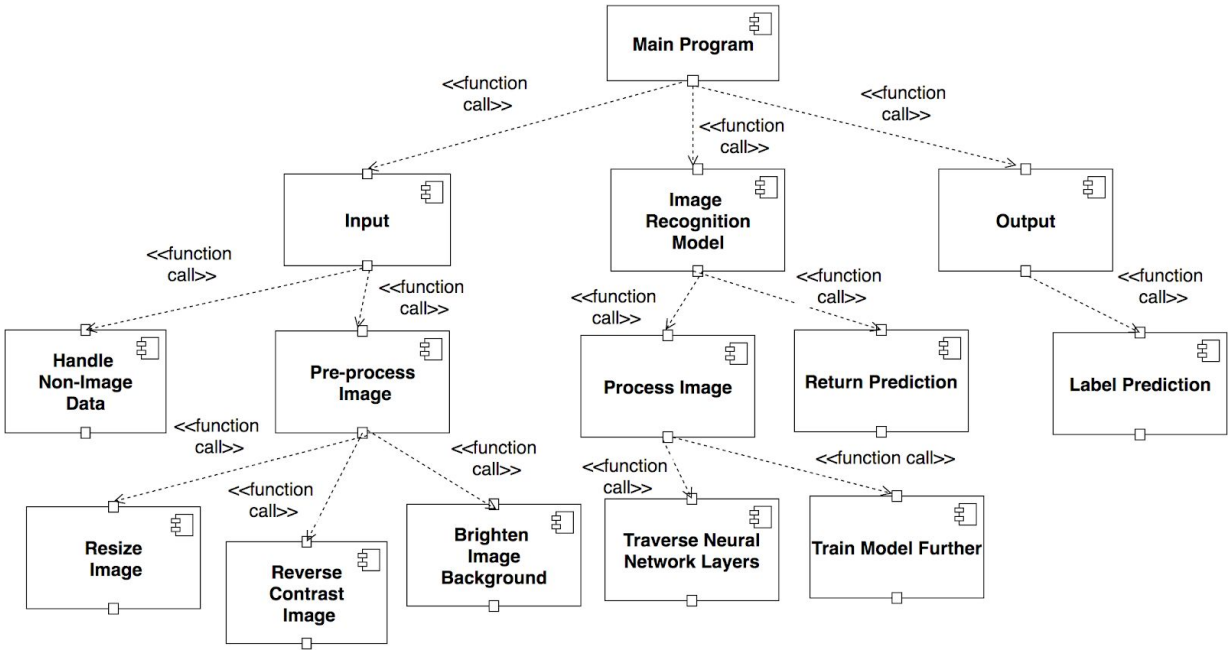
The back-end server has the main role of transforming the information received into a response that it sends back. Therefore it was identified to be a transformational system. The overall system was also identified as a transformational system due to its core purpose, but could have also been identified as a client-server type.

The server has additional purposes aside from running the image through the image recognition model, such as storing the model’s weights and all its subcomponents, as well as being able to preprocess the image. For this reason, and the fact that the system is transformational, it was identified that the system should use a Main Program and Subroutines architectural style.

Further reasoning for this is that the server has a main function that runs all the necessary steps to get a classification, and each of these functions themselves have functions.

The structural diagram of the server architecture is shown in Figure 3.2.

Figure 3.2



3.1.2 Application Subsystem

The Application subsystem provides the user the ability to take an image, upload the image, receive a classification, as well as additional activities such as view a product on the catalog, request a quote, and complete weight analysis. It does this by providing a user interface and data manipulation and control to respond to user requests (at a simple level).

The subsystem was identified as an interactive system because the user-application (system-actor) interaction consists of a fixed sequence of user requests and application responses, such as the user pressing the “Request quote” button and the application displaying a “Request sent” message. The application also responds to each and every user request or interaction, after pressing a button, entering keyboard input, etc. Each instance of the application being run on a device only interacts with one user at a time. For these reasons, it was clear that the subsystem was an interactive system.

The application is built on the Angular front-end framework, which uses Components and a template (HTML) to define the “view”, which is what the user sees and interacts with. The Components make use of services which provide the functionality and data control of the application. In this way, the user interface, data control, and page functionality are kept separate, i.e. the principle of Separation of Concerns is followed.

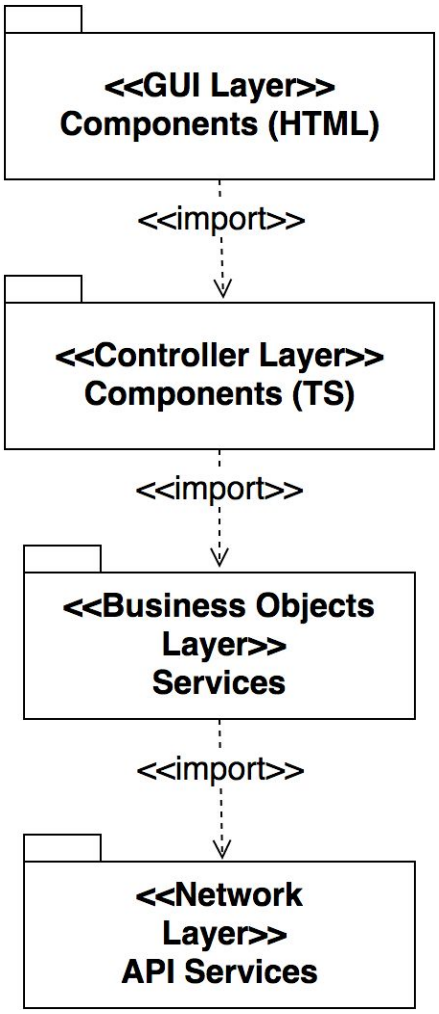
Since the system is an interactive type and there is a separation of concerns that exist at different levels, with each concern or function getting data from a neighbouring level, theN-Tier architectural style that was identified and applied to the Application system.

A user would interact with the application via the user interface and specifically via some “view” of the application. The Component responsible for this view provides the view’s functionality and then interacts with its respective service, which is responsible for controlling the data corresponding to the view. This service may then interact with an API for External System Integration or to process the data externally (receive user location). This aspect forms part of the network layer.

This process identifies the layers or “tiers” of the application, which both conceptually (logically) and physically make up the N-Tier architecture.

The package diagram of the Application architecture is shown in Figure 3.3.

Figure 3.3


<<GUI Layer>>
Components (HTML)
<<import>>
<<Controller Layer>>
Components (TS)
<<import>>
<<Business Objects
Layer>>
Services
<<import>>
<<Network
Layer>>
API Services