

Digital Marketing Project

N van der Linde

Project submitted for the degree *Baccalaureus Scientiae* in Information Technology at the North-West University

Supervisor: Dr. JT Janse van Rensburg

Co-supervisor: Mr. Z. Boonzaaier

Student number: 25909932

TABLE OF CONTENTS

| CHAPTER | 1 INTRODUCTION AND OVERVIEW 1 |
|---------|--------------------------------------|
| 1.1 | Introduction1 |
| 1.2 | Highlights1 |
| 1.3 | Challenges 1 |
| 1.4 | System Overview1 |
| CHAPTER | 2 USE CASE DIAGRAM3 |
| 2.1 | The Use of UML Diagrams3 |
| CHAPTER | 3 ERD DIAGRAM5 |
| 3.1 | The Use of ERD Diagrams5 |
| CHAPTER | 4 FLOW DIAGRAMS7 |
| 4.1 | The Use of Flow Diagrams7 |
| CHAPTER | 5 THE USER GUIDE10 |
| 5.1 | The Account Type Controller Class10 |
| 5.2 | The Repository Configuration Class11 |
| 5.3 | The General Response Class13 |
| 5.4 | The Account Transaction Class14 |
| 5.5 | Modulo and ModuloTest16 |
| CHAPTER | 6 CODE COVERAGE 18 |
| 6.1 | Code Coverage Report18 |

LIST OF FIGURES

| Figure 2-1: | Use Case Diagram | 3 |
|---------------|------------------------------------|----|
| Figure 3-1: | ERD Diagram | 5 |
| Figure 4-1: | Flow Diagram | 9 |
| Figure 5-1: | The Account Type Controller Class | 10 |
| Figure 5-2: | The Repository Configuration Class | 11 |
| Figure 5-3: | The General Response Class | 13 |
| Figure 5-4: | The Account Transaction Class | 14 |
| Figure 5-5-1: | The Modulo Class | 16 |
| Figure 5-5-2 | The Modulo Test Class | 17 |

CHAPTER 1 INTRODUCTION AND OVERVIEW

1.1 Introduction

This project is about designing an application for a South African digital marketing company. They have expressed the need for a web application that serves as an online platform allowing users to store, view and share photos.

Users should be able to create an account and login before utilising the platform. Once logged in, users should be able to view their own photos as well as the photos shared with them. The user should be able to create and manage the metadata recorded for each photo that the user has access to. The ability to share content with other registered users also needs to be built into the web application. Users should have the ability to download photos that they have access to.

The web application should be designed as consistent, intuitive, easy to use and easy to navigate without needing to provide excessive user guide documentation. All information and media items that are stored should be stored securely and in accordance with POPI.

1.2 Highlights

The highlights for this project was the ERD diagram as well as the Use Case diagram. Drawing these diagrams was valuable. It helps to stay updated on what Use Case as well as ERD diagrams need to look like or contain.

The weekly Client Sessions was another highlight. To be able to interact with someone working in this type of area was beneficial. It was a very good learning opportunity. She provided a lot of solutions and help with this project.

1.3 Challenges

The challenges for this project was the different technology stacks that was expected to be used to build this project. It was a bit difficult to implement all the different technologies in the project.

1.4 System Overview

Users within the digital marketing company currently send the photos back and forth over email. The users with whom the photos are shared are included in the recipients list of the email, with any additional information included in the body of the email. The additional information does not follow any specific format and is often excluded completely (or lost between versions of mail trails).

The current process sometimes creates confusion and miscommunication about which photos are the latest versions and which users the photos have been shared with. There is currently no defined and standardised way of recording any additional information relevant to the photos. There are also size limits when sending emails. These size limits are not often breached but when they are, the limitation needs to be mitigated. By switching to a centralised online platform, that all users have access to, the highlighted challenges can be mitigated and resolved.

The biggest risk to the project would be any limitations that a technology stack would present. Any technology limitations that prohibit the functional implementation of core requirements would need to be mitigated using a technology stack (or intermediate workaround) that alleviates the blocker or limitation.

The recommended approach for this business case is to pursue the development of an online platform that allows users to create, manage and share content with other users, with the ability to capture the applicable additional information (in the form of metadata). The content to be uploaded onto the platform should be grouped using albums and any content that has metadata recorded should be searchable by its metadata.

There are different technology stacks that need to be implemented for this project to run successfully. The following need to be used:

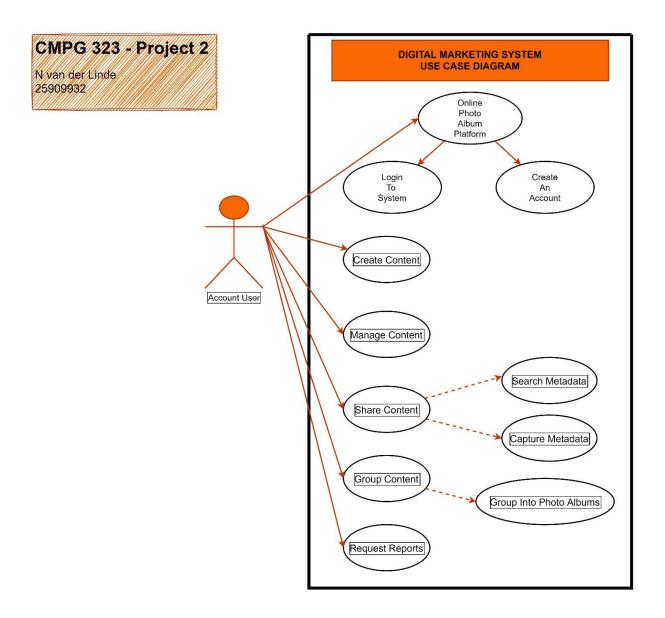
- GIT: this is used for the source control. GitHub was used for this project's source control.
- Backend (API): .NET Core was used for the backend of this project.
- Backend (Database): SQL server (a relational database in Oracle) was used to design the database.
- Frontend (Design): C# (ASP.NET) and CSS was used for the frontend.
- Cloud Technology: Google cloud was used to host this project.
- Diagrams: it is required to draw diagrams (ERD, Flow and Use Case diagrams) for this
 project. Any tool could have been used. For this project, draw.io was used to design the
 diagrams.
- Logging: the application will need logging. A logging framework was implemented.
- Code coverage: this is a type of tool, library, plugin, or framework that needs to be used to check the project's code coverage. A code coverage framework was implemented.

CHAPTER 2 USE CASE DIAGRAM

2.1 The Use of UML Diagrams

The Use Case Diagram is a diagram that depicts the interactions between the system and external systems and users. It graphically describes who will use the system and in what ways the user expects to interact with the system. It is also a set of modelling conventions that is used to specify or describe a software system in terms of objects. The use-case narrative is used in addition to textually describe the sequence of steps of each interaction.

Figure 2-1: Use Case Diagram



From the abovementioned diagram for the Digital Marketing System, the following can be derived:

- Online Photo Album System: the account user will either login to his or her account, using their unique username and password, or create an account.
- Create Content: the user can create photos, with the ability to capture the applicable additional information (in the form of metadata) and any content that has metadata recorded should be searchable by its metadata.
- Manage Content: the user can manage photos, with the ability to capture the applicable additional information (in the form of metadata) and any content that has metadata recorded should be searchable by its metadata.
- Share Content: the user can share photos, with the ability to capture the applicable additional information (in the form of metadata) and any content that has metadata recorded should be searchable by its metadata.
- Group Content: the account user makes use of this so that the content that was created,
 viewed, or stored can be uploaded onto the platform and grouped using photo albums.
- Request Reports: an extra Use Case added. This is for the user to see what content was created, viewed, or stored. It is also to show the metadata created.

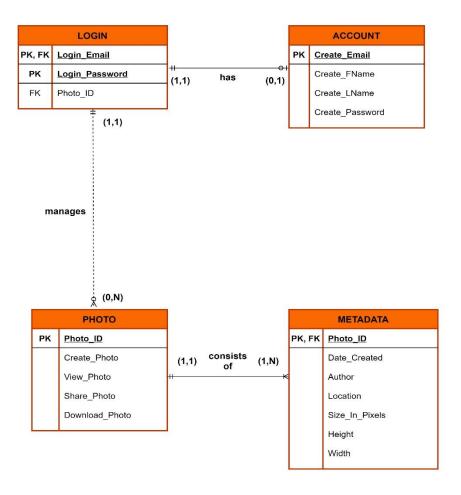
CHAPTER 3 ERD DIAGRAM

3.1 The Use of ERD Diagrams

The ERD diagram is a model or blueprint representing the technical implementation of the database. Therefore, the technical implementation for the Digital Marketing System is shown below:

Figure 3-1: ERD Diagram





From the abovementioned diagram for the Digital Marketing System, the following can be derived:

- Login: the Login entity is for the user to login into his or her account. The username, consisting of the user's email, and password have to be entered to login. The photo_id is for the user to enter that will take the user directly to their photos on the system.
- Account: this is if a user does not have an account on the Digital Marketing System.
 Then an account can be created by entering the new user's email, followed by their first name, last name and finally a password also must be created.
- Photo: here the user can either choose to create photos, view photos, share photos with other users or download photos.
- Metadata: in this entity all photos' metadata can be changed or altered accordingly.
 Anything from the date that the photo was created, or the author, or the location, or the size in pixels, or the height and the width of the photo can be reformed.

CHAPTER 4 FLOW DIAGRAMS

4.1 The Use of Flow Diagrams

The flow diagram below demonstrates the different application layering for the Digital Marketing System.

First off there is a Welcome Page. Followed by a Login Page. Then a Create Account Page comes next. Finally, a Photos Page will be the last page for this web application.

Below is a proper explanation of the flow diagram designed for the Digital Marketing System:

1. The Welcome Page:

- a. The user must choose to either login to their account or to create an account if one does not have an account.
- b. From there, the next page is loaded.

2. The Login Page:

- a. This page loads if the user already has an account with the Digital Marketing System.
- b. Here the user enters his or her credentials consisting of a username, which is the user's email, and a strong password.
- c. Then when that is finished, the next page is loaded.

3. The Create Account Page or Register Page:

- a. This page loads if the user does not yet have an account with the Digital Marketing System.
- b. Here the user fills out the form. Their first name, last name, email as well as a strong password must be entered. The password also has to be confirmed by entering it again.
- c. Then when that is finished, the next page is loaded.

4. The Photos Page:

- a. This page loads once the user has logged in to their account.
- b. Here the user can either choose to create photos, view photos, share photos with other registered users or download their photos.

c. On this page the user can also manage the metadata of their photos on their account. They can either create new metadata for a photo or change the already existing metadata of a specific photo.

Lastly, all the services, layers, and pages connect to a Database.

Figure 4-1: Flow Diagram

CHAPTER 5 THE USER GUIDE

5.1 The Account Type Controller Class

The AccountTypeController Class:

 This Java Class makes use of the GeneralResponse Class created in the Domain service. It returns the appropriate response when the program is installed. It can either return a positive or a negative result, as specified below:

Figure 5-1: The Account Type Controller Class

```
<u> Pile Edit View Navigate Code Refactor Build Run Tools VCS Window Help</u>
                                       ⊕ ፲ ፲ ♣ • → # logbackxml × • RepositoryConfig.java × • AccountTypeController.java
     > 🖿 .idea
       AccountSystemLogic
       AccountSystemRepository
       AccountSystemTranslater
                   za.ac.nwu.as.web.sb
                                                                                    response = GeneralResponse.class),
@ApiResponse(code = 500, message = "Interna
```

5.2 The Repository Configuration Class

The RepositoryConfig Class:

This Class shows the configuration for the Repository service.

Figure 5-2: The Repository Configuration Class

```
> Im AccountSystemLogic
> Im AccountSystemRepo
> Im AccountSystemTrans
> Im AccountSystemWeb
                                                             import org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
                                                             import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseBuilder;
import org.springframework.jdbc.datasource.embedded.EmbeddedDatabaseType;
                                                              import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
     AccountSystemTranslater
      AccountSystemWebSpringBo
                                                             @coomingsretton
@EnableTransactionManagement
@EnableOpaRepositories("persistence")
@EntityScan("persistence")
@PropertySource(value = "classpath:application-db.properties")
                                                                                                                                                                                                                                                           ଓ № ± +
                                                                                                                                                                                                                                                           > # Acc
                                                                          EmbeddedDatabaseBuilder builder = new EmbeddedDatabaseBuilder();
                                                                                         .addScript("script/schema.sql")
.addScript("script/data.sql")
                                                                    public LocalContainerEntityManagerFactoryBean entityManagerFactoryBean() {
   final LocalContainerEntityManagerFactoryBean entityManagerFactoryBean =
```

```
### AccountSystem CDDCommoduce 50 entityManagerFactoryBean.setJpaphroperties(buildJpaphroperties(DJLT_NAME); entityManagerFactoryBean.setJpaphroperties(DJLT_NAME); entityManage
```

5.3 The General Response Class

The GeneralResponse Class:

 This Java Class is created under the Domain service. This Class is used by the AccountTypeController Class, as can be seen in Section 5.1. above, as well as the AccountTransactionController Class.

Figure 5-3: The General Response Class

```
🛂 <u>File Edit View N</u>avigate <u>C</u>ode <u>Refactor Build Run Jools VCS Window H</u>elp AccountSystem [C:\DO\nwu\AccountSystem] - GeneralResponse.java [Accoun
                                                                                                                                                                                                > ## A
       AccountSystemDomain
         src
            main
       AccountSystemLogic
       Repository AccountSystemRepository
      AccountSystemTranslater
       AccountSystemWeb
    Scratches and Consoles
                                                                             if (o == null || getClass() != o.getClass()) return false;
     > R AccountSystemLogic
                                                                                                                                                                                                   > I
     > RaccountSystemRepository
     > 📭 AccountSystemTranslater
                                                                        public String toString() {
     > RaccountSystemWeb
  > III External Libraries
    Scratches and Consoles
```

5.4 The Account Transaction Class

The AccountTransaction Class:

- This Class is created to keep track of a member's transactions. All transactions made or completed will use this Class. It links with the AccountType Class.
- It was designed according to the ERD. This program is not entirely the same as the ERD, it is just similar.

Figure 5-4: The Account Transaction Class

```
🛂 Eile Edit View Navigate Code Refactor Build Run Tools VCS Window Help 🗆 AccountSystem [C:\DO\nwu\AccountSystem] - AccountTransaction.java [Accou
                                                                                                                                                                                                      0 12 ▲ 15 ▲ 1 ^ ∨
                                                                                                                                                                                                                            > #A
                                                                                                                                                                                                                                 ###
> ■
        AccountSystemLogic
      AccountSystemRepository
        AccountSystemTranslater
        RccountSystemWeb
                                                                                    private AccountTransactionDetailsDTO details:
  > IIII External Libraries
     Scratches and Consoles
                                                                                    public AccountTransaction(long transactionID, String accountTypeMnemonic,
                                                                                          this.memberID = memberID:
this.amount = amount;
this.transactionDate = transactionDate;
                                                                                                                                                                                                      ⊕ 12 <u>A</u> 15 <u>A</u> 1 ^ ∨
                                                                                                                                                                                                                               > ## A
                                                                                                                           long memberID, long amount, localDate transactionDate,
AccountTransactionDetailsDTO createAccountTransaction) {
                                                                                          this.accountTypeMnemonic = accountTypeMnemonic;
this.memberID = memberID;
                                                                                                                                                                                                                               ###

> ■

> ■
        RecountSystemTranslate
                                                                                          this.accountTypeMnemonic = accountTransaction.getAccountTypeMnemonic();
this.memberID = accountTransaction.getMemberID();
```

```
> #
                                    借A
                                    private long getAmount() {
    return amount;
             Account
              © Account
         > De service
   m target
  AccountSystemLogic
  AccountSystemTranslate
 AccountSystemWeb
AccountSystemWebSpri
                                                                                                                                                                           ##. €
##. €
                                    @JoinColumn(name = "accountTypeID")
public AccountType getAccountTypeID() { return accountTypeID; }
                                    public String getAccountTypeMnemonic() { return accountTypeMnemonic; }
                                                                                                                                                      9 12 A 9 A 1 ^
                                                                                                                                                                        > 編 A

> 編 A

> 日

> 日
                                   public void setMemberID(long memberID) { this.memberID = memberID: }
                                   public void setAmount(long amount) { this.amount = amount; }
                                   public AccountTransactionDetailsDTO getDetails() { return details; }
                                                                                                                                                                           AccountSystemReposit
AccountSystemTransla
  AccountSystemWeb
                                       if (this == 0) return true;
if (o == null || getClass() != o.getClass()) return false;
                                                                                                                                                                           AccountSystemTransla
Scratches and Consoles
```

5.5 Modulo and ModuloTest

The Modulo Class:

This Java Class will calculate what the mod will be of two integers. A Runtime Exception
is added to indicate whether there is an error present when the divisor equals null. If not,
then the program will return the answer.

Figure 5-5-5: The Modulo Class

```
🖳 File Edit View Navigate Code Refactor Build Run Iools VCS Window Help AccountSystem [C:\DO\nwu\AccountSystem] - Modulo.java [AccountSystemLogic]
                                                                                                                                                                                     \textbf{AccountSystem} \ \rangle \ \textbf{AccountSystemLogic} \ \rangle \ \mathsf{src} \ \rangle \ \mathsf{main} \ \rangle \ \mathsf{java} \ \rangle \ \mathsf{flow} \ \rangle \ \textcircled{\textcircled{\textbf{@}}} \ \mathsf{Modulo}
   Project ▼ 🕀 🗵 🛠 🗘 — 🏭 logback.xml × 📇 ModuloTest.java × 📇 Modulo.java ×

✓ Markage flow;

AccountSystem C:\DO\nwu\Acco 1 package flow;

      > ■ .idea
      → ■ AccountSystemDomain 3 public class Modulo

→ ■ AccountSystemLogic 4 {

✓ ■ main

                   > 🗖 config 8
                          CreateAccount 10
                  © FetchAccount 12
© FetchAccount 12
© ModifyAccour 13
© Modulo 14
> ■ za.ac.nwu.as.logic 15
■ test 16
             > test
      > RaccountSystemRepository
       > AccountSystemTranslater
       > RaccountSystemWeb
       > R AccountSystemWebSpringBo
      Scratches and Consoles
```

The ModuloTest Class:

• This Java Class tests the Modulo program, as explained above. Two integers are inserted for the dividend and the divisor, respectively. The result is printed out as an integer value. Then, another program tests if the two integers inserted can be divided by zero. This program has an exception error thrown out when the answer is equal to zero.

Figure 5-5-6 The Modulo Test Class

```
🛂 Elle Edit View Navigate Code Refactor Build Run Iools VCS Window Help - AccountSystem [C:\DO\nwu\AccountSystem] - ModuloTestjava [AccountSystemLogic]
  ✓ AccountSystem C:\DO\nwu\Ac
                                  package za.ac.nwu.as.logic.flow;
                                                                                                                                 Indexing... 😘 🔩
                                                                                                                                                           m
    > R AccountSystemLogic
    > R AccountSystemRepository
    > R AccountSystemTranslater
    > RaccountSystemWeb
    AccountSystemWebSpringE
  > III External Libraries
   Scratches and Consoles
                                      private Modulo mod;
                                      public void testMod()
                                          assertEquals("This should be equal to 4.", 4, result.intValue());
    → AccountSystemTra
    > R AccountSystemWe
    > AccountSystemWe
  > IIII External Libraries
   Scratches and Consoles
  ≔ TODO 🛭 Problems 🛭 Terminal 📚 Dependencies
```

CHAPTER 6 CODE COVERAGE

6.1 Code Coverage Report

The code coverage report does not want to show. I did insert the correct dependency, the JaCoCo dependency, according to the videos provided. I tried to install my project and extract the report from *target*, *site* and then open the *index.html* file in Browser, but it did not want to work.

You are more than welcome to have a look at the JaCoCo dependency on GitHub, where I have uploaded the entire Digital Marketing System's code.