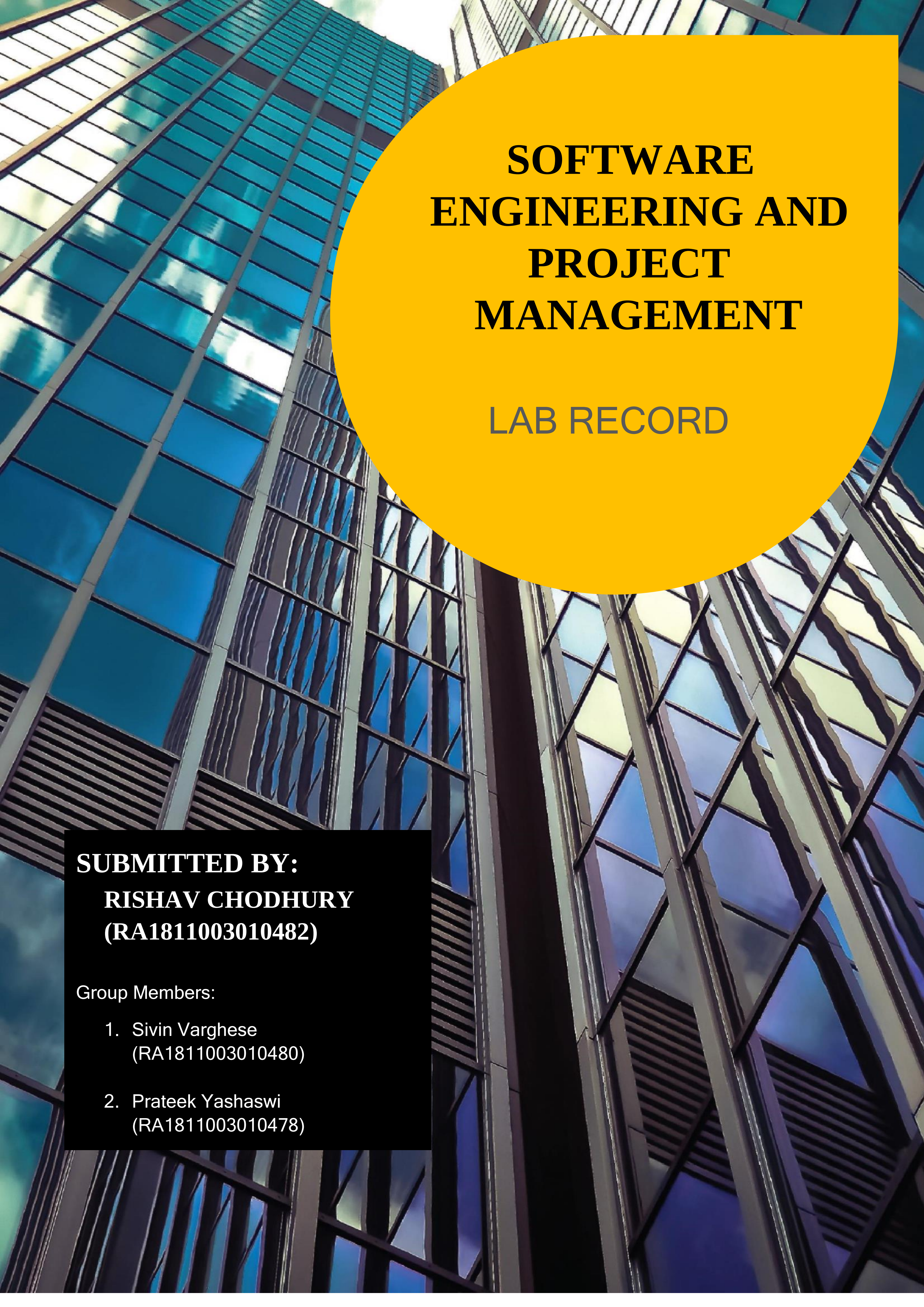


SOFTWARE ENGINEERING AND PROJECT MANAGEMENT

LAB RECORD

SUBMITTED BY:

RISHAV CHODHURY (RA1811003010482)

Group Members:

1. Sivin Varghese (RA1811003010480)
2. Prateek Yashaswi (RA1811003010478)

BUSINESS CASE TEMPLATE

AIM:

To generate a Business Case Template for a QR based, Inventory Management System called **CarGO.**

PROJECT –

- The project attempts to simplify and enhance the existing inventory management and goods shipment system.
- A software application that shall identify packages using their QR Code.
- Arrival and departure of goods and its records are maintained in a backend database that can only be authorized personnel.
- The frontend has an engaging and easy-to-use UI, so that people with 0 technology knowledge, can withstand how to operate it.
- The Search option has various fields to show current inventory.

THE HISTORY –

- Existing system is manual, where package IDs are entered into logbook.
- System is not updated at the moment goods arrive or are shipped.
- Even for computerised systems, tediously long tracking IDs, need to be generated to mark the packages uniquely.
- No search and find options are available in the pre-existing systems.

LIMITATIONS –

- The system or mobile device needs to be connected to the internet to all times.
- Employees need a mobile device or a QR Code Scanner.
- Cached data maybe redundant for large number of packages and hence occupy memory.

APPROACH –

- Android Studio
- Google Lens
- Firebase

BENEFITS –

- System is simple and designed for user’s comfort. No technical expertise is required for the application.
- No additional resources are necessary.
- System are based on QR Codes, bot long and complex Tracking IDs.
- Search, Find and Tracking options are available.

RESULT:

The business case study was successfully completed for the intended project carGO.

STAKEHOLDER REGISTER

AIM:

To enlist all the stakeholders of the **carGO** Project, their interest in the project, their role and impact on the outcome of the project.

Project Stakeholders Name	**Specific Information Needs**	**Project Interests**	**Impact on Project**	**Role**
	Types and Frequency of Communication	Specific Areas of Interest and Participation	Positive, Negative, Influencer, Support, Roadblock	Decision Maker, Collaborator, Participant, Consultant, Information Recipient
Mr. Nikola Kovacic	Occasional Consultation	To make an interactive UI and sync the backend database.	Influencer	Consultant
Ms. Audrey Esparaza	Occasional Counselling	Can contribute to the success by advertising the application on open platforms.	Support	Paid Collaborator
Mr. Sivin Varughese	Daily Updates	Can develop frontend and backend application on Android Studio.	Positive	Decision Maker (Team Lead)
Mr. Rishav Chowdhury	Daily Designing	Can contribute design elements and add an uncluttered look to the UI.	Positive	Participant (UI Designer)
Mr. Prateek Yashaswi	Daily Programming	Manage backend database and Firebase Application.	Positive	Participant (Firebase Developer)

RESULT:

The Stakeholder Register was made and all the important stakeholders were enlisted along with their roles, impacts and interests.

USER STORY

AIM:

To list the User Story of every Stakeholder along with Success Criteria of their specific job profile.

1. **User Story: Nikola Kovacic –**

   As a Consultant, I want to bring my expertise, so that the team is well led.

   Success Criteria:

   - The UI is clean.
   - The UI is user friendly.
   - The UI is optimised.
   - The UI is should not have any bugs.

2. **User Story: Audrey Esparaza –**

   As a Collaborator, I want to advertise the product across social media and digital media platforms, so its outreach pans out to a larger audience and it gets more traction, therefore becoming successful.

   Success Criteria:

   - The user application is advertised across the maximum number of platforms and media.
   - The Public Relations for the product will be handled through proper channels.

3. **User Story: Sivin Varughese –**

   As the Team Lead, I want to guide the team through my technical and entrepreneurial expertise, so that a robust work ethic is promoted.

   Success Criteria:

   - The team should work efficiently.
   - The team should work diligently.
   - The team should work do the job which will result in optimal output of our project.

4. **User Story: Rishav Chowdhury –**

As the frontend designer, I want to design an uncluttered User Interface with easily navigable tools and options, so that user experience is enhanced.

Success Criteria:

- The User Interface is free of faults and bugs.
- The User Interface has an uncluttered look and is easily navigable.
- The user experience is enhanced and the User Interface is easy to use, so that technical expertise is not required to operate it.

5. **User Story: <u>Prateek Yashaswi</u> –**

   As the backend designer, I want to synchronise an effective backend database, with proper SQL queries, so that functionality of the application is maintained.

   Success Criteria:

   - The SQL Queries are robust and well framed.
   - The synchronisation of the backend database is automatized.
   - The database has well defined information fields.

<u>RESULT:</u>

The User Story of every Stakeholder was listed and collated together.

REQUIREMENT GATHERING

AIM:

To identify the various types of requirements of this project.

Theory:

Requirements are defined during the early stages of a system development process, as a specification of what should be implemented or as a constraint of a specific kind on the system.

Types of Requirements:

- System Requirements
- Functional Requirements
- Non-Functional Requirements
- User Requirements

For our project, i.e. **carGO** – an automated database management system, based on QR Codes, the requirements are identified as follows –

1. **User Requirements :**
   - User Requirements are basically the use cases for a project. It describes how the whole system will work in the hands of a user and all the possible scenarios that and user can face while operating the product.
   - Our application will be used for inserting a new entry and updating the status, once the item in question is withdrawn from the inventory for delivery.
   - Along with these options, a tab for searching any entry is given is also included in the application, in the scenario that the user to look up any item present in the inventory.

2. **System Requirements:**
   - A System Requirement is a set of documents which describes the features and behaviour of the system or the software application.
   - Our application will consist of clean and simple frontend User Interface with separate tabs for operations like **Entry, Update** and **Search.**
   - When new items enter the inventory, the **Entry** option provides the functionality of launching a QR Scanner and indicating the user to scan the code on the product.
   - Once the code is scanned, it will generate a new row in the database and then add the essential information fields regarding the product.
   - Similarly, the **Update** and **Search** options will be executed frim their respective tabs.

- We can use **Google Lens** and **Mongo DB/Firebase** for our application.

3. **Functional Requirements:**
    - A Functional Requirement is a description of the services that the software application must offer. The functional requirements should include complete and comprehensive information about the workflow, data handling logic and outputs generated by the system.
    - When our application will reach in the hands of users, it needs to satisfy all the possible scenarios it is being put into. For example, if a particular user wants to update the contact number of any customer or any remark is needed to be added for any product, such functionalities should be implemented easily.
    - Along with these functionalities, if the user wants to sort the whole database according to the data and time of entry or departure, or alphabetically or according to some other criterion, there must be an option provided in the application to satisfy this functionality.

4. **Non-Functional Requirements:**
    - Constraints on the services are functions offered by the system.
    - Our project will provide like **Entry, Update, Search** for any item in the inventory but if anyone wants to update the name of the item in any case of misspelt words, it cannot be done in any case.
    - Our application will require maintenance once a week, and the servers will be non-functional for a period of 30-45 minutes.

RESULT:

All the necessary requirements for the **carGO** were identified and studied successfully.

PROJECT PLAN

AIM:

To identify roles and responsibilities and calculate Project Effort.

Roles and Responsibilities:

NAME	ROLE	RESPONSIBILITIES
Mr. Sivin Varughese	Team Lead	• Look over the entire quality of the technical deliverables • Facilitates Team level change review process • Design, Implement and Manage software programs and backend
Mr. Rishav Chowdhury	UI Developer	• Design the Frontend of the Project • Provide interactive elements and uncluttered look to the UI
Mr. Prateek Yashaswi	Firebase Developer	• Design and Manage the database for the Project • Maintain and Update the database during the Application Maintenance Phase
Ms. Audrey Esparaza	Paid Collaborator	• Advertising the application on all open platforms • Look for companies who need an automated database managing app.
Mr. Nikola Kovacic	Consultant	• Suggest improvement in the application performance • Sync UI and Backend Database

Estimation of Project Effort:

- The scenario in which the team presently finds itself in, is one where partial data is available.
- Hence, accurate financial analysis is not a feasible option.
- We can estimate the project cots and effort using the technique of Function Point Analysis.

Information Domain Values:

Measurement Parameters	Count	Simple	Average	Complex	Total
Number of User Inputs	2	3	4	6	8
Number of User Outputs	5	4	5	7	25
Number of User Queries	3	3	4	6	12
Number of Internal Logic Files	4	7	10	15	40
Number of External Logic Files	3	5	7	10	21

Unadjusted Function Point (UAF) = 106

Complex Weighing Factors:

- Does the system require backup and recovery? – 5
- Is the UI interactive and bug-free? – 4
- Does the system differentiate between different levels of administrative access? – 3
- Does the system require online data entry? – 5
- Is the internal processing complex? – 1
- Is the redundancy managed efficiently? – 5
- Is performance critical? – 4

$\sum F_i = \underline{27}$

AFP = UFP * CAF

*[Where Complex Adjusting Factor (CAF) = 0.657 + (0.01 * $\sum F_i$)]*

Adjusted Function Point (AFP) = 100.70

Programming Language – OOP

$LOC / FP_{avg} = \underline{30}$

LOC / FP = 3021 ~ 3 KLOC (Kilo Lines of Code)

3 KLOC signifies **Organic Model**

$a_b = \underline{2.4}$, $b_b = \underline{1.05}$, $c_b = \underline{2.5}$, $d_b = \underline{0.38}$

$E = a_b * (KLOC)^{b_b} = \underline{7.66\ PM}$

Development Time (D) = $c_b (E)^{d_b}$ = 5.42 Months

RESULTS:

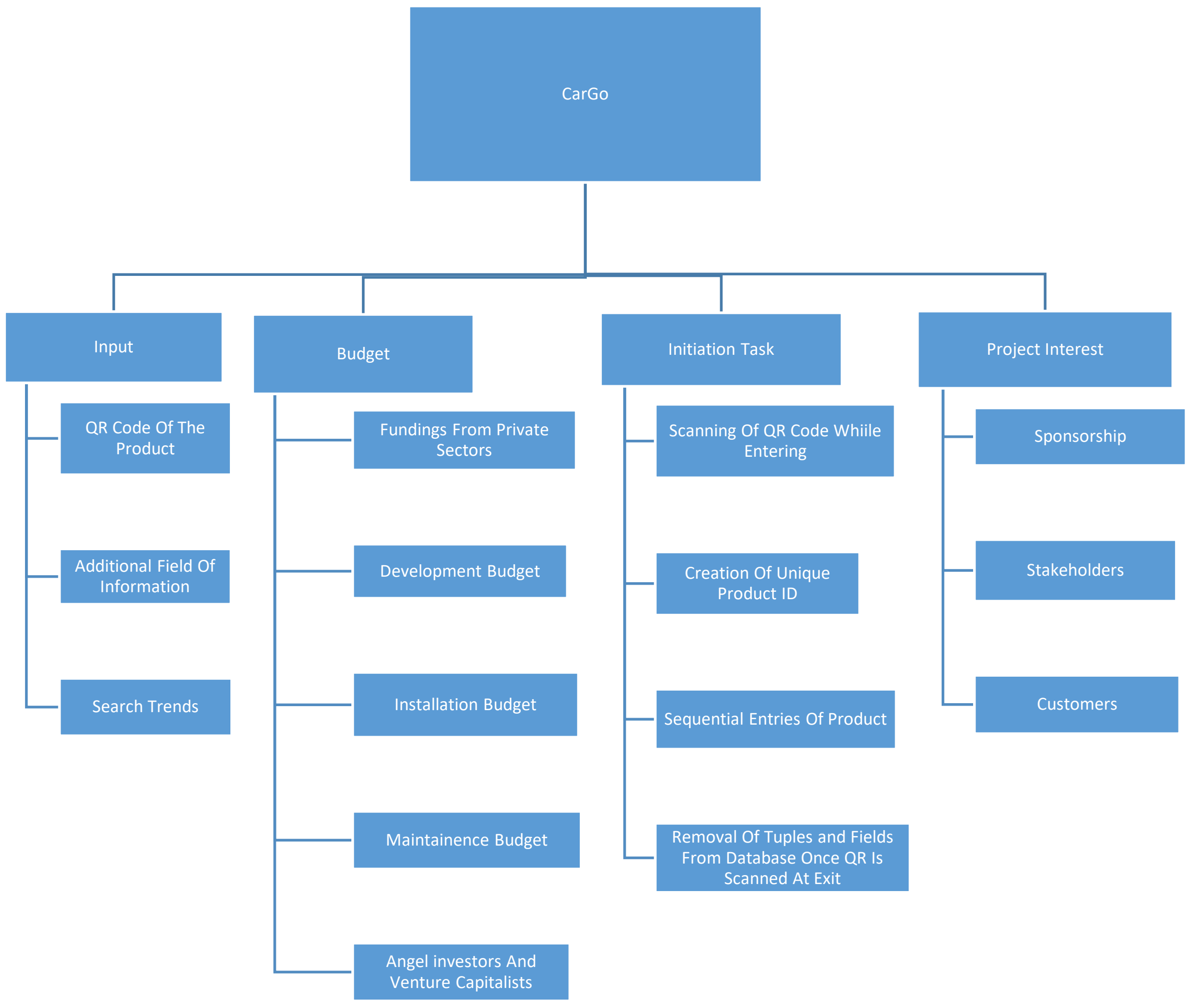
Job Roles and Responsibilities were identified and Effort and Development Time were calculated successfully,

WORK BREAKDOWN STRUCTURE & RISK ANALYSIS

AIM:

To carry out WBS, Timeline and risk analysis with respect to the current project.

Diagrammatic Representation:


CarGo
Input
QR Code Of The Product
Additional Field Of Information
Search Trends
Budget
Fundings From Private Sectors
Development Budget
Installation Budget
Maintainence Budget
Angel investors And Venture Capitalists
Initiation Task
Scanning Of QR Code Whiile Entering
Creation Of Unique Product ID
Sequential Entries Of Product
Removal Of Tuples and Fields From Database Once QR Is Scanned At Exit
Project Interest
Sponsorship
Stakeholders
Customers

Work Breakdown Structure (WBS):

In terms of project management, the phase WBS is a deliverable oriented breakdown of a project into smaller modules or components.

Risk Identification:

It is a systematic attempt to specify threats to the project plan. By identifying risks, the project manager can rectify the problems and try to minimize the risk as much as possible.

Our project **CarGO**, needs to be maintained almost every day because of a lot of database entries are adding to the root file in every session and needs to be stored in the cloud storage for further use.

SWOT Analysis:

SWOT analysis helps us to identify the four crucial factors about our project which are as follows:

- Strengths
- Weaknesses
- Opportunities
- Threats

- **Strengths:** The positive points about our projects
- **Weaknesses:** The negative points about our projects
- **Opportunities:** The upgrades that can be implemented in our project.
- **Threats:** The external factors that can be impede the progress of the project

STRENGTHS	WEAKNESSES
Automated database manager Easy sorting of data User friendly UI No data redundancy	Inefficient for large amount of data
OPPORTUNITIES	**THREATS**
Database sorting algorithm can be optimized Compatibility on Platforms like Firebase, Mongo DB	Coming up with these QR code ideas is easy and that will increase the competition in the market.

Risk Mitigation Template:

RESPONSE	STRATEGY	EXAMPLE
Avoid	Risk avoidance is a strategy where precautions are taken to avoid risks.	• Extending schedule • Change in work ethic.
Transfer	Risk transfer involves transferring the threats and impact to third party	• Purchasing insurance • Performance bands • Warranties
Mitigate	Risk Management is a strategy where action is taken to reduce the damage caused	• Increasing testing • Reducing complexity
Accept	In this strategy, the team accepts the consequences and no action is taken	• Event contingency • Budget • Management

RESULT:

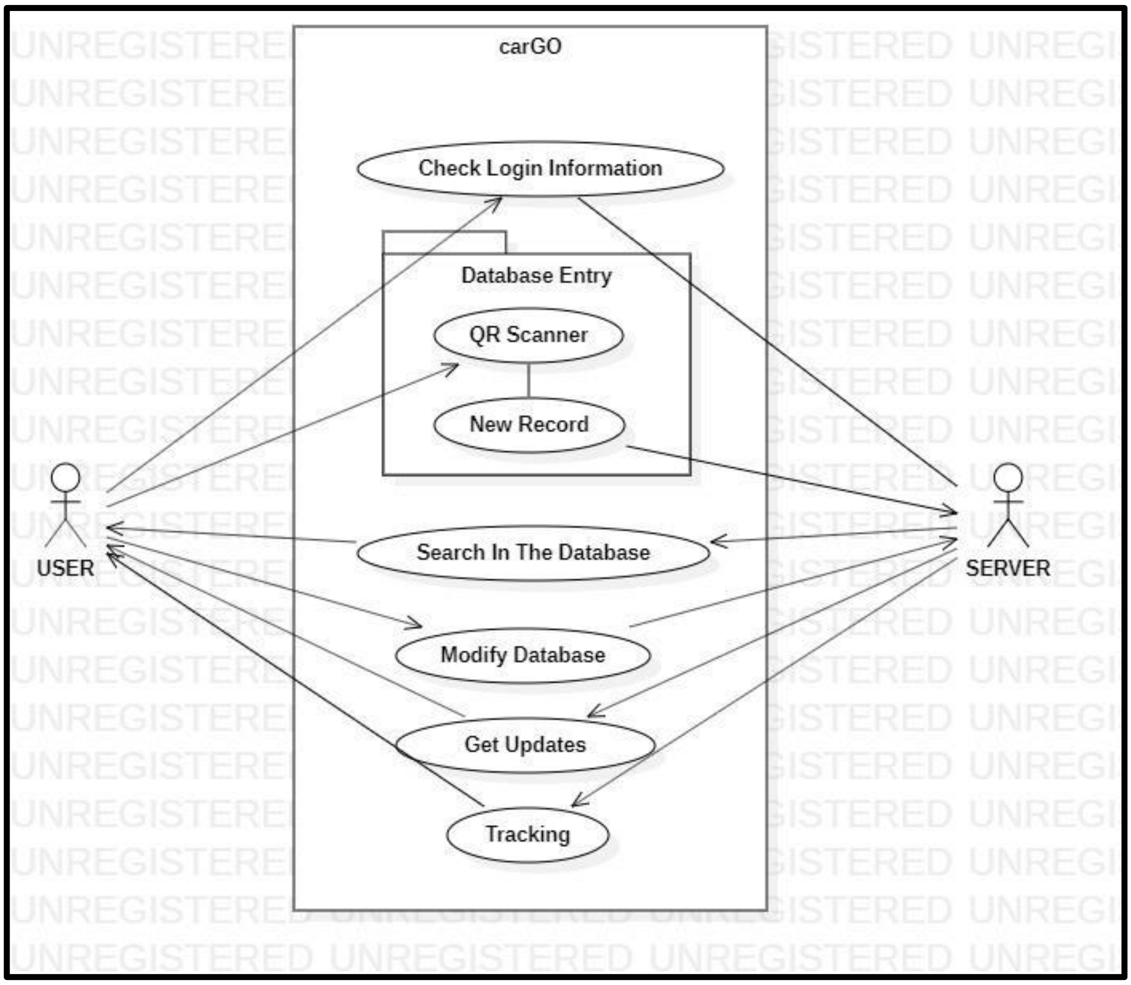
WBS, Timeline and Risk Analysis were carried out successfully with respect to the chosen project idea.

USE CASE DIAGRAM & PROJECT METRICS

AIM:

To generate a Use Case Diagram for the project and calculate the Project Metrics.

Use-Case Diagram:



Features of Use-Case Diagram:

- Functionalities to Be Represented As Use Case
- Actors
- Relationships among the Use Cases and Actors

Uses of Use-Case Diagram:

- Requirement Analysis and High Level Design.
- Model the Context of a System.
- Reverse Engineering.
- Forward Engineering.

RESULT:

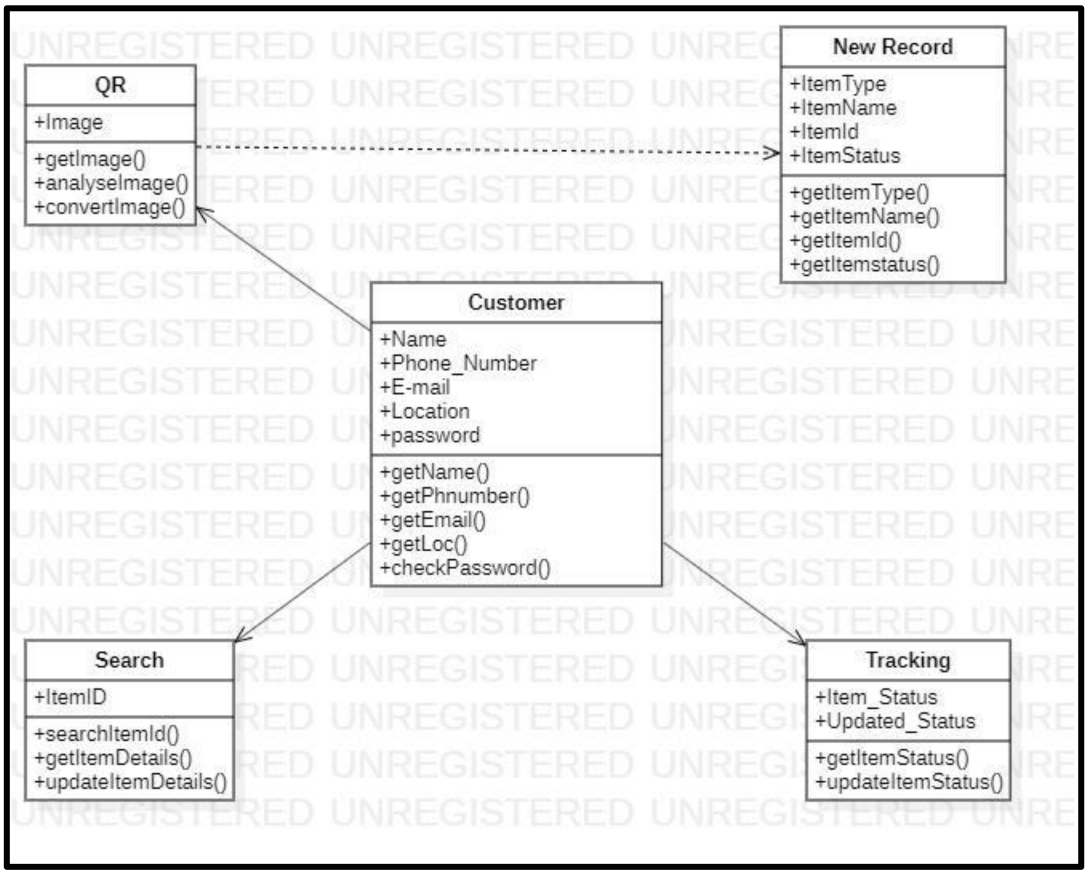
The Use-Case Diagram for the **CarGO** was successfully constructed and its uses were described.

CLASS DIAGRAM & ENTITY RELATIONSHIP DIAGRAM

AIM:

To generate the Class Diagram and Entity-Relationship (ER) Diagram for the **CarGO** Project, using the STAR UML software.

Class Diagram:



Features of Class Diagram:

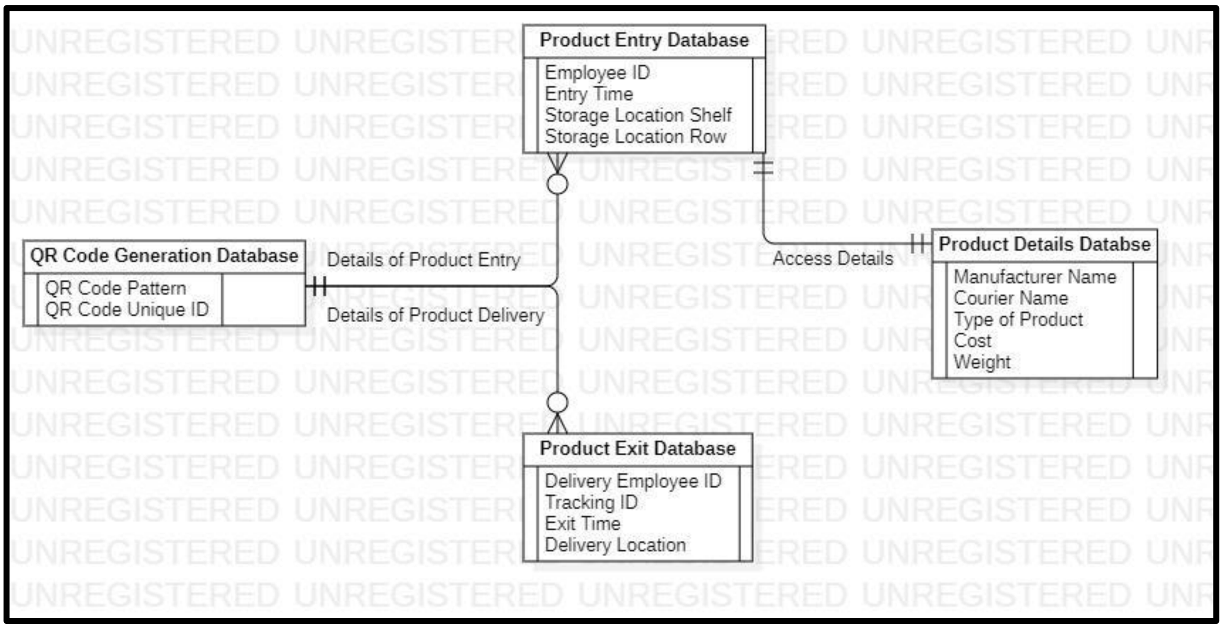
- Analysis and Design of the Static View of an Application.
- Describe Responsibilities of a System.
- Base for Component and Deployment Diagrams.
- Forward and Reverse Engineering.

Uses of Class Diagram:

- Describing the Static View of the System.

- Showing the Collaboration among the Elements of the Static View.
- Describing the Functionalities performed by the System.
- Construction of Software Applications using Object Oriented Languages.

ER (Entity-Relationship) Diagram:



Features of ER Diagram:

- ER Diagram stands for Entity-Relation Diagram.
- An Entity is an Object with Physical Existence.
- Set of all Entities is known as an Entity Set.

Uses of ER Diagram:

- ER Diagram shows the Relationship between all the Entities or Entity Sets present in the Project.
- Types of Relationships are listed as follows:
  1. One-To-One
  2. One-To-Many
  3. Many-To-One
  4. Many-To-Many

RESULT:

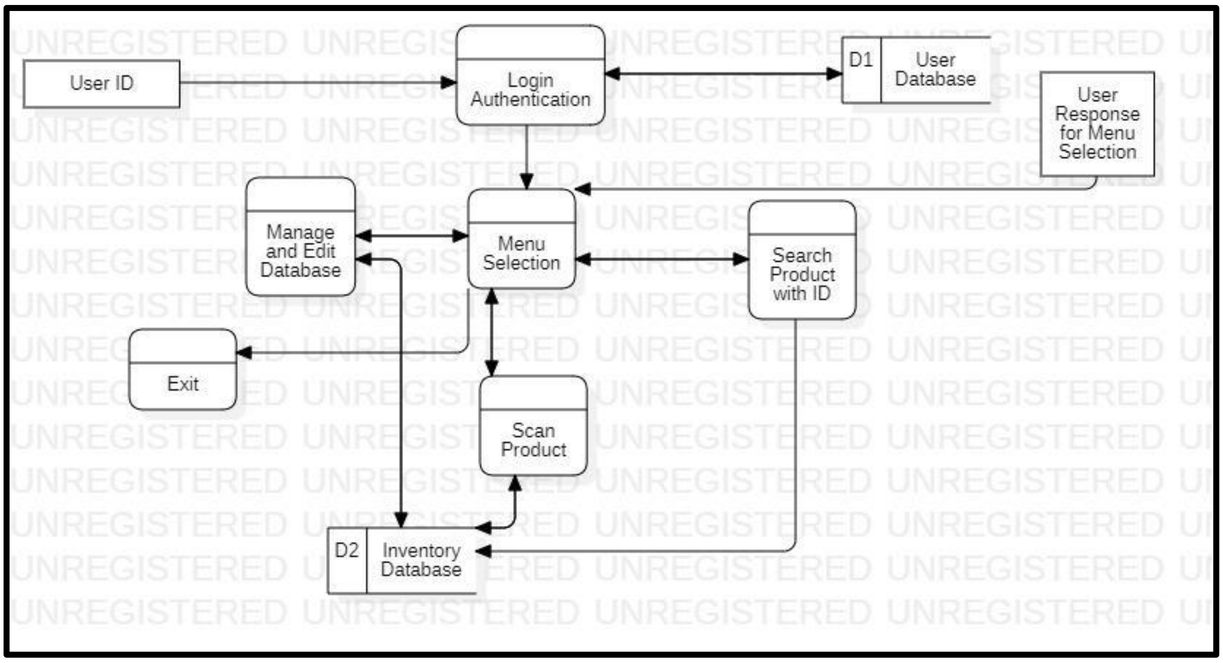
The Class and ER Diagrams for the **CarGO** Project were constructed in STAR UML.

DATAFLOW DIAGRAM & SEQUENCE DIAGRAM

AIM:

To generate the Dataflow Diagram and the Sequence Diagram for the **CarGO** Project.

Dataflow Diagram:



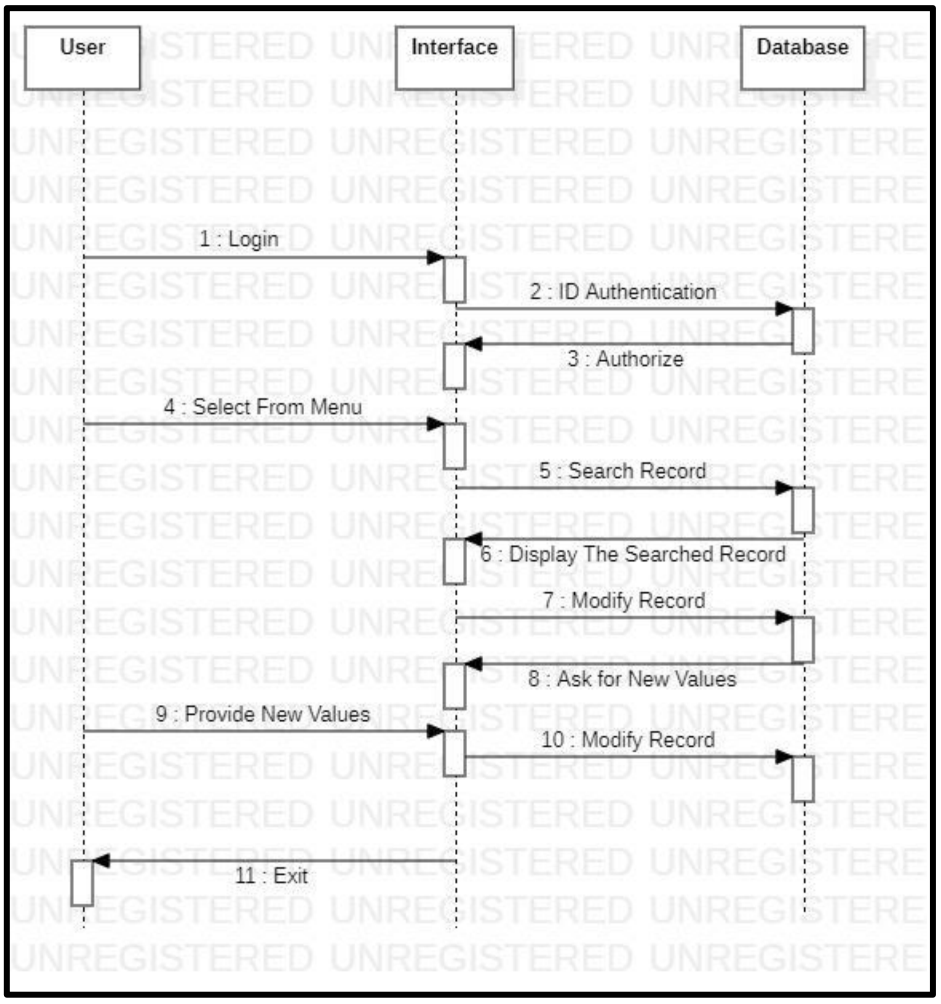
Features of Dataflow Diagram:

- Dataflow Diagram shows the Flow of Data of a Process or a System.
- It has no Control Flow.
- There are no Decision Rules or Loops.

Uses of Dataflow Diagram:

- **DFD in Software Engineering:** This is where Dataflow Diagrams got their main start in the 1970s. DFDs can provide a focused approached to Technical Development, in which more research is done up front to get to coding.
- **DFD in Business Analysis:** Business Analysts use DFDs to analyse existing Systems and find inefficiencies. Diagramming the Process can uncover steps that might otherwise be missed or not fully understood.
- **DFD in System Structures:** Any System or Process can be analysed in progressive detail to improve it, on both a Technical and Non-Technical basis.

Sequence Diagram:



Features of Sequence Diagram:

- Objects taking part in the Interaction.
- Message Flows among the Objects.
- The Sequence in which the Messages are flowing.
- Object Organization.

Uses of Sequence Diagram:

- To Model the Flow of Control by Time Sequence.
- To Model the Flow of Control by Structural Organizations.
- For Forward Engineering.
- For Reverse Engineering.

RESULT:

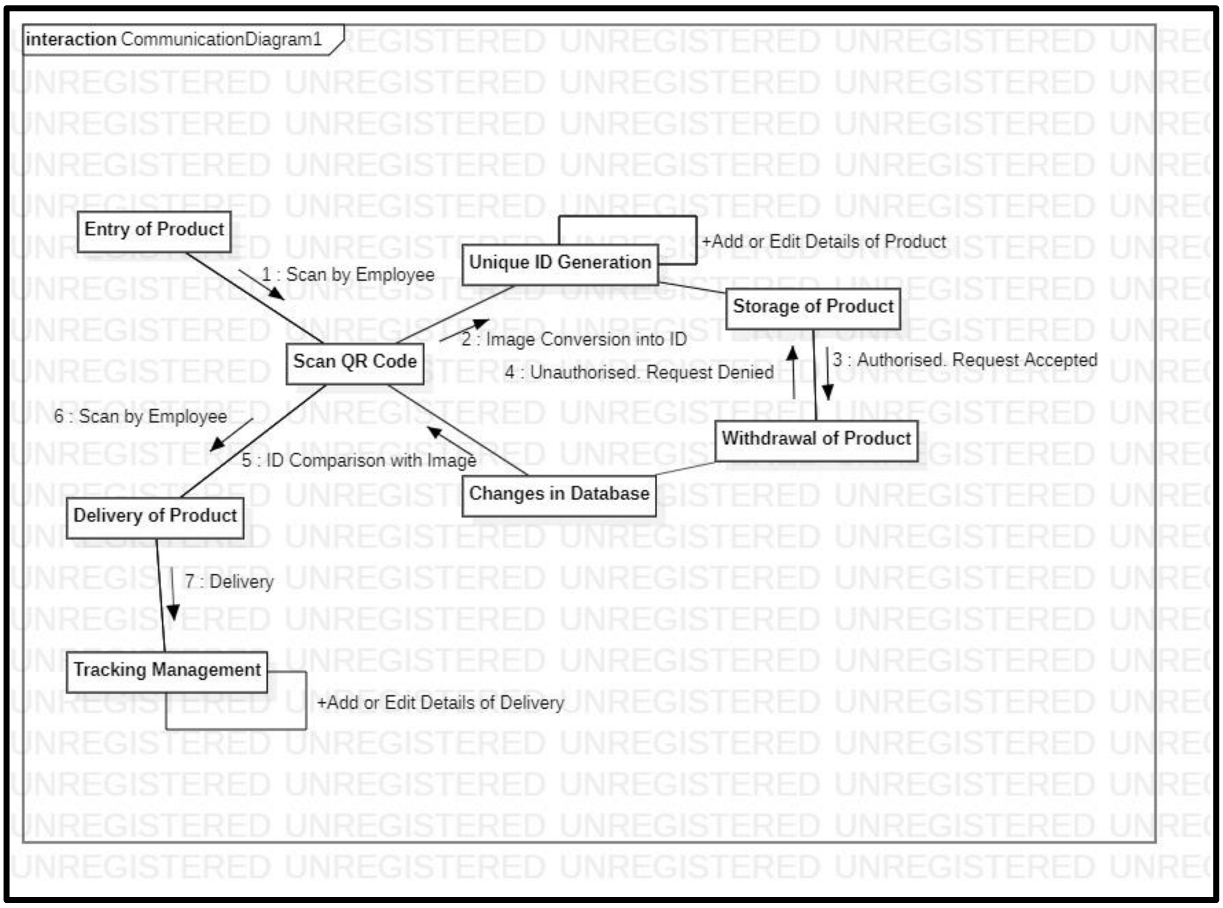
The Dataflow Diagram and the Sequence Diagram were constructed for the **CarGO** Project, in STAR UML.

COLLABORATION DIAGRAM & STATECHART DIAGRAM

AIM:

To generate the Collaboration Diagram and the Statechart Diagram for the **CarGO** Project in the STAR UML software.

Collaboration Diagram:


interaction CommunicationDiagram1
Entry of Product
1 : Scan by Employee
Unique ID Generation
+Add or Edit Details of Product
Storage of Product
Scan QR Code
2 : Image Conversion into ID
4 : Unauthorised. Request Denied
3 : Authorised. Request Accepted
6 : Scan by Employee
Withdrawal of Product
5 : ID Comparison with Image
Changes in Database
Delivery of Product
7 : Delivery
Tracking Management
+Add or Edit Details of Delivery

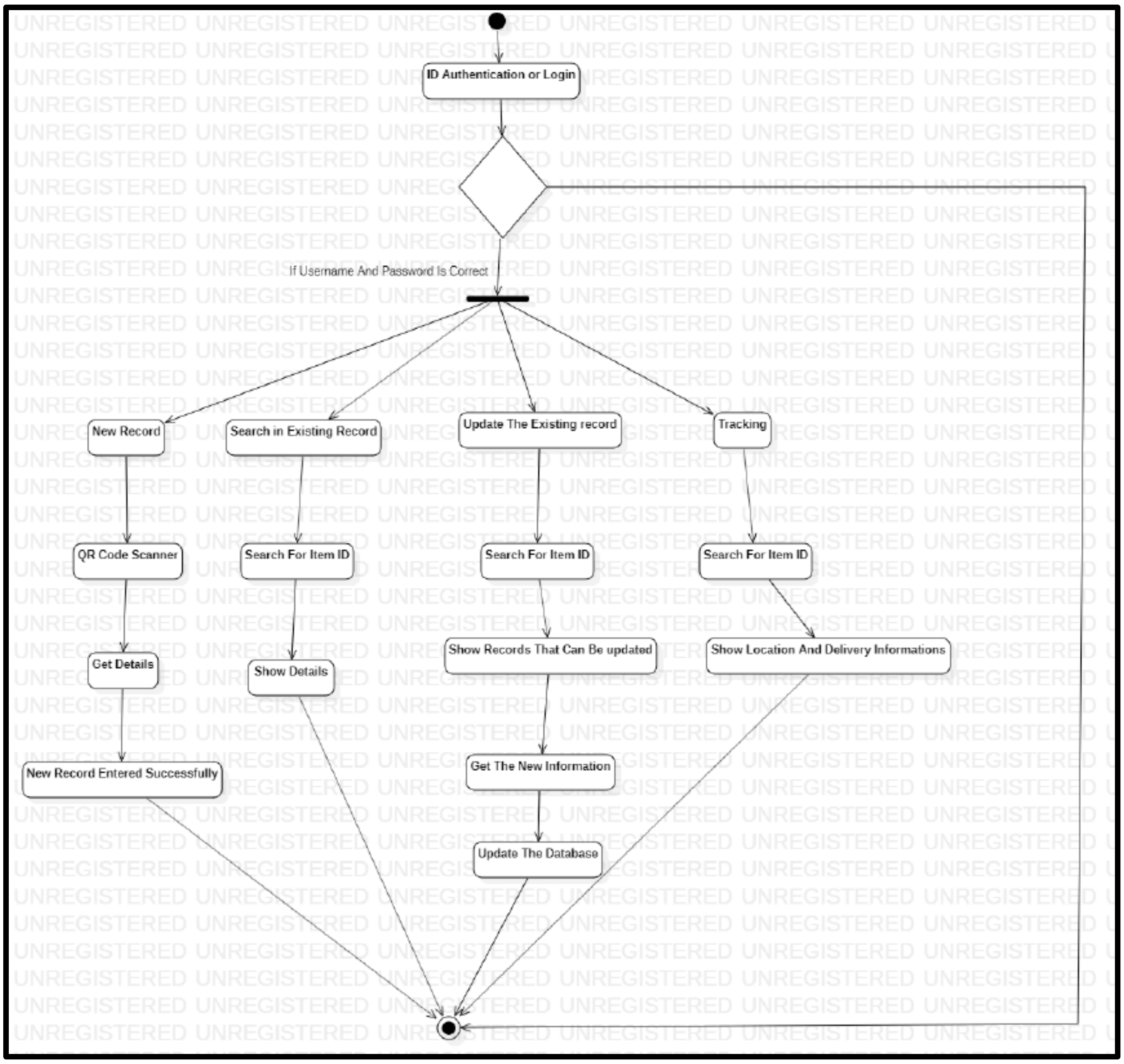
Features of Collaboration Diagram:

- A Collaboration is a collection of Named Objects and Actors with Links connecting them. They collaborate in performing some Task.
- A Collaboration defines a set of Participants and Relationships that are meaningful for a given set of purposes.

Uses of Collaboration Diagram:

- A Collaboration between Objects working together provides Emergent Desirable Functionalities in Object-Oriented Systems.
- Each Object (Responsibility) partially supports Emergent Functionalities.
- Objects are able to produce (Usable) High-Level Functionalities by working together.
- Objects collaborate by Communicating (Passing Messages) with one another in order to work together.

Statechart Diagram:


ID Authentication or Login
If Username And Password Is Correct
New Record
Search in Existing Record
Update The Existing record
Tracking
QR Code Scanner
Search For Item ID
Search For Item ID
Search For Item ID
Get Details
Show Details
Show Records That Can Be updated
Show Location And Delivery Informations
New Record Entered Successfully
Get The New Information
Update The Database

Features of Statechart Diagram:

- To Model the Dynamic Aspect of a System.
- To Model the Life Time of a Reactive System.

- To describe different States of an Object during its Lifetime.
- Define a State Machine to Model the States of an Object.

Uses of Statechart Diagram:

- To Model the Object States of a System.
- To Model the Reactive System. Reactive System consists of Reactive Objects.
- To Identify the Events responsible for State Changes.
- Forward and Reverse Engineering.

RESULT:

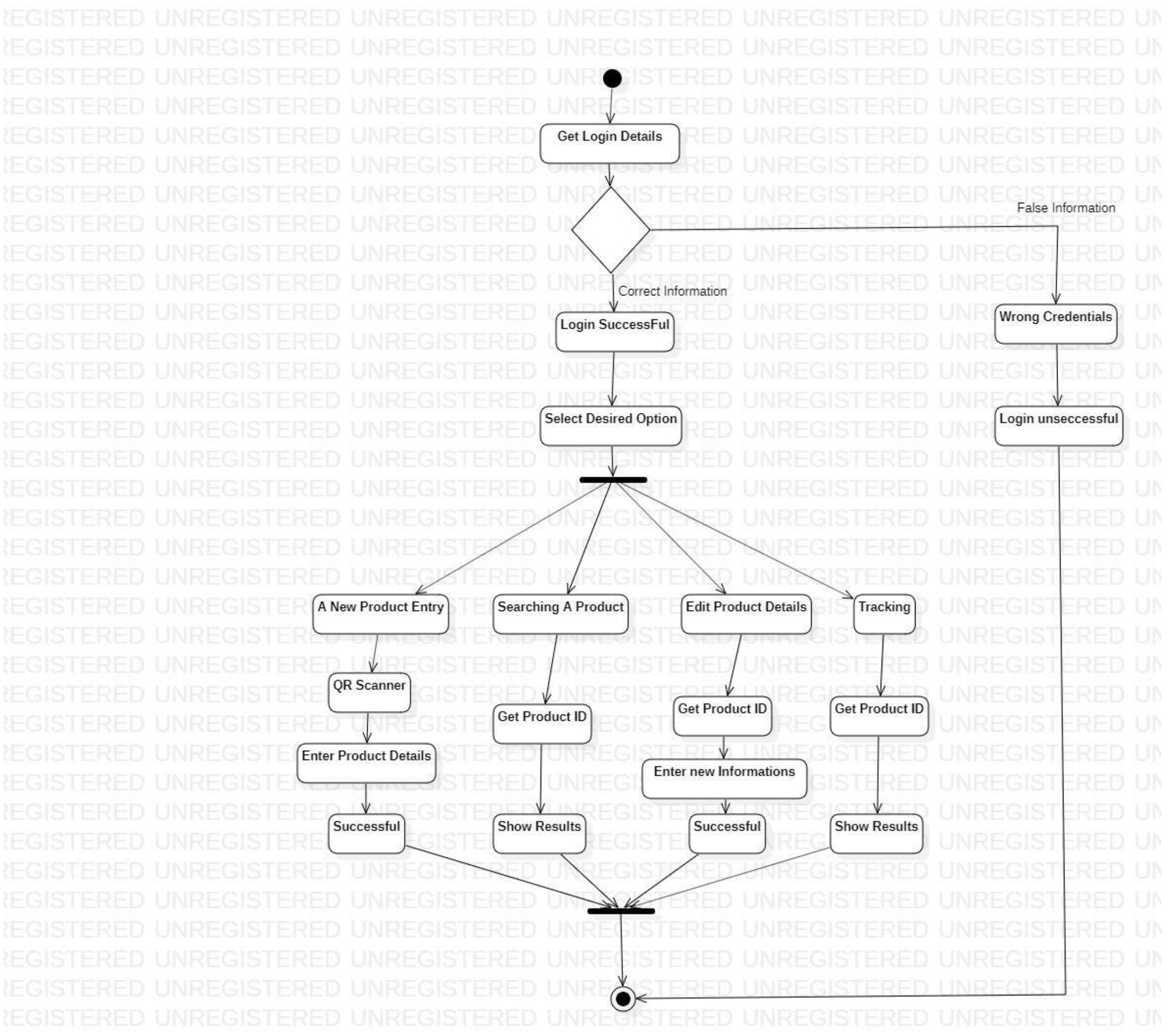
The Collaboration Diagram and the Statechart Diagram were constructed for the **CarGO** Project, in STAR UML.

ACTIVITY DIAGRAM

AIM:

To generate the Activity Diagram for the **CarGO** Project in the STAR UML software.

Activity Diagram:



Features of Activity Diagram:

- Draw the Activity Flow of a System.
- Describe the Sequence from one Activity to another.
- Describe the Parallel, Branched and Concurrent Flow of the System.

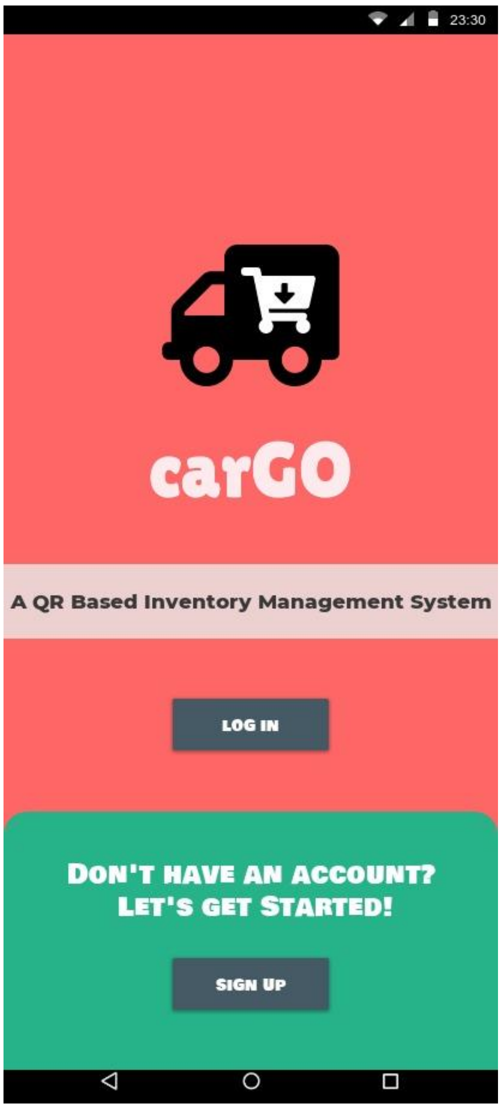
Uses of Activity Diagram:

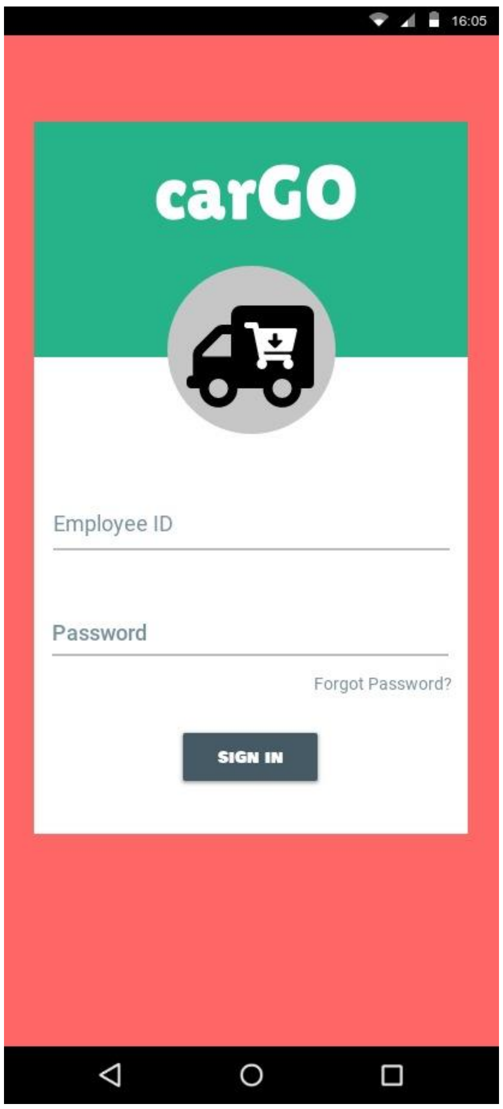
- Modelling Work Flow by using Activities.
- Modelling Business Requirements.
- High Level understanding of the System's Functionalities.
- Investigating Business Requirements at a later stage.

RESULT:

The Activity Diagram was constructed for the **CarGO** Project, in STAR UML.

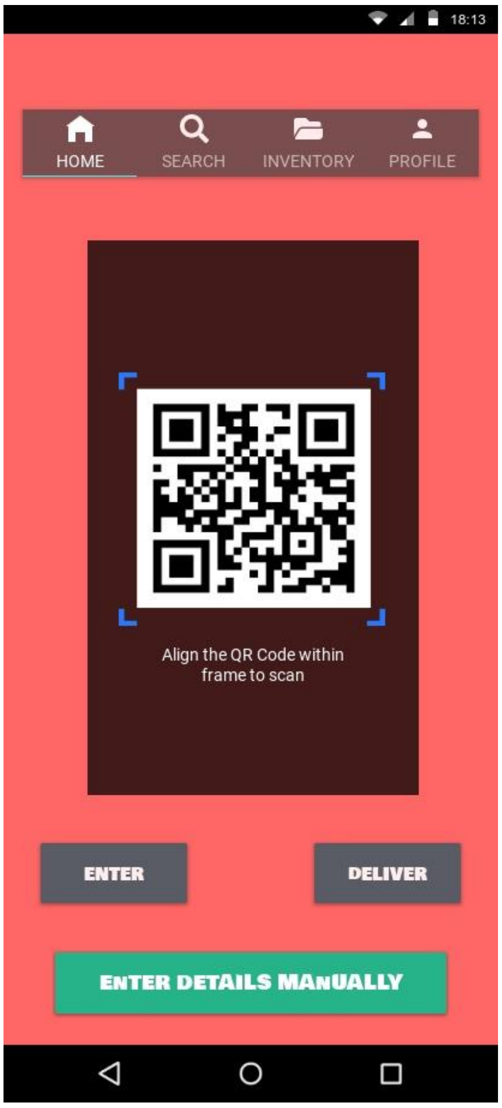
SOME SCREENSHOTS OF THE FRONT END AND THE CODE FOR OUR PROJECT:


23:30
carGO
A QR Based Inventory Management System
LOG IN
DON'T HAVE AN ACCOUNT? LET'S GET STARTED!
SIGN UP

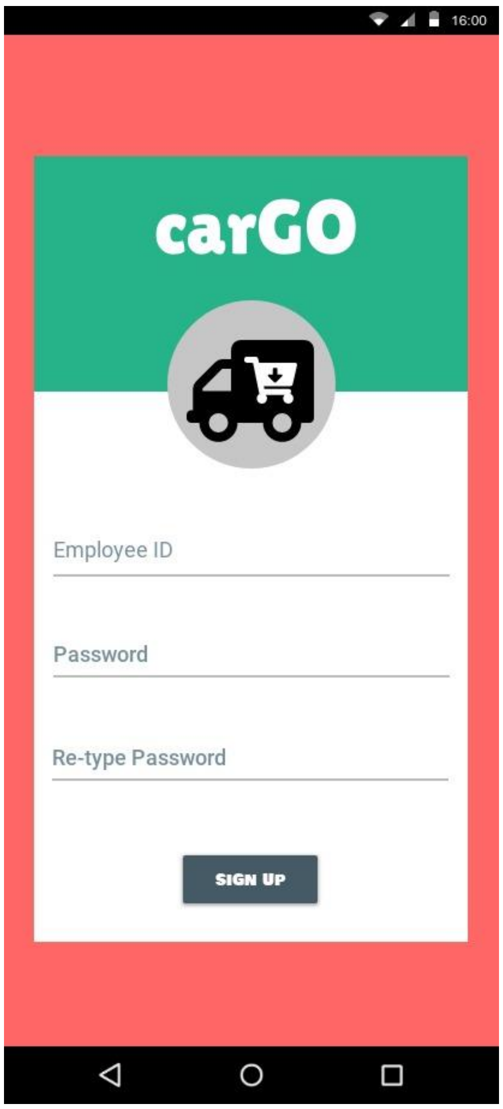


INRODUCTION PAGE

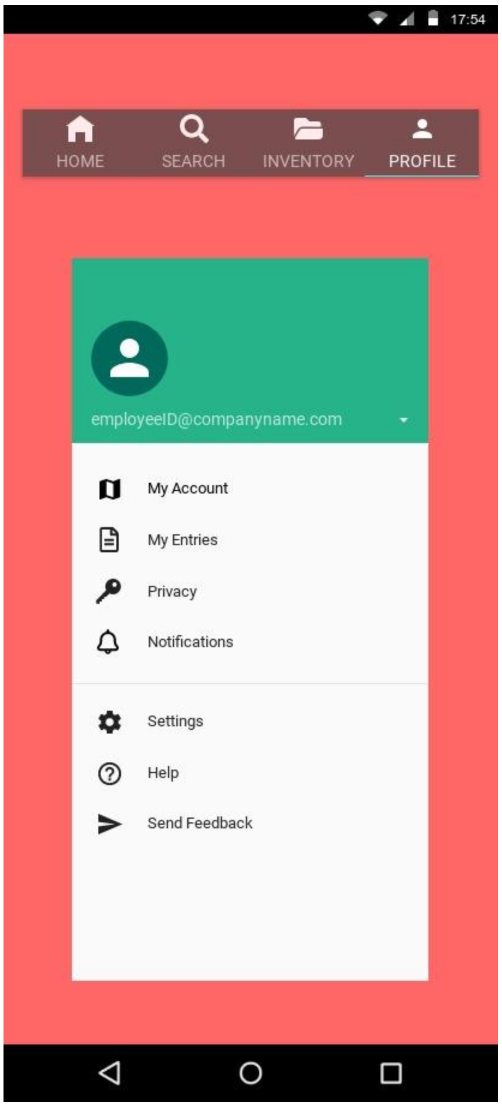
SIGN IN PAGE



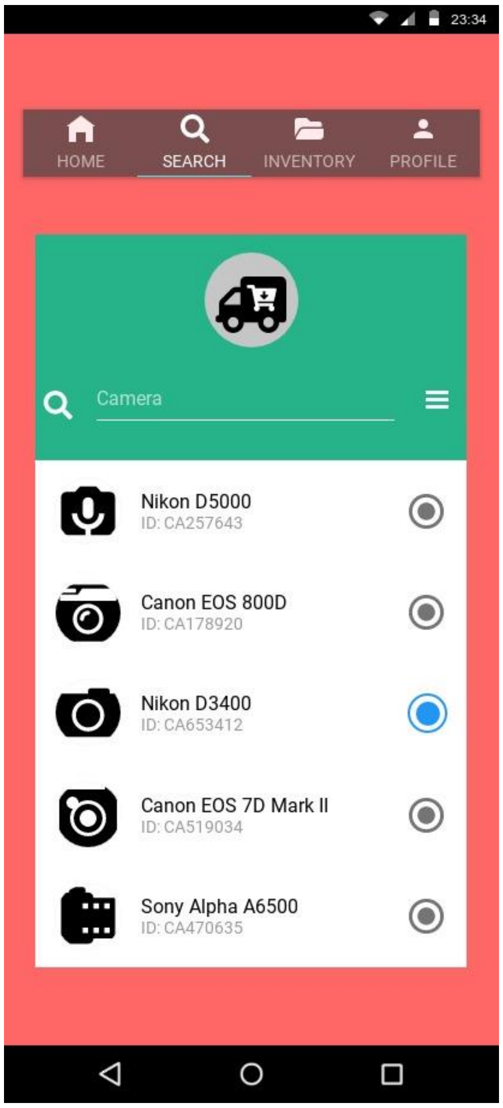
QR SCANNER



SIGN UP PAGE



OPTIONS PANE



Nikon D5000
ID: CA257643

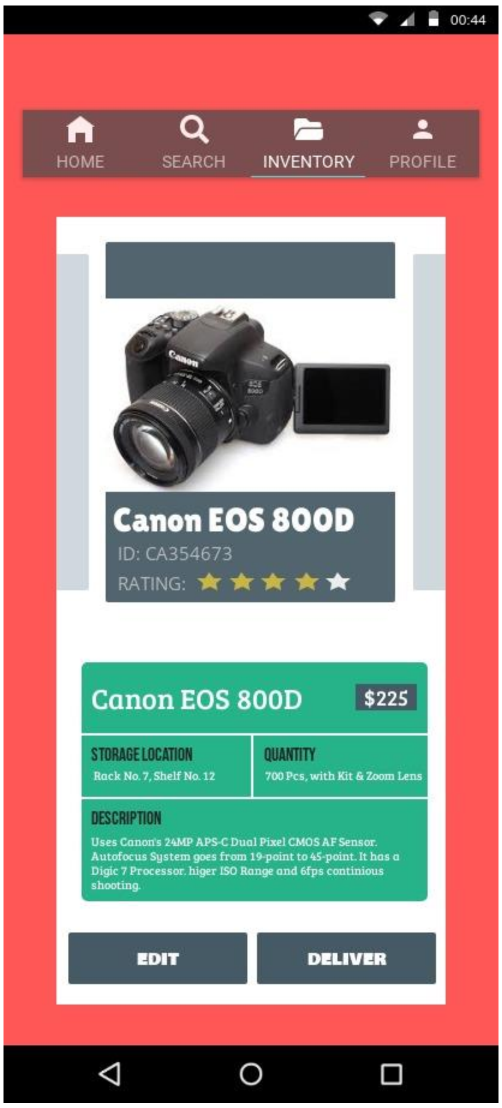
Canon EOS 800D
ID: CA178920

Nikon D3400 [✓]
ID: CA653412

Canon EOS 7D Mark II
ID: CA519034

Sony Alpha A6500
ID: CA470635

PRODUCTS WINDOW



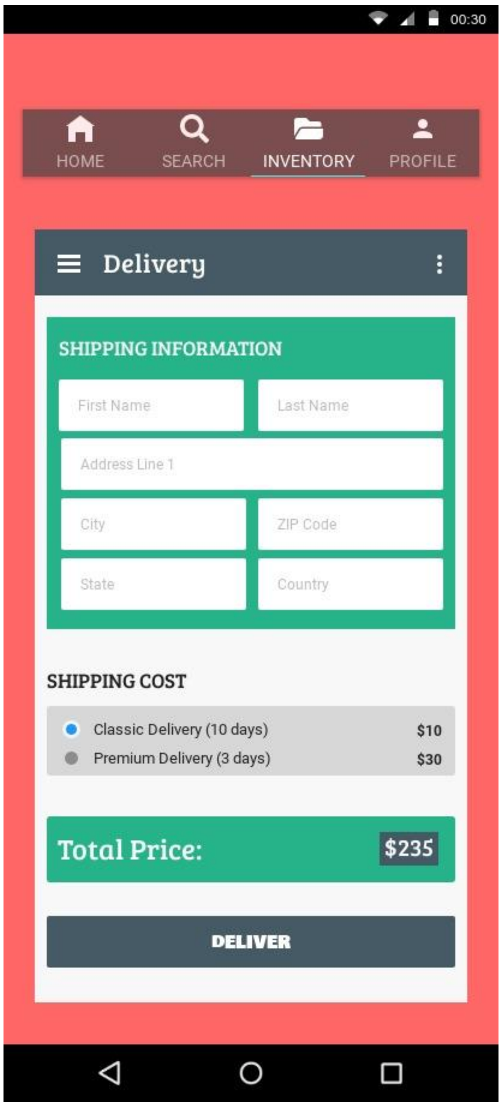
Canon EOS 800D
ID: CA354673
RATING: ★★★★☆

Canon EOS 800D $225

STORAGE LOCATION	QUANTITY
Rack No. 7, Shelf No. 12	700 Pcs, with Kit & Zoom Lens

DESCRIPTION
Uses Canon's 24MP APS-C Dual Pixel CMOS AF Sensor. Autofocus System goes from 19-point to 45-point. It has a Digic 7 Processor. higer ISO Range and 6fps continious shooting.

PRODUCT DETAILS



SHIPPING INFORMATION

SHIPPING COST

● Classic Delivery (10 days)	$10
● Premium Delivery (3 days)	$30

Total Price: $235

DELIVERY INFORMATION

ANDROID STUDIO CODE WRITTEN IN KOTLIN

1. Gradle Build code:

```
apply plugin:
'com.android.applicatio
n'
apply plugin: 'kotlin-android'



android {

    compileSdkVersion 25

    buildToolsVersion '25.0.1'



    defaultConfig {

        applicationId "com.kotlindroider.devaj"

        minSdkVersion 15

        targetSdkVersion 25

        versionCode 1

        versionName "1.0"

    }

    buildTypes {

        release {

            minifyEnabled false
            proguardFiles
getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'

        }

    }

}




dependencies {

   compile fileTree(dir: 'libs', include: ['*.jar'])

   compile 'com.android.support:appcompat-v7:25.0.1'

   compile 'com.android.support:design:25.0.1'

   compile "org.jetbrains.kotlin:kotlin-stdlib-
jdk7:$kotlin_version"

}

repositories {

   mavenCentral()

}
```

2. Android Manifest code:

```
<?xml version="1.0"
    encoding="utf-8"?>

<manifest
    xmlns:android="http://schemas.android.com/apk/r
    es/android"

   package="com.kotlindroider.devaj">
```

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme">
    <activity

android:name="com.kotlindroider.devaj.LoginActi
vity"
        android:label="@string/app_name"
        android:theme="@style/AppTheme.Dark">
        <intent-filter>
            <action
android:name="android.intent.action.MAIN" />


            <category
android:name="android.intent.category.LAUNCHER"
/>
        </intent-filter>
    </activity>
    <activity

android:name="com.kotlindroider.devaj.MainActiv
ity"
        android:theme="@style/AppTheme.Dark" />
    <activity
```

```
android:name="com.kotlindroider.devaj.SignupAct
ivity"
        android:theme="@style/AppTheme.Dark" />

</application>


</manifest>
```

3. **Login Activity Code:**

```
packag
e
com.ko
tlindr
oider.
devaj
```

```
import android.app.Activity

import android.app.ProgressDialog

import android.content.Intent

import android.os.Bundle

import android.support.v7.app.AppCompatActivity

import android.util.Log

import android.widget.Button

import android.widget.EditText

import android.widget.TextView
import android.widget.Toast



class LoginActivity : AppCompatActivity() {



   var _emailText: EditText? = null

   var _passwordText: EditText? = null

   var _loginButton: Button? = null

   var _signupLink: TextView? = null



   public override fun onCreate(savedInstanceState: Bundle?)
{
       super.onCreate(savedInstanceState)

setContentView(com.kotlindroider.devaj.R.layout.activity_log
in)



       _loginButton = findViewById(R.id.btn_login) as Button

       _signupLink = findViewById(R.id.link_signup) as
TextView

       _passwordText = findViewById(R.id.input_password) as
EditText

       _emailText = findViewById(R.id.input_email) as
EditText

       _loginButton!!.setOnClickListener { login() }
```

```
        _signupLink!!.setOnClickListener {

            // Start the Signup activity

            val intent = Intent(applicationContext,
SignupActivity::class.java)

            startActivityForResult(intent, REQUEST_SIGNUP)

            finish()

            //
overridePendingTransition(com.kotlindroider.devaj.R.anim.pus
h_left_in, com.kotlindroider.devaj.R.anim.push_left_out);

        }

    }



    fun login() {

        Log.d(TAG, "Login")



        if (!validate()) {

            onLoginFailed()

            return

        }



        _loginButton!!.isEnabled = false



        val progressDialog =
ProgressDialog(this@LoginActivity,
com.kotlindroider.devaj.R.style.AppTheme_Dark_Dialog)
        progressDialog.isIndeterminate = true

        progressDialog.setMessage("Login...")

        progressDialog.show()



        val email = _emailText!!.text.toString()

        val password = _passwordText!!.text.toString()



        // TODO: Implement your own authentication logic
here.



        android.os.Handler().postDelayed(

                {

                    // On complete call either onLoginSuccess
or onLoginFailed

                    onLoginSuccess()

                    // onLoginFailed();

                    progressDialog.dismiss()

                }, 3000)

    }
    override fun onActivityResult(requestCode: Int,
resultCode: Int, data: Intent) {
        if (requestCode == REQUEST_SIGNUP) {
            if (resultCode == Activity.RESULT_OK) {



                // TODO: Implement successful signup logic
here
                // By default we just finish the Activity and
log them in automatically
                this.finish()
            }
        }
    }


    override fun onBackPressed() {
        // Disable going back to the MainActivity
        moveTaskToBack(true)
    }


    fun onLoginSuccess() {
        _loginButton!!.isEnabled = true
//          finish()
        startActivity(Intent(this, MainActivity::class.java))
    }
    fun onLoginFailed() {

        Toast.makeText(baseContext, "Login failed",
Toast.LENGTH_LONG).show()



        _loginButton!!.isEnabled = true

    }



    fun validate(): Boolean {

        var valid = true



        val email = _emailText!!.text.toString()

        val password = _passwordText!!.text.toString()



        if (email.isEmpty() ||
!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches(
)) {

            _emailText!!.error = "enter a valid email
address"

            valid = false

        } else {

            _emailText!!.error = null

        }
        if (password.isEmpty() || password.length < 4 ||
password.length > 10) {
            _passwordText!!.error = "between 4 and 10
alphanumeric characters"
            valid = false
        } else {
            _passwordText!!.error = null
        }

        return valid
    }

    companion object {
        private val TAG = "LoginActivity"
        private val REQUEST_SIGNUP = 0
    }
}
```

4. **Main activity code:**

```
Package
com.kotlindroider
.devaj
```

```
import android.os.Bundle

import android.support.v7.app.AppCompatActivity




class MainActivity : AppCompatActivity() {



   override fun onCreate(savedInstanceState:
Bundle?) {
       super.onCreate(savedInstanceState)


setContentView(com.kotlindroider.devaj.R.layout.a
ctivity_main)



   }



}
```

5. **SignUp Activity Code:**

```
package
com.kot
lindroi
```

der.dev
aj

```
import android.app.ProgressDialog

import android.content.Intent

import android.os.Bundle

import android.support.v7.app.AppCompatActivity

import android.util.Log

import android.widget.Button

import android.widget.EditText

import android.widget.TextView

import android.widget.Toast



class SignupActivity : AppCompatActivity() {



   var _nameText: EditText? = null

   var _addressText: EditText? = null

   var _emailText: EditText? = null

   var _mobileText: EditText? = null

   var _passwordText: EditText? = null

   var _reEnterPasswordText: EditText? = null

   var _signupButton: Button? = null
   var _loginLink: TextView? = null



   public override fun onCreate(savedInstanceState:
Bundle?) {

       super.onCreate(savedInstanceState)


setContentView(com.kotlindroider.devaj.R.layout.activity_si
gnup)



       _nameText = findViewById(R.id.input_name) as
EditText

       _addressText = findViewById(R.id.input_address) as
EditText

       _emailText = findViewById(R.id.input_email) as
EditText

       _mobileText = findViewById(R.id.input_mobile) as
EditText

       _passwordText = findViewById(R.id.input_password) as
EditText

       _reEnterPasswordText =
findViewById(R.id.input_reEnterPassword) as EditText



       _signupButton = findViewById(R.id.btn_signup) as
Button

       _loginLink = findViewById(R.id.link_login) as
TextView



       _signupButton!!.setOnClickListener { signup() }
```

```
        _loginLink!!.setOnClickListener {
            // Finish the registration screen and return to
the Login activity
            val intent = Intent(applicationContext,
LoginActivity::class.java)
            startActivity(intent)
            finish()

overridePendingTransition(com.kotlindroider.devaj.R.anim.pu
sh_left_in, com.kotlindroider.devaj.R.anim.push_left_out)
        }
    }


    fun signup() {
        Log.d(TAG, "Signup")


        if (!validate()) {
            onSignupFailed()
            return
        }


        _signupButton!!.isEnabled = false
        val progressDialog =
ProgressDialog(this@SignupActivity,

com.kotlindroider.devaj.R.style.AppTheme_Dark_Dialog)
        progressDialog.isIndeterminate = true

        progressDialog.setMessage("Creating Account...")

        progressDialog.show()



        val name = _nameText!!.text.toString()

        val address = _addressText!!.text.toString()

        val email = _emailText!!.text.toString()

        val mobile = _mobileText!!.text.toString()

        val password = _passwordText!!.text.toString()

        val reEnterPassword =
_reEnterPasswordText!!.text.toString()



        // TODO: Implement your own signup logic here.



        android.os.Handler().postDelayed(

                {

                    // On complete call either
onSignupSuccess or onSignupFailed
                    // depending on success

                    onSignupSuccess()

                    // onSignupFailed();
                    progressDialog.dismiss()

                }, 3000)

    }




    fun onSignupSuccess() {

        _signupButton!!.isEnabled = true

//          setResult(Activity.RESULT_OK, null)

//          finish()

        startActivity(Intent(this,
MainActivity::class.java))

        finish()

    }



    fun onSignupFailed() {

        Toast.makeText(baseContext, "Login failed",
Toast.LENGTH_LONG).show()



        _signupButton!!.isEnabled = true

    }



    fun validate(): Boolean {
        var valid = true



        val name = _nameText!!.text.toString()

        val address = _addressText!!.text.toString()

        val email = _emailText!!.text.toString()

        val mobile = _mobileText!!.text.toString()

        val password = _passwordText!!.text.toString()

        val reEnterPassword =
_reEnterPasswordText!!.text.toString()



        if (name.isEmpty() || name.length < 3) {

            _nameText!!.error = "at least 3 characters"

            valid = false

        } else {

            _nameText!!.error = null

        }



        if (address.isEmpty()) {

            _addressText!!.error = "Enter Valid Address"

            valid = false

        } else {

            _addressText!!.error = null
        }

        if (email.isEmpty() ||
!android.util.Patterns.EMAIL_ADDRESS.matcher(email).matches
()) {
            _emailText!!.error = "enter a valid email
address"
            valid = false
        } else {
            _emailText!!.error = null
        }

        if (mobile.isEmpty() || mobile.length != 10) {
            _mobileText!!.error = "Enter Valid Mobile
Number"
            valid = false
        } else {
            _mobileText!!.error = null
        }

        if (password.isEmpty() || password.length < 4 ||
password.length > 10) {
            _passwordText!!.error = "between 4 and 10
alphanumeric characters"
```

```
            valid = false

        } else {

            _passwordText!!.error = null

        }




        if (reEnterPassword.isEmpty() ||
reEnterPassword.length < 4 || reEnterPassword.length > 10
|| reEnterPassword != password) {

            _reEnterPasswordText!!.error = "Password Do not
match"

            valid = false

        } else {

            _reEnterPasswordText!!.error = null

        }




        return valid

    }




    companion object {

        private val TAG = "SignupActivity"

    }

}
```