



hellomarkets

Graduation Project, Part-II (SWE 497)
Software Engineering Department
CCIS, KSU

Project Advisor:
Dr. Achraf Gazdar

Submitted by
Rakan Saleh Alnasser, 436102329
Saad Aldwsry, 436100544
Osamah Ali Alghamdi, 436100794
Mohammed Abdulrahman altheeb, 436102409

Date submitted
10/12/2019

ABSTRACT

This report describes the development process of a supermarket offers application aiming to help customers accessing easily the best offers. Customers usually struggle to find the optimal offer between all the available supermarkets and our app aims to get rid of that as everything will be clear for the customer to see. With our app, customers will no longer be dependent on old fashioned supermarket magazines in their search for offers since our App will be more efficient in their search.

Table of Contents

1. INTRODUCTION	1
2. ANALYSIS AND DESIGN	2
2.1 UPDATED FUNCTIONAL REQUIREMENTS	2
2.2 UPDATED NON-FUNCTIONAL REQUIREMENTS	4
2.3 UPDATED USE CASE REQUIREMENTS	5
2.4 DESIGN CLASSES	8
2.5 SEQUENCE DIAGRAM	9
2.6 SOFTWARE ARCHITECTURE	13
3. PROTOTYPE DESCRIPTION	13
3.1 IMPLEMENTATION PLATFORM	13
3.2 MAPPING BETWEEN REQUIREMENTS AND IMPLEMENTED FUNCTIONS	14
3.3 IMPLEMENTATION DETAILS	17
3.4 ACTUAL DATABASE SCHEMA	22
4. TESTING	23
4.1 EXPECTED TEST SCENARIOS	23
4.2 UNIT TEST	23
4.3 FUNCTIONAL TEST	25
4.3 USABILITY TEST	30
5. DEPLOYMENT OF THE SYSTEM	31
6. LIMITATION OF THE SYSTEM	31
7. CONCLUSION AND FUTURE WORK	32
8. REFERENCE	32

List of Figures

Figure 1. Administrator use cases	5
Figure 2. Seller use cases	6
Figure 3. Customer use cases	7
Figure 4. Design Class.....	8
Figure 5. Ban customer.....	9
Figure 6. Ban seller	10
Figure 7. View Statistics	11
Figure 8. Notify Changed offer price	12
Figure 9. Three-Tier-architecture	13
Figure 10. Add an offer	18
Figure 11. View an offer	20
Figure 12. View categories	21
Figure 13. Database schema	22
Figure 14. Selenium testcase 1	24
Figure 16. Selenium testcase 2	24
Figure 16. Selenium testcase 3	25
Figure 16. Test scenario 1.....	28
Figure 14. Test scenario 2.....	29
Figure 15. Test scenario 3.....	30
Figure 16. Expected Deployment.....	31

List of Tables

Table 1. Admin's Requirements	14
Table 2. Seller's Requirements	15
Table 3. Customer's Requirements	16
Table 4. Unit test functionl	23
Table 5. Scenario testing.....	25
Table 6. Tested function	26
Table 7. Equivalence classes testing	26
Table 8. Valid equivalence classes	27
Table 9. Scenario path for testcase	28
Table 10. Scenario path for testcase	29

1. Introduction

In our increasingly technological age, there has been a decrease in customers going through supermarket paper magazines while shopping; and since the regular customer doesn't shop at one supermarket there has been an abundance of Supermarket apps on his phone.

Our app addresses this issue and resolves it as it will contain all offers from a various local supermarket. The users of our app will have more convenience than those who are old fashioned and still browse magazines while shopping, as our app will include a variety of features that will ease the search for the customers.

The emphasis of this project is to develop an all embodying supermarket app that eases customers search. Customers will no longer rely on other applications when they use our app, as our will include more features and an easier interface to browse. Customers will be notified of new and exciting offers as soon as they release, so they do not miss it.

For our project to succeed customer satisfaction needs to be high and maintained, as they are our target users. We will do so by making sure the user interface is easy to use and user friendly for people of all ages. Adults and Elderly alike will find it convenient to use and will find no difficulty whatsoever.

The rest of the report is organized as follows. The requirement phase is described through sections 2. Starting with 2.1, we identified the functional requirements and in 2.2, we identified the non-functional requirements of the system. In section 2.3, we did the use cases for the requirements we identified in section 2.

The design phase is detailed throughout sections 2, beginning with section 2.4, which is the design class, and ending with section 2.6, which is the expected Software Architecture.

2. Analysis and Design

2.1 Updated Functional Requirements

2.1.1 Administrator requirements

1. The system shall allow admin to login.
2. The system shall allow admin to logout.
3. The system shall allow admin to view message of contact customer service come from seller.
4. The system shall allow admin to view message of contact customer service come from customer.
5. The system shall allow admin to manage admin's accounts.
 - 5.1 The system shall allow admin to view admin's account information except password.
 - 5.2 The system shall allow admin to edit admin's account information.
 - 5.3 The system shall allow admin to add admin account.
6. The system shall allow admin to manage sellers' accounts.
 - 6.1 The system shall allow admin to create sellers' accounts.
 - 6.2 The system shall allow admin to view sellers' accounts information except passwords.
 - 6.3 The system shall allow admin to delete sellers' accounts.
 - 6.4 The system shall allow admin to edit sellers' accounts.
 - 6.5 The system shall allow admin to ban seller account.
 - 6.6 The system shall allow admin to unban seller account.
7. The system shall allow admin to manage customers' accounts.
 - 7.1 The system shall allow admin to view customers' accounts information except password.
 - 7.2 The system shall allow admin to delete customers' accounts.
 - 7.3 The system shall allow admin to edit customers' accounts.
 - 7.4 The system shall allow admin to ban customer accounts.
 - 7.5 The system shall allow admin to unban customer accounts.
8. The system shall allow admin to manage offer.
 - 8.1 The system shall allow admin to view offers.
 - 8.2 The system shall allow admin to delete an offer.
 - 8.3 The system shall allow admin to edit an offer.
9. The system shall allow admin to manage categories.
 - 9.1 The system shall allow admin to create category.
 - 9.2 The system shall allow admin to view category.
10. The system shall allow admin to view statistic.
 - 10.1 The system shall allow admin to view numbers of customers.
 - 10.2 The system shall allow admin to view numbers of sellers.
 - 10.3 The system shall allow admin to view numbers of offers.
 - 10.4 The system shall allow admin to view the best offer.
 - 10.5 The system shall allow admin to view the best seller.

2.1.2 Seller requirements

1. The system shall allow seller to login.
2. The system shall allow seller to logout.
3. The system shall allow seller to manage offer.
 - 3.1. The system shall allow seller to add an offer.
 - 3.2. The system shall allow seller to edit an offer.
 - 3.3. The system shall allow seller to delete an offer.
 - 3.4. The system shall allow seller to view their offers.
4. The system shall allow seller to contact customer service.
5. The system shall allow seller to view statistic.
 - 5.1. The system shall allow seller to view numbers of customers.
 - 5.2. The system shall allow seller to view numbers of sellers.
 - 5.3. The system shall allow seller to view numbers of offers.
 - 5.4. The system shall allow seller to view the best offer.
 - 5.5. The system shall allow seller to view the best seller.

2.1.3 Customer requirements

1. The system shall allow customer to signup.
2. The system shall allow customer to login.
3. The system shall allow customer to logout.
4. The system shall allow customer to manage account.
 - 4.1 The system shall allow customer to edit account.
 - 4.2 The system shall allow customer to delete account.
5. The system shall allow customer to view categories.
6. The system shall allow customer to view offers.
7. The system shall allow customer to search an offer.
8. The system shall allow customer to view an offer.
9. The system shall allow customer to view a seller information.
10. The system shall send notification for new price of specific offer saved by customer when seller changes price.
11. The system shall send notification for a specific search keyword saved by customer when seller add an offer.
12. The system shall allow customer to rate.
 - 12.1 The system shall allow customer to rate offer including comment.
 - 12.2 The system shall allow customer to rate seller.
13. The system shall allow customer to view rate.
 - 13.1 The system shall allow customer to view offers' rate.
 - 13.2 The system shall allow customer to view sellers' rate.
14. The system shall allow customer to save favorite offer.
15. The system shall allow customer to view favorite offer
16. The system shall allow customer to view statistic.
 - 16.1 The system shall allow customer to view the best offer.
 - 16.2 The system shall allow customer to view the best seller.
17. The system shall allow customer to be redirected to offers' page on sellers' website.
18. The system shall view customer any similar offers.
19. The system shall allow customer to contact customer service.

2.2 Updated Non-Functional Requirements

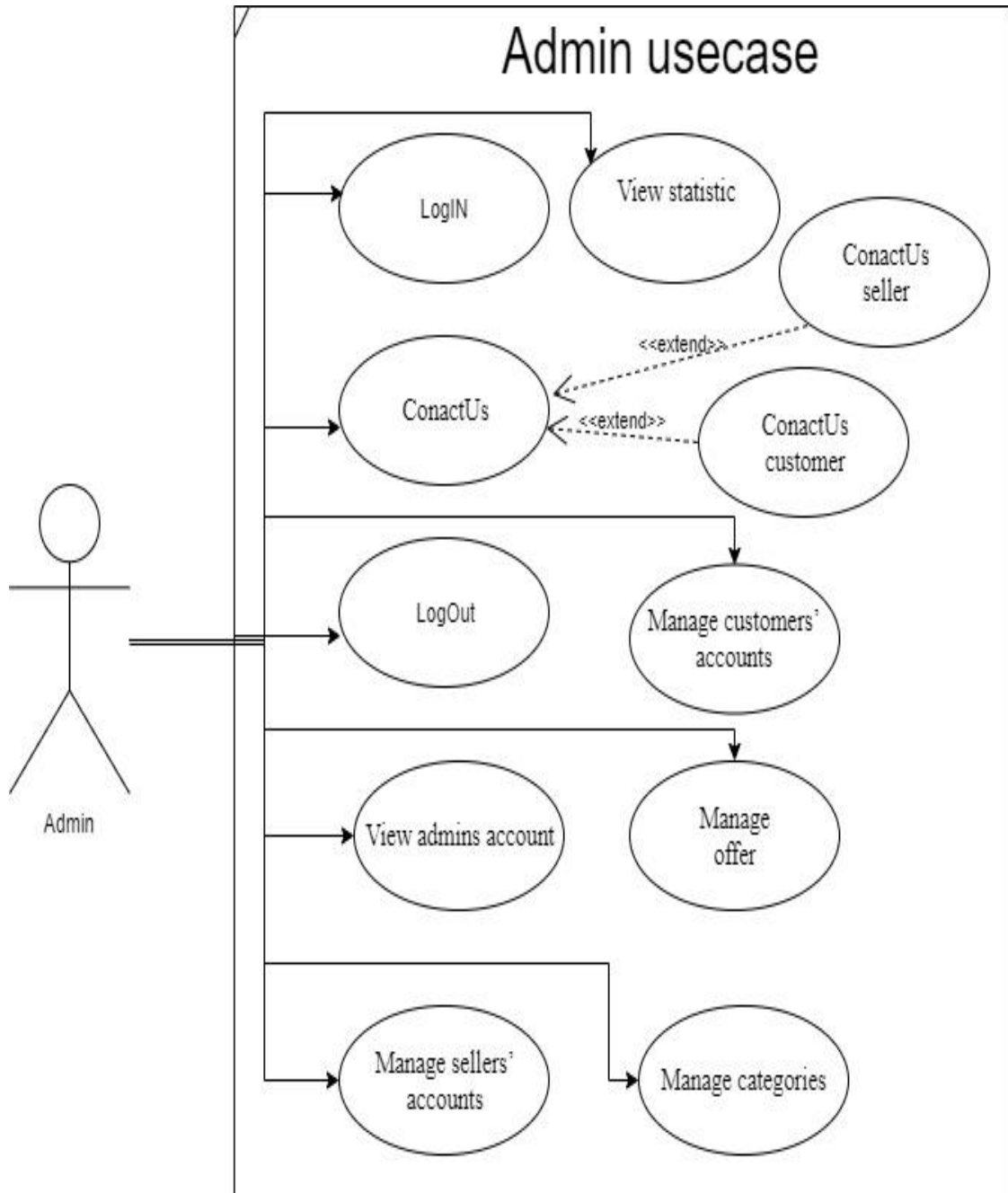
1. Reliability: The system shall be recovered and back upped within half an hour or less.
2. Maintainability: The system software documentation shall be fully understood in less than a week.
3. Usability: The seller shall be able to place an offer within minute.
4. Availability: The system shall be available 95% of the time.
5. Correctness: The system will support English language.
6. Design constrain: The developed system should run under flutter and php laravel

2.3 Updated Use Case Requirements

In this section we divided the system into 3 main actors. For each actor we identified all the possible use cases and their inclusions and extensions.

- Admin usecase

Figure 1, the administrator holds more privileges than customers and sellers.



- Figure 1. Administrator use cases

- Seller use case

Figure 2, the Seller does not have the privilege of creating an account as it can only be created by the Administrator.

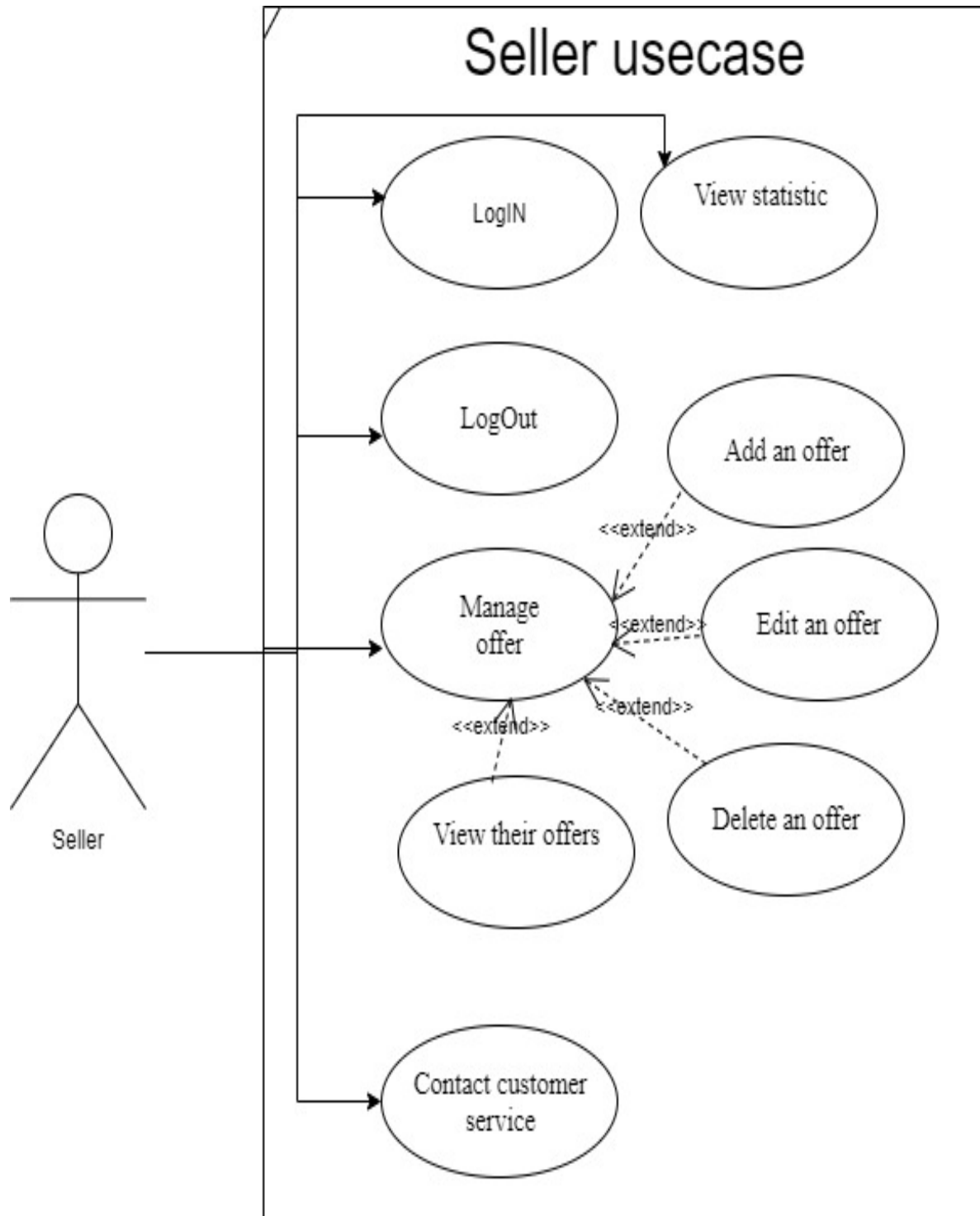


Figure 2. Seller use cases

- Customer use cases

Figure 3, the customer can create an account and has more use cases than the Seller and Administrator.

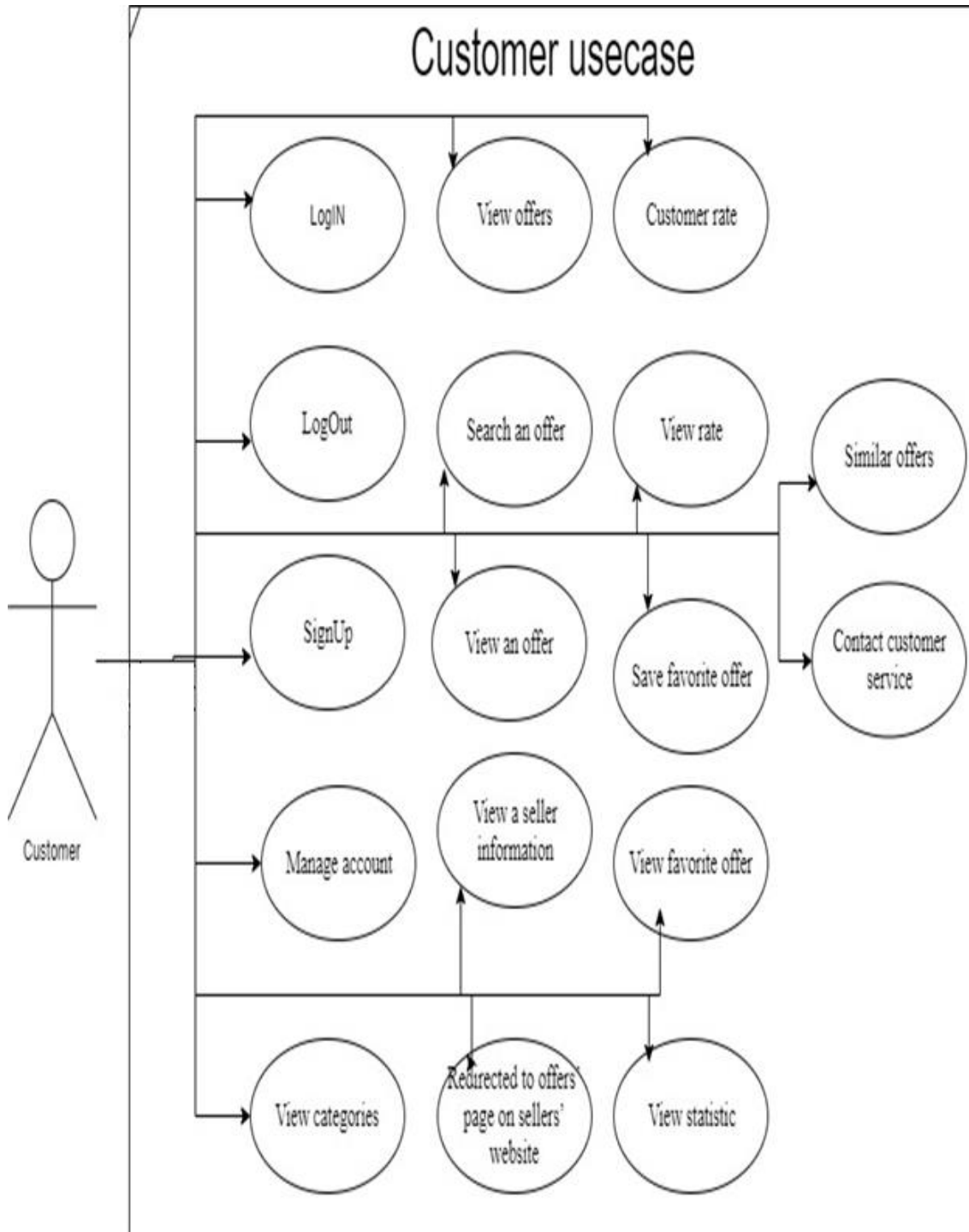


Figure 3. Customer use cases

2.4 Design Class

Figure 4, here we identify all the boundary class, control class and entity class for all the usecases.

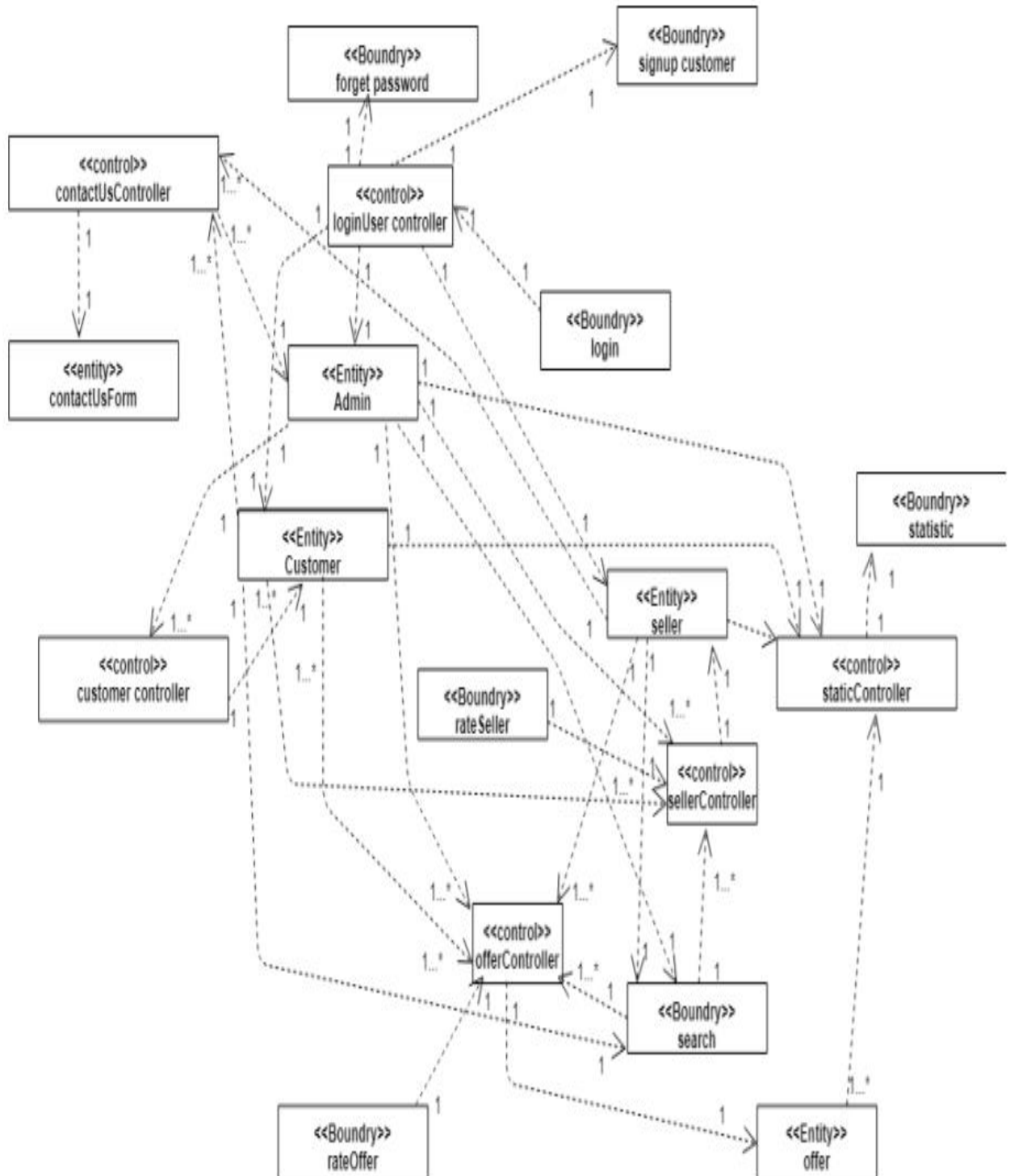


Figure 4. Design Class

2.5 Sequence Diagram

2.5.1 Ban customer

Figure 5, we showcase the admins' ban customer interaction. Admin will first need to view customers in order to select whom to ban. After selection is done, the admin will be able to ban customer.

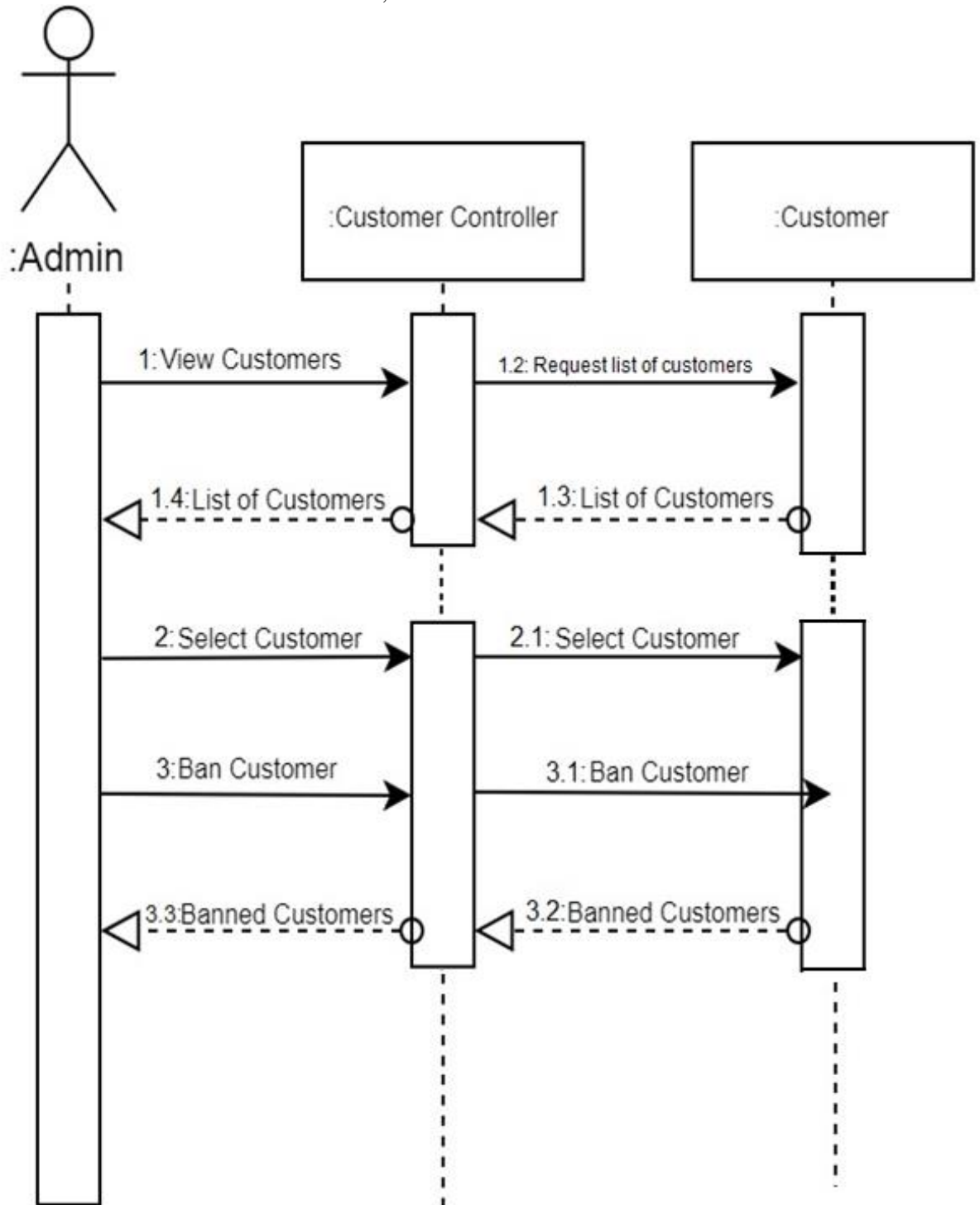


Figure 5. Ban customer

2.5.2 Ban seller

Figure 6, we showcase the admins' ban customer interaction. Admin will first need to view sellers in order to select whom to ban. After selection is done, the admin will be able to ban seller.

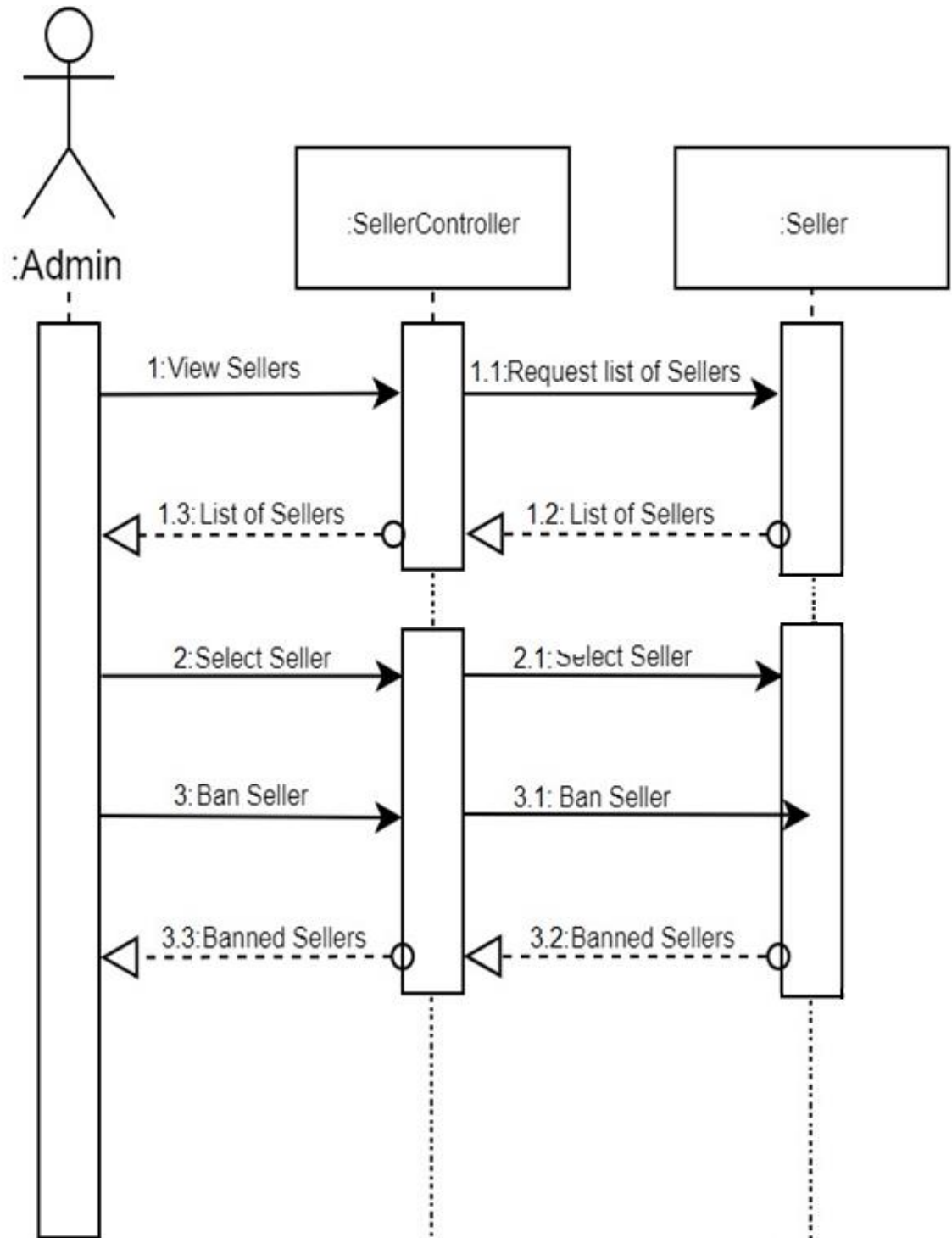


Figure 6. Ban seller

2.5.3 View Statistics

Figure 7, we showcase the view statistics interaction. It is simply done by clicking view statistics and then a list of statistics will be returned to whoever requested.

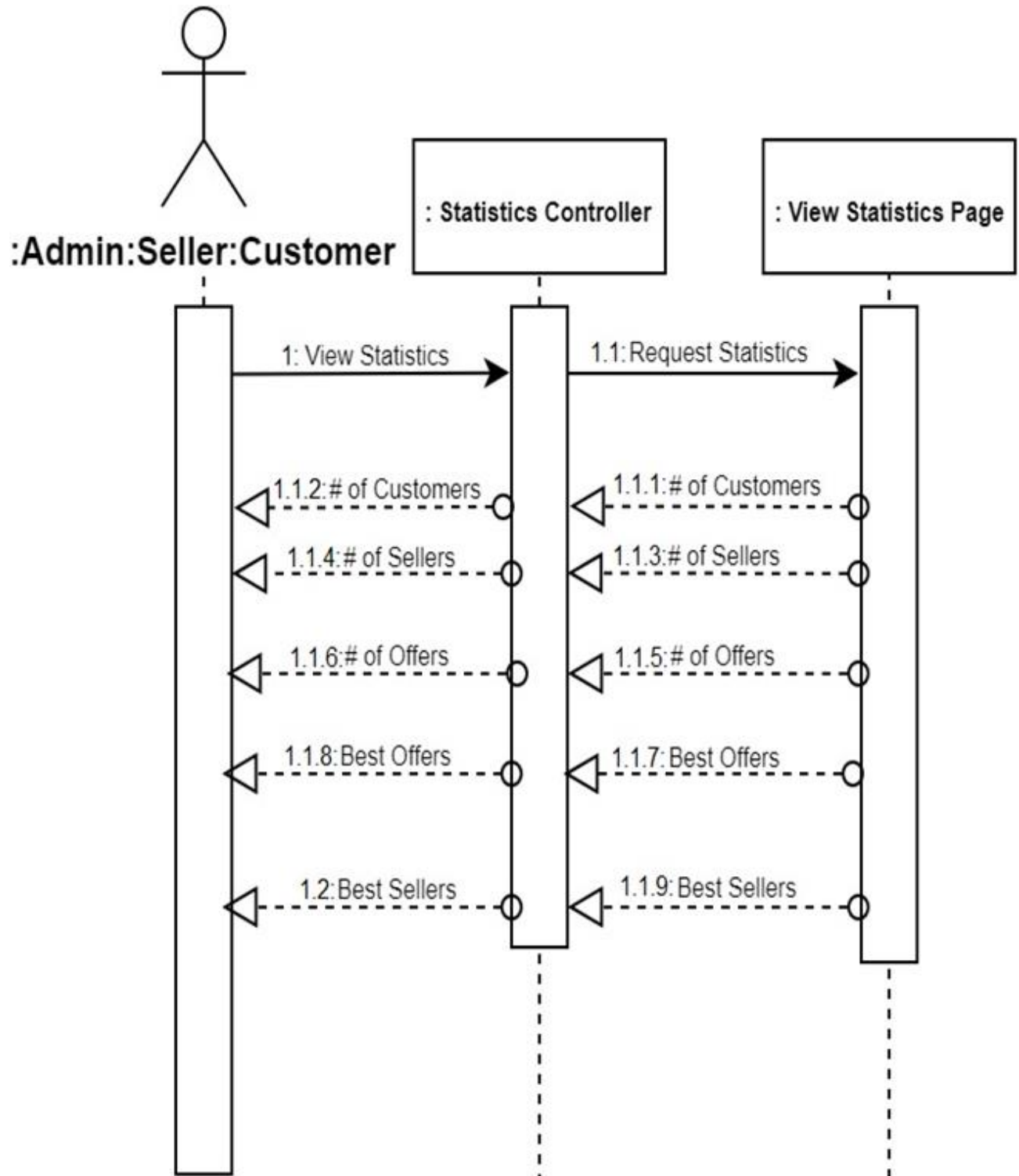


Figure 7. View Statistics

2.5.4 Notify Changed offer price

Figure 8, we showcase the customers' notify changed price interaction. Customer will first need to view offers in order to save the offers he would like to have an eye on, so after a seller changes price for that offer the customer will be notified.

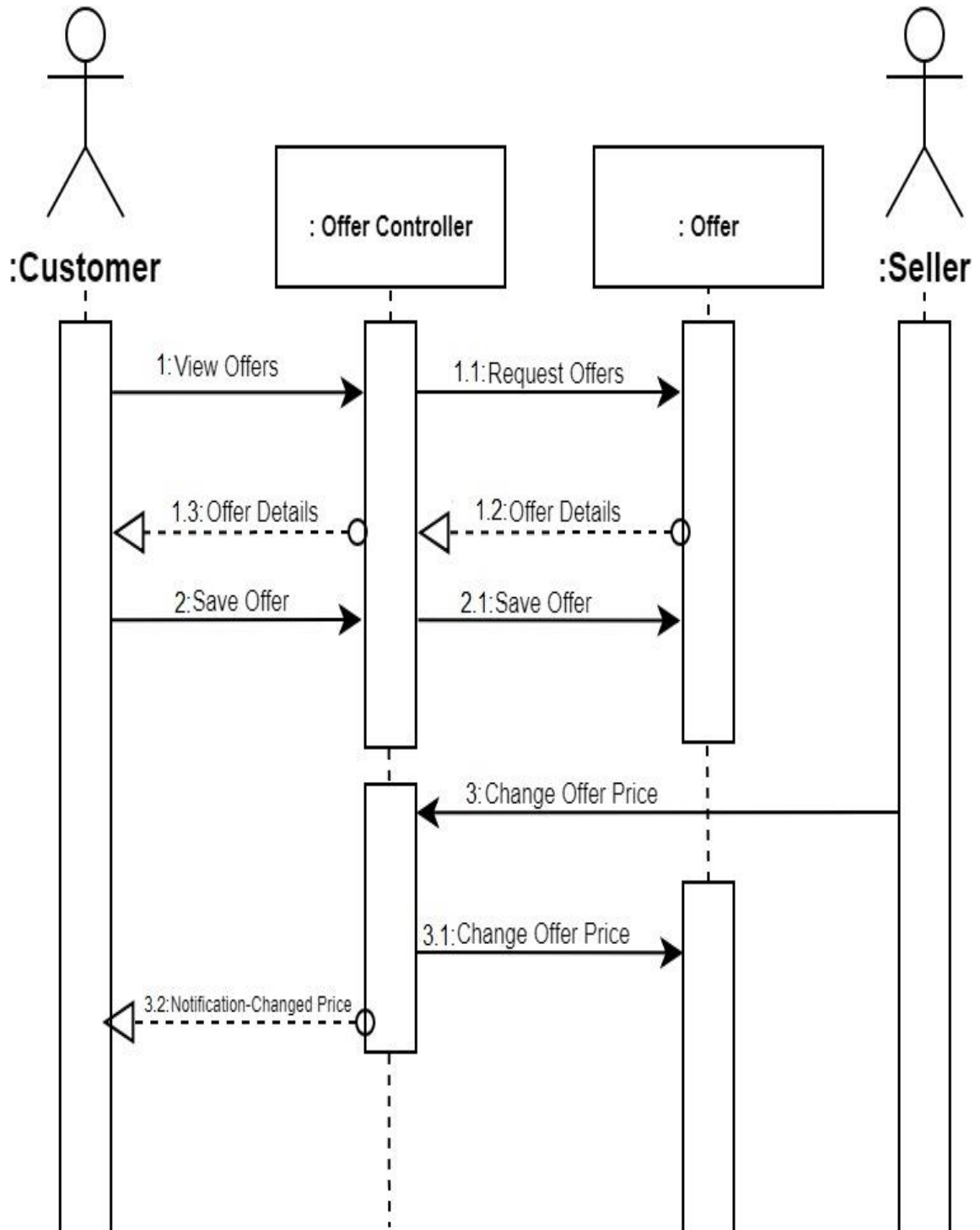


Figure 8. Notify Changed offer price

2.6 Software Architecture

In this section, we describe the system architecture of our application and how this architecture will affect the different qualities of the application. Our application is built with one architecture: 3 tier architecture.

Figure 9, shows three-tier-architecture which divides the system into 3 main components.

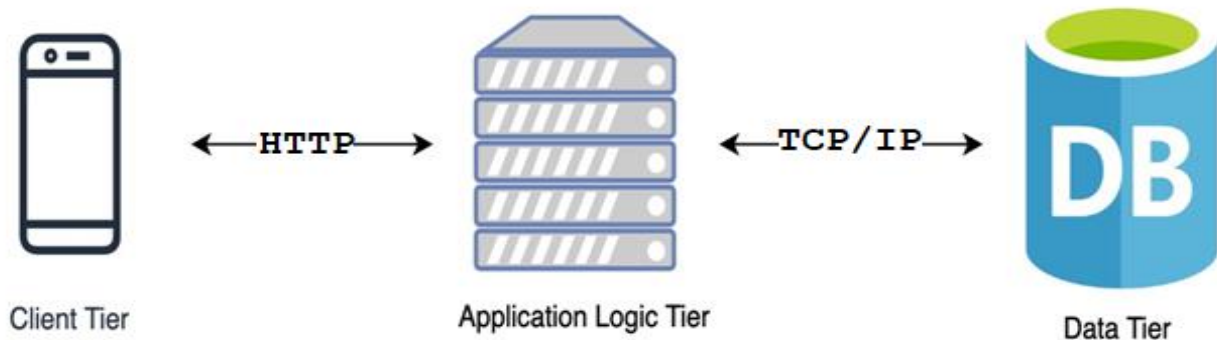


Figure 9. Three-Tier-architecture

- 1- User interface (presentation).
- 2- Functional process logic ("business rules").
- 3- Computer data storage and data access.

They are developed and maintained as independent modules, most often on separate platforms.

We are using the three-tier-architecture because it fits perfectly to our application, also the system will not be affected by the increase in the number of users, the three-tier-architecture will increase the performance because each component will stand alone, also it has a high degree of flexibility platform and configuration, it decreases the reusability of the system components, it will improve the data integrity, it will also improve the security (Client has no direct access to database).

3. Prototype Description

We developed the login and sign up functionality of the system for customers. Sellers however can only login. Once the customer is logged on a list of offers are displayed for him. The customer can search for products with the search bar and the search is based off keywords. The customer can also view offers and the seller has the ability of deleting and editing the offers.

3.1 Implementation Platform

- | | | |
|-------------------------------|----------------------|------------------------|
| 1. Laptop. | 2.Windows 10. | 3.Laravel for backend. |
| 4.Visual Studio as an editor. | 5.Xampp for hosting. | 6.Android Studio. |
| 7.Flutter framework. | 8.Simulator. | |

3.2 Mapping between Requirements and Implemented Functions

1- ADMIN

Functional requirements	Functions/Class that implemented this feature
Login.	<code>public function loginPost(Request \$request)</code> path: app/http/controller/AdminController.php
Logout.	<code>public function logout(Request \$request)</code> path: app/http/controller/AdminController.php
View message of contact customer service come from seller.	<code>public function index()</code> path: app/http/controller/Admin/sellersContactUsController.php
View message of contact customer service come from customer.	<code>public function index()</code> path: app/http/controller/Admin/customersContactUsController.php
View statistic.	<code>public function home()</code> path: app/http/controller/AdminController.php
View admin's account information.	<code>public function index()</code> path: app/http/controller/Admin/adminsController.php
edit admin's account information.	<code>public function update()</code> path: app/http/controller/Admin/adminsController.php
Add admin account.	<code>public function store(Request \$request)</code> path: app/http/controller/Admin/adminsController.php
Create sellers' accounts.	<code>public function store()</code> path: app/http/controller/Admin/sellersController.php
View sellers' accounts information.	<code>public function index()</code> path: app/http/controller/Admin/sellersController.php
Delete sellers' accounts.	<code>public function destroy(\$id)</code> path: app/http/controller/Admin/sellersController.php
Edit sellers' accounts.	<code>public function update(\$id)</code> path: app/http/controller/Admin/sellersController.php
Ban seller account.	<code>public function ban(\$id)</code> path: app/http/controller/Admin/sellersController.php
Unban seller account.	<code>public function unBan(\$id)</code> path: app/http/controller/Admin/sellersController.php
View customers' accounts information.	<code>public function index()</code> path: app/http/controller/Admin/customerContactUsController.php
Delete customers' accounts.	<code>public function destroy(\$id)</code> path: app/http/controller/Admin/customersController.php
Edit customers' accounts.	<code>public function update(\$id)</code> path: app/http/controller/Admin/customersController.php
Ban customer accounts.	<code>public function ban(\$id)</code> path: app/http/controller/Admin/customersController.php

Unban customer accounts.	<code>public function unBan(\$id)</code> path: app/http/controller/Admin/customersController.php
View offers.	<code>public function index()</code> path: app/http/controller/Admin/offersController.php
Delete an offer.	<code>public function destroy(\$id)</code> path: app/http/controller/Admin/offersController.php
Edit an offer.	<code>public function update(\$id)</code> path: app/http/controller/Admin/offersController.php
Create category.	<code>public function store(Request \$request)</code> path: app/http/controller/Admin/categoriesController.php
View category.	<code>public function index()</code> path: app/http/controller/Admin/categoriesController.php

Table 1. Admin's Requirements

2- SELLER

Functional requirements	Functions/Class that implemented this feature
Login.	<code>public function login(Request \$request)</code> path: app/http/controller/sellerController.php
Logout.	<code>public function logout()</code> path: app/http/controller/sellerController.php
Add offer.	<code>public function addOffer(Request \$request)</code> path: app/http/controller/sellerController.php
Edit an offer.	<code>public function editOffer(\$id , Request \$request)</code> path: app/http/controller/sellerController.php
Delete an offer.	<code>public function deleteOffer(\$id)</code> path: app/http/controller/sellerController.php
View their offers.	<code>public function myOffers()</code> path: app/http/controller/sellerController.php
Contact customer service.	<code>public function contactCustomerService(Request \$request)</code> path: app/http/controller/sellerController.php
View statistic.	<code>public function statistics()</code> path: app/http/controller/sellerController.php

Table 2. Seller's Requirements

3- CUSTOMER

Functional requirements	Functions/Class that implemented this feature
Signup.	<code>public function signUp(Request \$request)</code> path: app/http/controller/customerController.php
Login.	<code>public function login(Request \$request)</code> path: app/http/controller/customerController.php
Logout.	<code>public function logout()</code> path: app/http/controller/customerController.php
Edit account.	<code>public function update(Request \$request)</code> path: app/http/controller/customerController.php
Delete account.	<code>public function delete()</code> path: app/http/controller/customerController.php
view categories.	<code>public function getAllCategories()</code> path: app/http/controller/homeController.php
View offers.	<code>public function offers()</code> path: app/http/controller/customerController.php
Search an offer.	<code>public function offers()</code> path: app/http/controller/customerController.php
View offer.	<code>public function getOffer(\$id)</code> path: app/http/controller/customerController.php
view a seller information.	<code>public function getSeller(\$id)</code> path: app/http/controller/customerController.php
Send notification for a specific search keyword saved by customer when seller add an offer.	<code>public function CustomerKeyword(Request \$request)</code> path: app/http/controller/customerController.php
send notification for new price of specific offer saved by customer when seller changes price.	<code>public function saveOffer(Request \$request)</code> path: app/http/controller/customerController.php
Rate offer including comment	<code>public function rateOffer(Request \$request)</code> path: app/http/controller/customerController.php
Rate seller.	<code>public function rateSeller(Request \$request)</code> path: app/http/controller/customerController.php
View offers' rate.	<code>public function getOffer(\$id)</code> path: app/http/controller/customerController.php
View sellers' rate.	<code>public function getOffer(\$id)</code> path: app/http/controller/customerController.php
Save favorite offer.	<code>public function saveOffer(Request \$request)</code> path: app/http/controller/customerController.php

View favorite offer	<code>public function getSavedOffer</code> path: app/http/controller/customerController.php
View statistic.	<code>public function statistics()</code> path: app/http/controller/customerController.php
Similar offers.	<code>public function getSimilarOffers(\$id)</code> path: app/http/controller/customerController.php
Contact customer service.	<code>public function contactCustomerService(Request \$request)</code> path: app/http/controller/customerController.php

Table 3. Customer's Requirements

3.3 Implementation Details

1.1 Api (add offer)

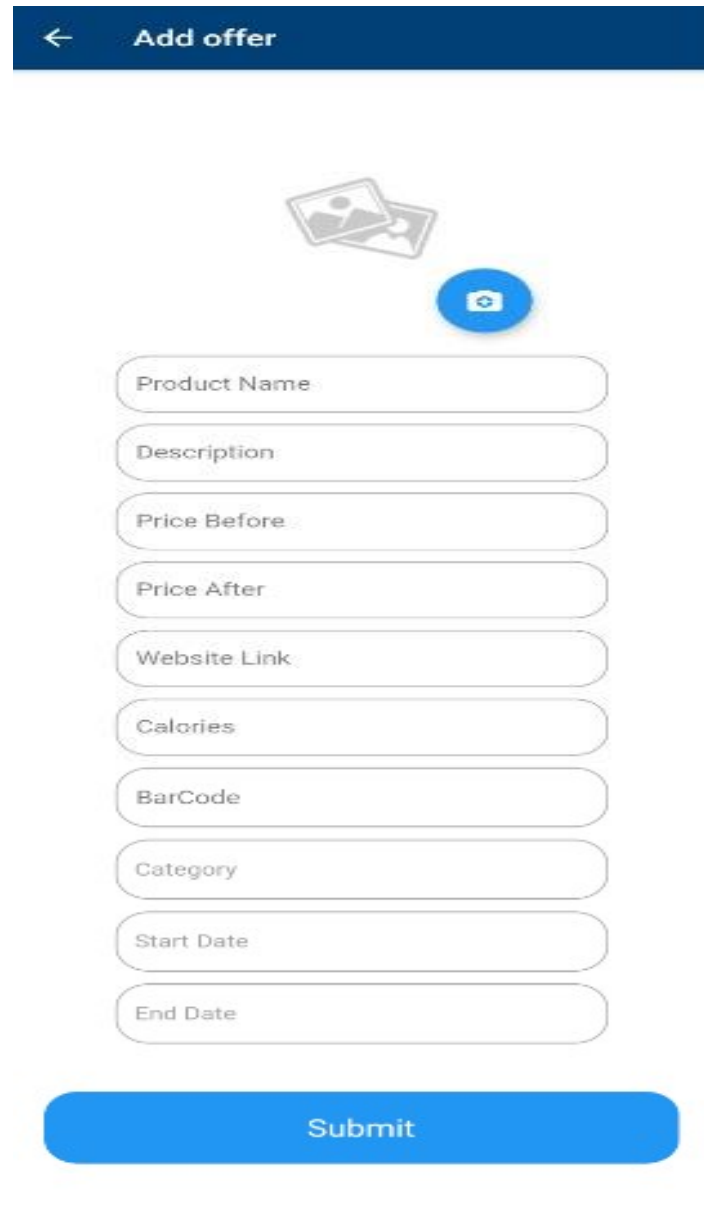
```
Route::post('offer' , 'SellersController@addOffer');
```

1.2Add offer function

```
public function addOffer(Request $request)
{
    $data = $request->validate([
        'name'          => "required",
        'description'    => "required",
        'priceBefore'    => "required",
        'priceAfter'     => "required",
        'website'        => "required",
        'calories'       => "required",
        'photo'          => "required",
        'dateStart'      => "required|date",
        'dateEnd'        => "required|date",
        'barCode'        => 'required',
        'category_id'    => 'required'
    ]);
    $photo = $request->get('photo');
    if($photo){
        $file_name = saveBase64Image($photo);
        $data['photo'] = $file_name ;
    }
    $data['seller_id'] = auth()->user()->id ;
    $offer = Offer::create($data);
    return ["offer" => $offer, "success" => true];
}
```

1.3 Graphical UI

As illustrated in Figure 10, the seller fills the fields of new offer that provides of offer's name, image, price before discount, price after discount, period of offer, seller's website, description of offer and serial number of offer to compare with other similar offers after that tap on publish.



The screenshot shows a mobile application interface for adding a new offer. At the top, there is a dark blue header bar with a white back arrow icon and the text "Add offer". Below the header, there is a large, light gray rectangular area for uploading an image, indicated by a faint image icon and a blue circular button with a white camera icon. Below this area, there is a vertical stack of ten rounded rectangular input fields, each with a placeholder text: "Product Name", "Description", "Price Before", "Price After", "Website Link", "Calories", "BarCode", "Category", "Start Date", and "End Date". At the bottom of the form, there is a large, solid blue button with the white text "Submit".

Figure 10. Add an offer

2.1 API (view offer)

```
Route::get('offers/{id}', 'CustomersController@getOffer');
```

2.2 View offer function

```
public function getOffer($id){
    $offer = Offer::whereId($id)
        ->with(['seller','rates'])
        ->first();
    $response = array("succes" => false );
    if($offer){
        $response = array(
            "success" => true ,
            "offer"   => $offer
        );
    }
    return $response;
}
```


2.3 Graphical UI

As illustrated in Figure 11, after publishing, customer can view details of an offer, similar offers and rates with comments of other customers who rated this offer.

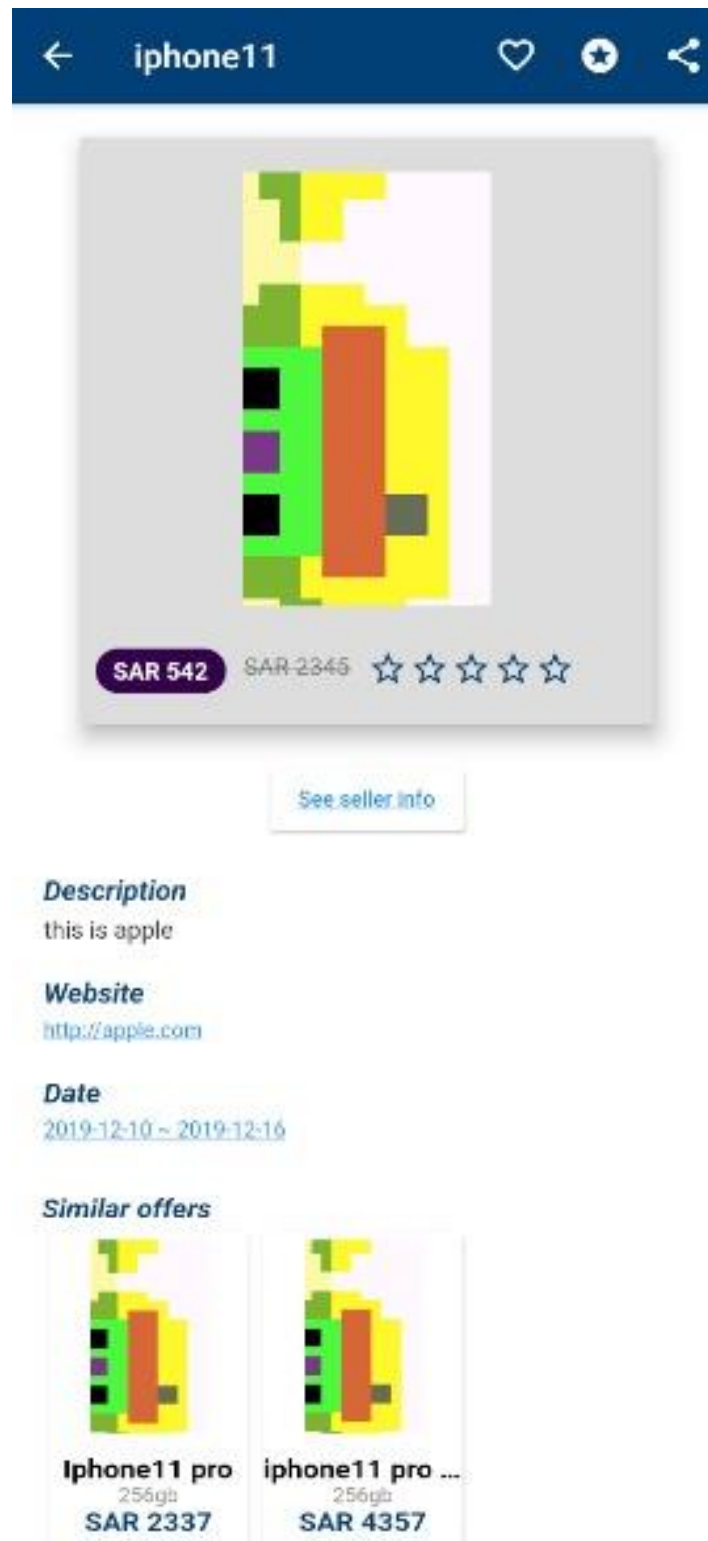


Figure 11. View an offer

3.1 API (view categories)

```
Route::get('categories' , 'HomeController@getAllCategories');
```

3.2 view categories function

```
public function getAllCategories(){  
    $categories = Category::all();  
    return ["categories" => $categories, "success" => true];  
}
```

3.3 Graphical UI

As illustrated in Figure 12, When you want to show offers, you must before select category for more usability



Figure 12. View categories

3.4 Actual Database Schema

In Figure 13, we provide screen shot of the developed database, shows how the system will store data.

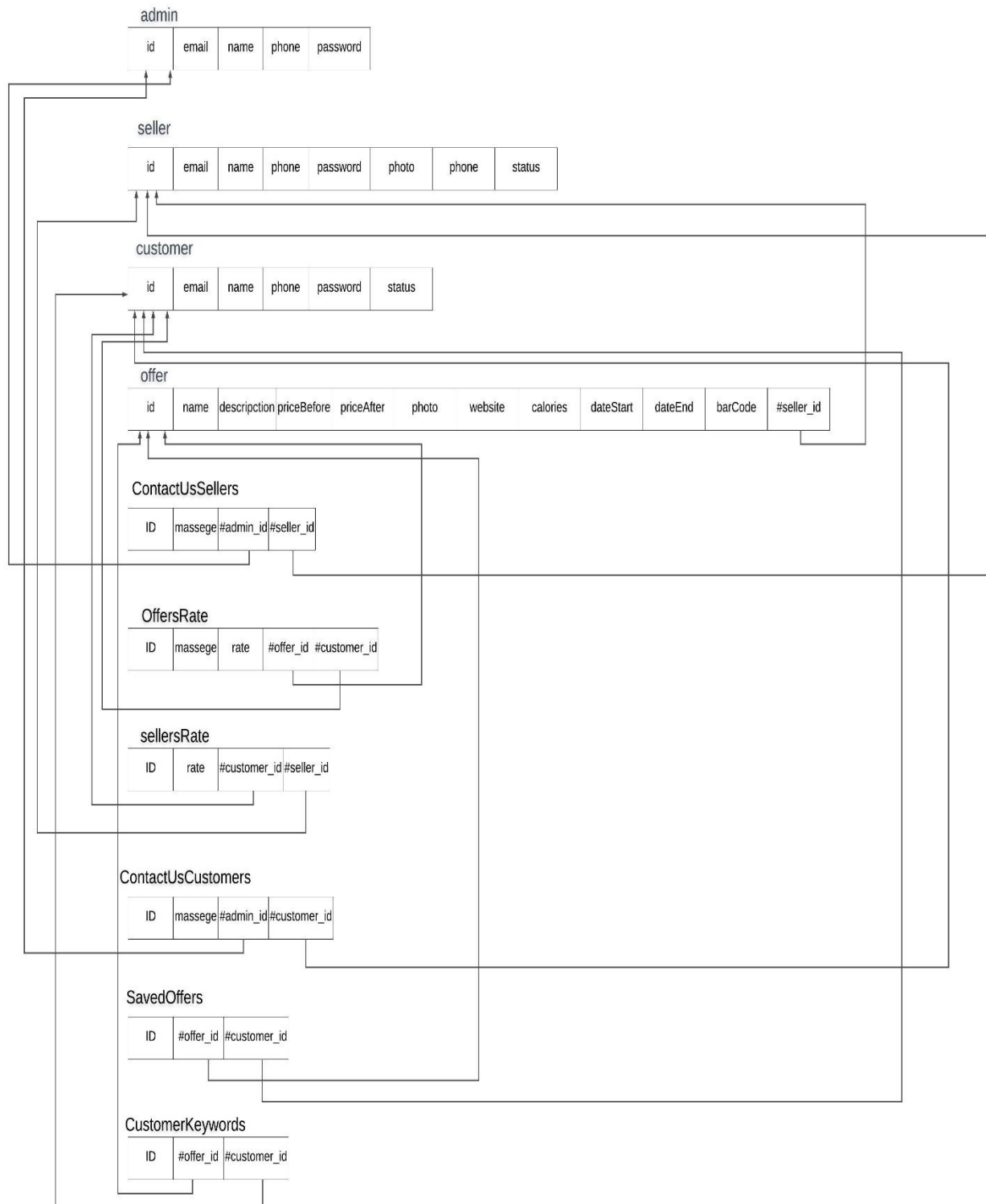


Figure 13. Database schema

4. Testing

We conducted several tests on the system in web application to make sure that the system functionality is working as meant to be. In this part of the document we list some test cases, unit tests, functional tests and Junit test.

4.1 Expected Test Scenarios

We tested the following functions:

- 1) Login to the system.
- 2) Add products.
- 3) Search for a product.
- 4) Customer signup.
- 5) View all products.
- 6) Contact customer service.

4.2 Unit Test

Unit testing is a testing approach that focuses on testing some part of an application.

We have performed tests on some units, the table below shows the result of these tests.

Test Unit	Goal	Expected Value	Actual Value	Test Result
Add products [Figure 14]	Ensuring that a seller can add product.	The product added successfully	The product added successfully	Pass
View all products [Figure 15]	Ensuring that the products there	The products displayed	The products displayed successfully	Pass
Signup [Figure 16]	Ensuring that a customer can signup without helping	The customer signup Successfully	The customer signed up	Pass

Table 4. Unit test function

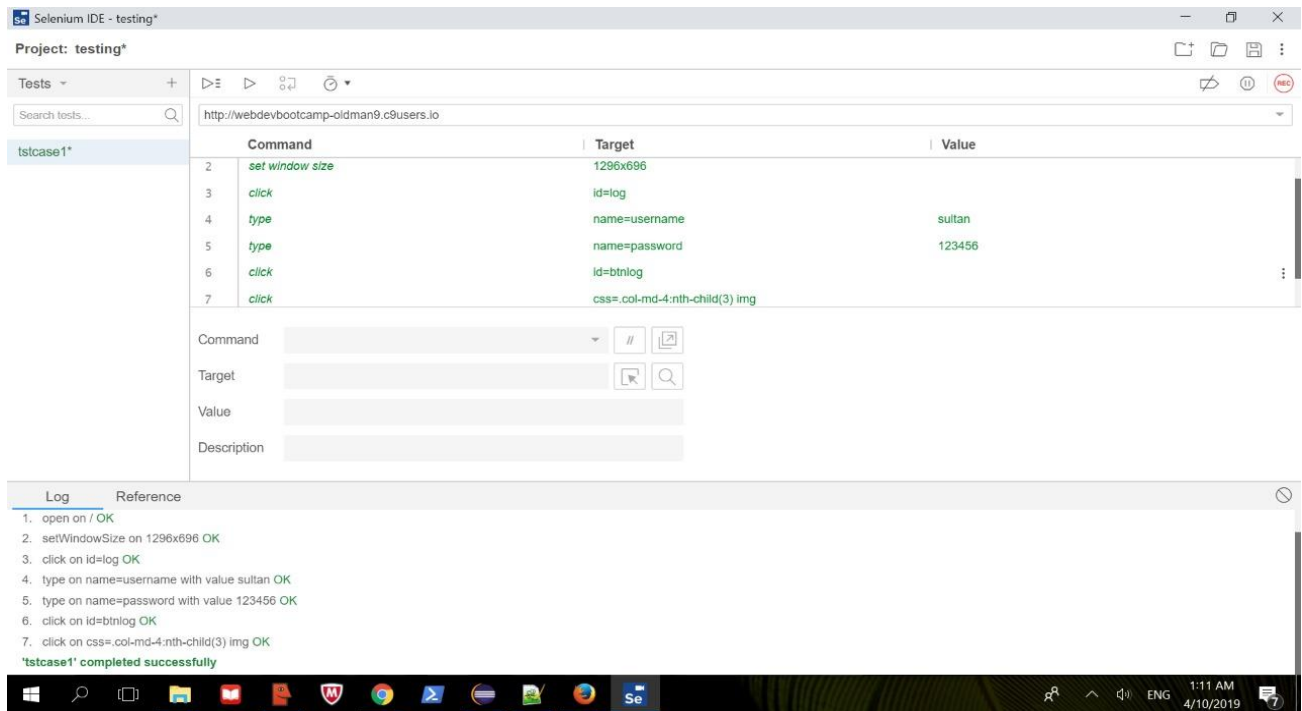


Figure 14. Selenium test case1.

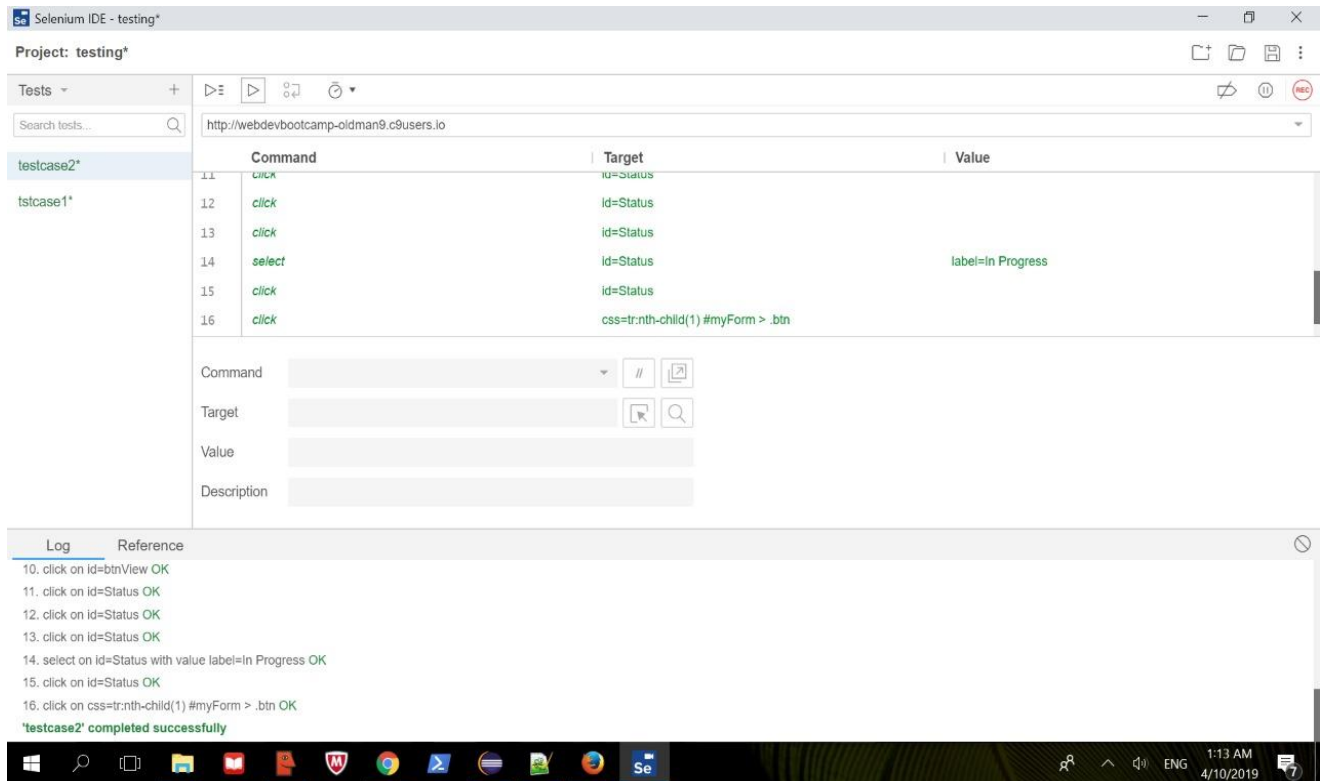


Figure 15. Selenium test case 2.

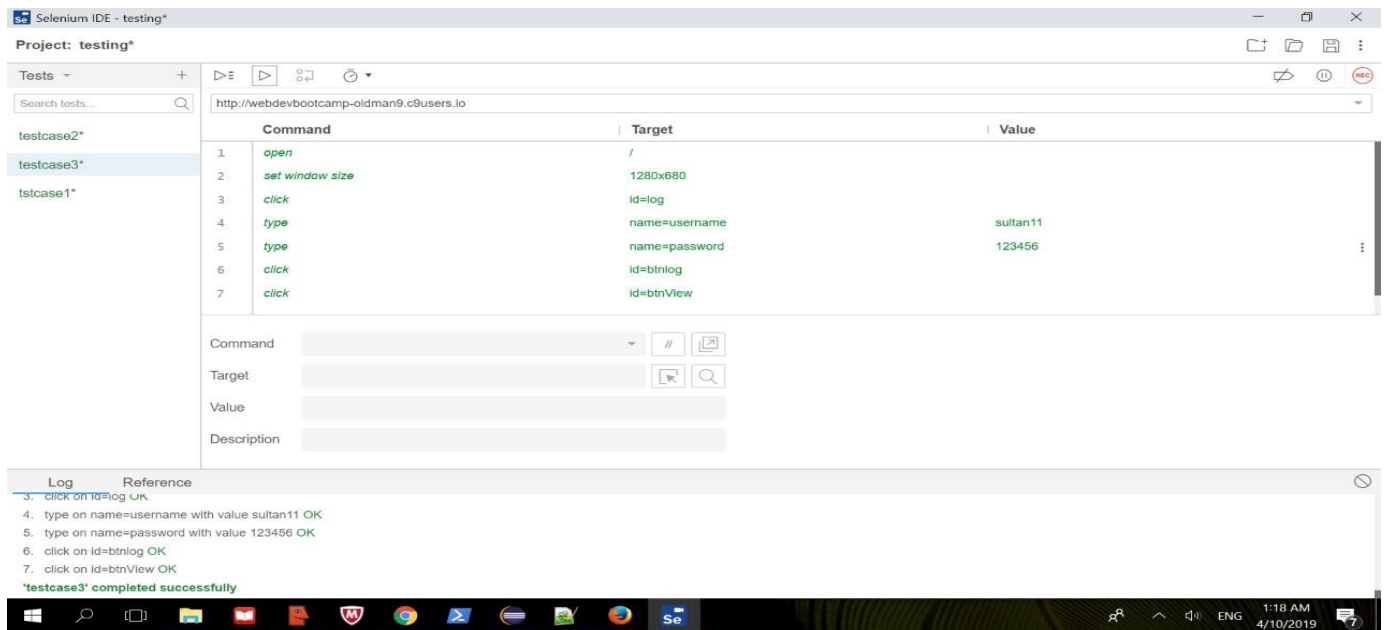


Figure 16. Selenium test case 3

4.3 Functional Test

- Scenario Testing

1-Function to be tested: Add product

ID	Scenario	Expected Output	Precondition	Actual Output	Test status
1	Login to the system	Log user in and go to home page	-	As Expected	Pass
2	Seller click on add product button	Add product form is shows	-	As Expected	Pass
3	Seller fill the blanks	The product added	-	As Expected	Pass

Table 5. Scenario testing.

2-Function to be tested: View products

ID	Scenario	Expected Output	Precondition	Actual Output	Test status
1	Login to the system	View category page	System ask for username and password	As Expected	Pass
2	Choose category	The home page	-	As Expected	Pass
3	The products displayed	Products displayed	-	As Expected	Pass

Table 6. Tested functions.

- Equivalence Classes Testing

1-Adding an offer Test case:

Input Condition	Valid Equivalence Classes	Invalid Equivalence Classes
Product Name	Length >0 (1) Length <=20 (2)	Length = 0 (3) Length >6 (4)
Description	Length>0 (5) Length<=60 (6)	Length=0 (7) Length >60 (8)
Price Before	Numeric value >0 (9) Numeric value <=6 (10)	Numeric value =0 (11) Numeric value >6 (12)
Price after	Numeric value >0 (13) Numeric value <=6 (14)	Numeric value =0 (15) Numeric value >6 (16)
Website link	Length >0 (17) Length <= 40 (18)	Length = 0 (19) Length >10 (20)
Calories	Numeric value >0 (21) Numeric value <=5 (22)	Negative number (23) Numeric value=0 (24) Numeric value>5 (25)
BarCode	Numeric value >0 (26) Numeric value <=15 (27)	Numeric value =0 (27) Numeric value >15 (28) Negative number (29)
Category	Length >0 (30) Length <=10 (31)	Length =0 (32) Length >10 (33)
Start date	Date (34)	Date > current date (35)
End date	Date (35)	-

Table 7. Equivalence classes testing.

2-Valid Equivalence Classes

Use Case	Use Case Data	Boundaries	Class(es) Covered
1	X s 2 1 https://www.a.com 0 111011101 Electronics 2019/11/20 2019/12/20	Lower	(1)(5)(9)(13)(17)(21) (26)(30)(34)(35)
2	Nova bottle of water Bottle of water 1.5letter 1.5 1 https://www.nova.com 0 111011101 Food 2019/11/20 2019/12/20	Upper	(2)(6)(10)(13)(18)(22) (27)(31)(34)(35)

Table 8. Valid equivalence classes.

- **Branch coverage testing**

1-Use Case: Ban user.

Actor: Administrator.

Pre-conditions:

Administrator must be logged in.

Basic Scenario:

1. Administrator click on users' button.
2. System displays Users page.
3. Administrator choose a specific user.
4. System display User information.
5. Administrator Click on band Button.
6. System added the user to banned page.

Alternative Flows:

- 1.a: In step 1, seller does not enter his username correctly.

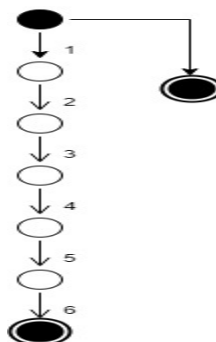


figure 17. Test Scenario.

Scenarios for take a test:

The table shown below describes all possible scenario paths.

ID	Events	Description
1	1-2-3-4-5-6	The administrator ban the user.
2	1-1.a	The administrator incorrectly

Table 9. Scenario path for test case.

2.Use Case: View statistic.

Actor: Seller.

Pre-conditions:

Statistics must be displayed.

Basic Scenario:

1. Seller Login to his user.
2. Click on the view statistics.
3. statistic will be displayed.

Alternative Flows:

- 1.a: In step 1, seller does not enter his username incorrectly.

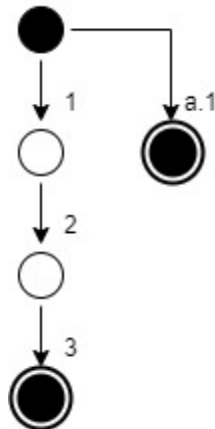


figure 18. Test Scenario.

Scenarios for take a test:

The table shown below describes all possible scenario paths.

ID	Events	Description
1	1-2-3	Seller displayed the statistics.
2	1-1.a	Seller don't enter his username incorrectly.

Table 10. Scenario path for test case.

4.4 Usability Test

Since software usability is major issue to consider in software development we were confused how to measure the usability of our website, so we decided to create a survey. We read many usability surveys online and in older projects, which helped us to create our own questions based on the acquired knowledge.

Our survey was published using Google Forms. We sent the link of the form along with a link to visit the website to get user's feedback. Survey is the best way to get user's honest feedback since for them it's anonymous. With the power of social media, we got 23 results in a very short time. Google Forms arrange the results in pie chart that make it visually clear. 3 questions and their results are listed below:

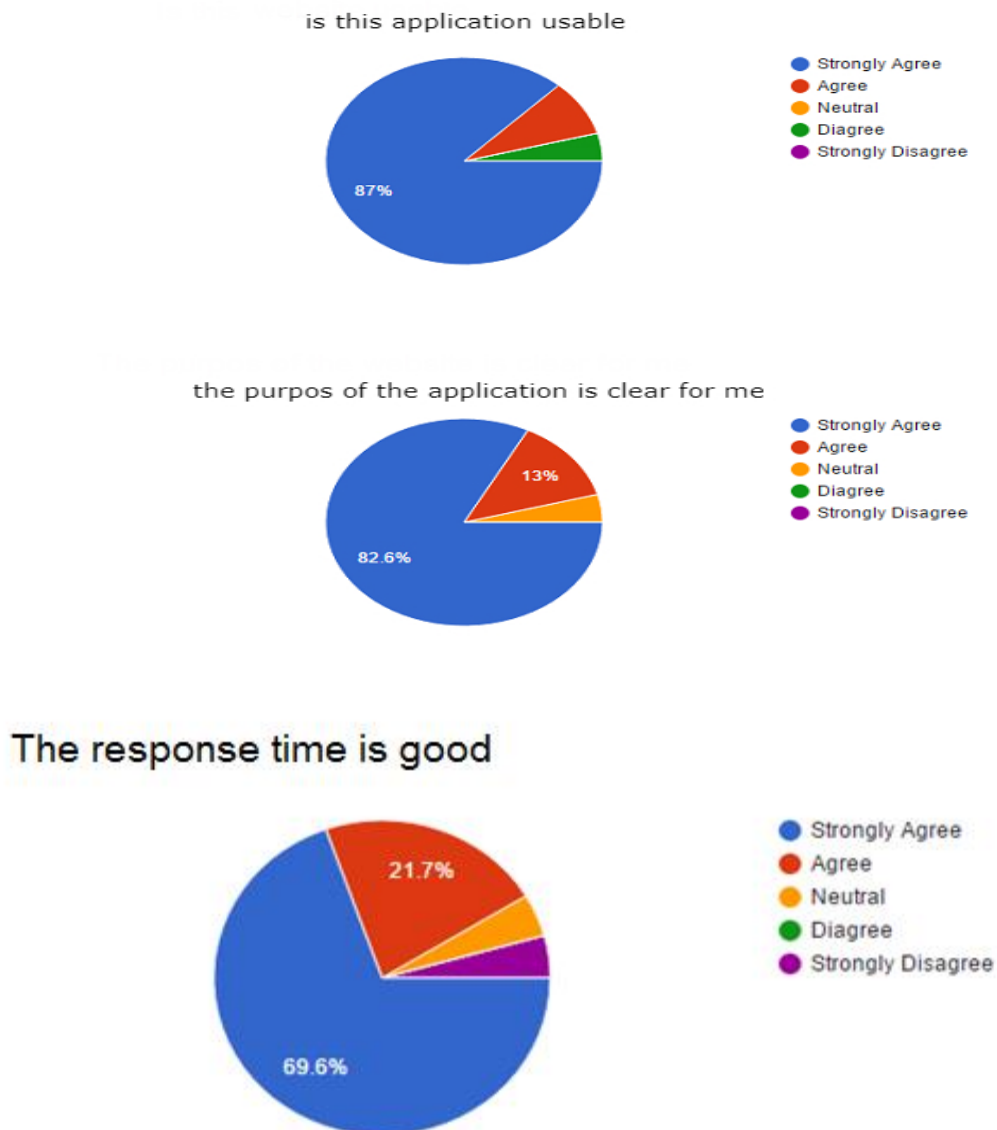


figure 19. Statistics testing.

5. Deployment of the System

In this Figure 20, we showcase the deployment of our system in the hardware environment. As we can see the mobile connects to the application server which is connected to the database server.

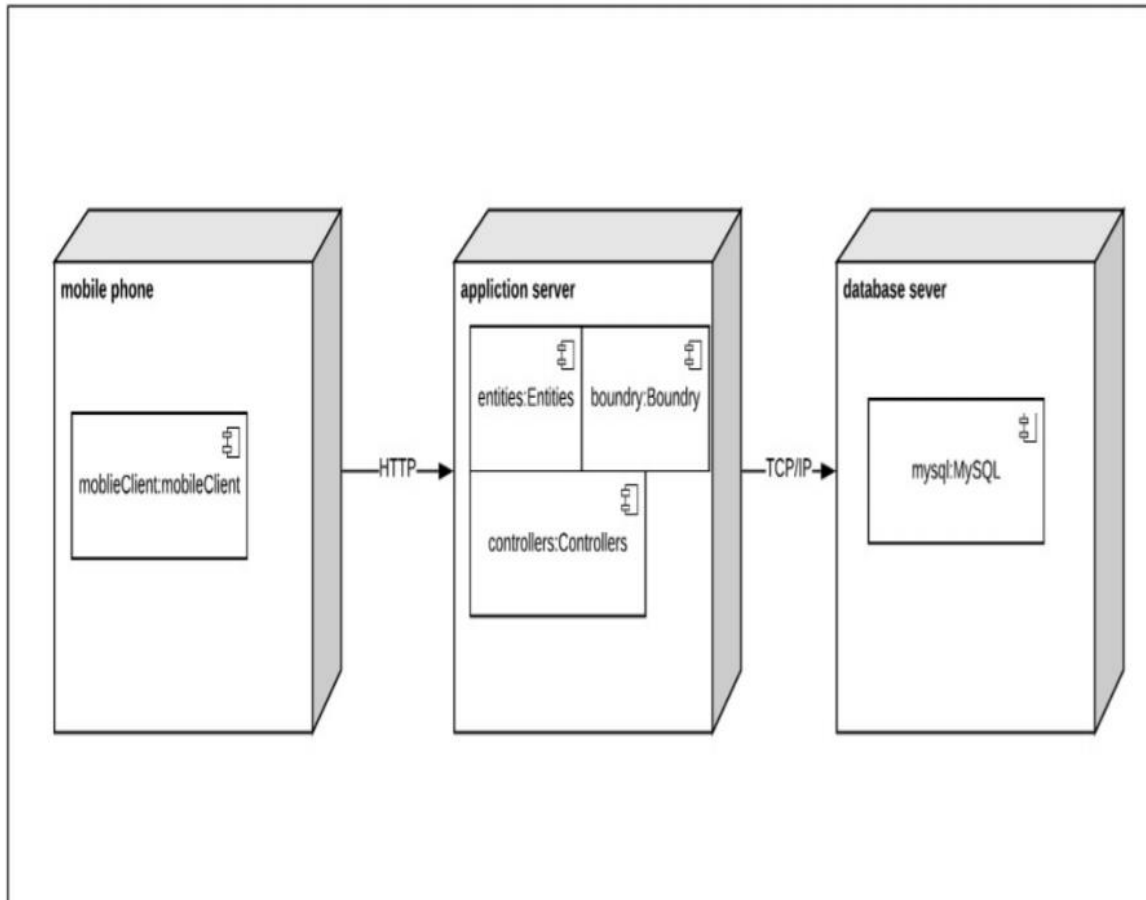


Figure 20. Expected Deployment

6. Limitation of the System

- Time & Lack of experience are the greatest obstacles in our way.
- Inability to test a few non-functional requirements in part one of the system.
e.g. availability: The system shall be available 95% of the time.

7. Conclusion and Future Work

In this document we have updated our first phase of the graduation project documentation so that it matches the new and developed state of the system.

We hope to further develop our system and refine its features so that it triumphs over other similar applications in the market. We strive to be the best and to achieve that we need to include features of delivery as that feature is essential in this type of applications. On top of that we would like to include more markets of different varieties in the application so that it attracts more customers of different backgrounds and interests.

We hope also to keep customers satisfied so we will try to keep the application constantly free of bugs. Customer satisfaction is our utmost priority. We wish to continue developing our application whenever possible and keep maintaining our quality.

8. Reference

- 1- How to calculate TAX <https://www.vat.gov.sa/ar/vat-rate> [last accessed: 21/12/2019]
- 2- Selenium testing <https://selenium.dev/> [last accessed: 21/12/2019]
- 3- Drawing diagram <https://www.draw.io/> [last accessed: 21/12/2019]