“Shop with points”

Moibile Application

Proposed by:

Alireza Samadifardheris 1961823

Ashkan Ansarifard 1970082

<https://github.com/alirezasamadii/ShopwithPoints>

**Contents**

[**1. General high level description of the application 3**](#_Toc106873313)

[**2. Mock ups: 6**](#_Toc106873314)

[**3. FUNCTION POINT COUNTING PROCESS 7**](#_Toc106873315)

[**Introduction 7**](#_Toc106873316)

[** Record Element Type (RET) 8**](#_Toc106873317)

[** File Type Referenced (FTR) 8**](#_Toc106873318)

[** Data Element Type (DET) 8**](#_Toc106873319)

[** Rating 8**](#_Toc106873320)

[** Transaction DET’s 8**](#_Toc106873321)

[** EXTERNAL INPUTS 9**](#_Toc106873322)

[** EXTERNAL OUTPUTS 9**](#_Toc106873323)

[** EXTERNAL INQUIRIES 9**](#_Toc106873324)

[** INTERNAL LOGICAL FILES: 10**](#_Toc106873325)

[** EXTERNAL INTERFACE FILES 10**](#_Toc106873326)

[**4. CoCoMo II 11**](#_Toc106873327)

[** The 5 Scale Drivers are: 11**](#_Toc106873328)

[** Product Factors: 11**](#_Toc106873329)

[** Platform Factors: 11**](#_Toc106873330)

[** Project Factors: 11**](#_Toc106873331)

[**COCOMO II Effort Equation 12**](#_Toc106873332)

[** Effort Adjustment Factor 12**](#_Toc106873333)

[** COCOMO II Schedule Equation 12**](#_Toc106873334)

[** Function Point Modeler And COCOMO 12**](#_Toc106873335)

[**5. Scrum 14**](#_Toc106873336)

[**6. Docker 17**](#_Toc106873337)

[**7. Final User Interface 17**](#_Toc106873338)

# General high level description of the application

The application we would like to implement is a mobile application called **shop with points** and the idea is that the more users contribute to the environment of the application to review businesses (restaurants, quality of cloths, behavior of hotel staff or any other business), the more they will get points. Later they can use their collected points to get benefits from discounts from any other business registered in the application.

In this application, there are two main types of users:

1. The first type of users are owners of businesses. The owner of the business requests to create a business on the application, saying a restaurant, and then the email address of that user would be registered as the business owner’s email address.
2. The second type of users are normal users: these users have the capability of rating businesses (for instance: restaurants) and posting reviews for them. The more they contribute by giving feedback and reviews and improving the community, the more they will have points in their wallet. These points can be used, to benefit from some discounts from any shop. (one point per review)

Note that, it is the business owner who decides how much discount he would like to give to a specific amount of points by putting this information on their page when they request to create an account. Some examples could be:

* 10% discount on pizza by 10 coins redeemed from your wallet
* 1 Cola if you spend at least 10 euros by redeeming 7 credits from your wallet
* Buy jeans and get up to 60% discount ( 1 credit = 1% )

You may ask why business owners should care about some virtual credits in their wallet? Though this part is not intended to be implemented yet , probably they can use sum of credits redeemed from their customers to get some benefits provided by the app ( payment from an income source of the app to owners, advertising their business through this app, or any other app, etc )

* 1. **Authentication:**

The users’ log-in and account creation will be securely authenticated by authentication-providing services (such as Auth0) that provide user management like associating roles to users. It is only based on login information, where the user is going to be directed (either owner page or normal user page)

* 1. **The frontend:**

1. **Business owner users:** These users have the ability to redeem from normal users’ credits upon credit owners’ request. Owner of business is connected to normal user’s wallet by scanning with the camera a QR code generated on normal users’ shopwithPoints Mobile application, then they can redeem the credits from their wallet.
2. **Normal users:** logging in as a normal user will allow users to see businesses nearby on the map by pins as well as the list of businesses. Users should be able to go to the businesses’ page, see the name, description, how much discount would they earn by a specified amount of credits as well as comments by other users and the average score of the restaurant. The users are also allowed to write review and score the restaurant (maximum one review per user per business is the easiest way to avoid spam and we aim to have it in this way). The should have ability to go to a section , say wallet, to check how many credits they gathered so far and also generate a QR code valid only for a short period to be scanned by a business owner. (The idea of using QR code is to exchange the information of a JWT which could contain information about account information and the number of points user has.
   1. **The Backend**

In each view, there will be handled some CRUD operations by the use of REST APIs for interacting with a database application. the information is retrieved by API calls asynchronously to avoid UI becoming unresponsive meanwhile since API calls are usually considered slow procedures. when calling an API, a completion handler must be specified which is a defined function and will be executed after data are retrieved and a new thread will be created to wait for the result so the performance of the application which is running on the main thread won’t be affected. Some obvious examples of these API calls that they are highly probable to be needed and implemented are:

asynchronous call of the API retrieves (GET)the information for each element on the floating panel where user can see the lists of the businesses

asynchronous call of the API retrieves(GET) the information for each pin on the map where user can see the location of the businesses

asynchronous call of the API retrieves (GET)the information of businessID for the tapped business including image, description, average score, as well as comments, the username of the commenter, and the score to be shown on the new view

asynchronous call of the API posts (POST) the comment and score of user with UserID for a business with businessID. when submit review is tapped if the text is null, then the application must request for the text of the review, else a review will be created containing user-id, review text, review score, the user id is saved in a data holder shared all among the views and the app. Then when the review is created the API call should post this and return ok if successful, else it will return an error. After submitting the review, our review should be seen in the reviews, and the average score is should be also updated. This can be also done for any other business

asynchronous call of the API deletes (DELETE) the comment and score of user with UserID saved in a data holder on the restaurant with businessID

asynchronous call of the API retrieves (GET) wallet credits of the user saved in the database to be shown in the wallet section

asynchronous call of the API retrieves (GET) wallet credits of the user saved in the database to be shown in redeem page of business owner

asynchronous call of the API updates (UPDATE) wallet credits of the user saved in the database decreased proportionally as much as requested. business owner from BusinessOwnerView can simply open the camera through the app, point the camera to the QR code generated by the user, if the QR code is valid, it will be directed to another view saying partner redeem view controller that asks how many points of the user want to be used, when the points redeems successfully it will be sent through API call to decrease from user points and return a successful redeemed done message to partner. On the user side also, the points should proportionally be decreased

etc.

* 1. **backend deployment**

***Frontend Swift***

user requests an action: ex. Taps on restaurant

app performs API call

AWS API Gateway receives this call

AWS API Gateway invokes lambda

Lambda returns result

Results are shown in the new view: ex.restaurant information

**Another view is initialized**

Lambda starts the code

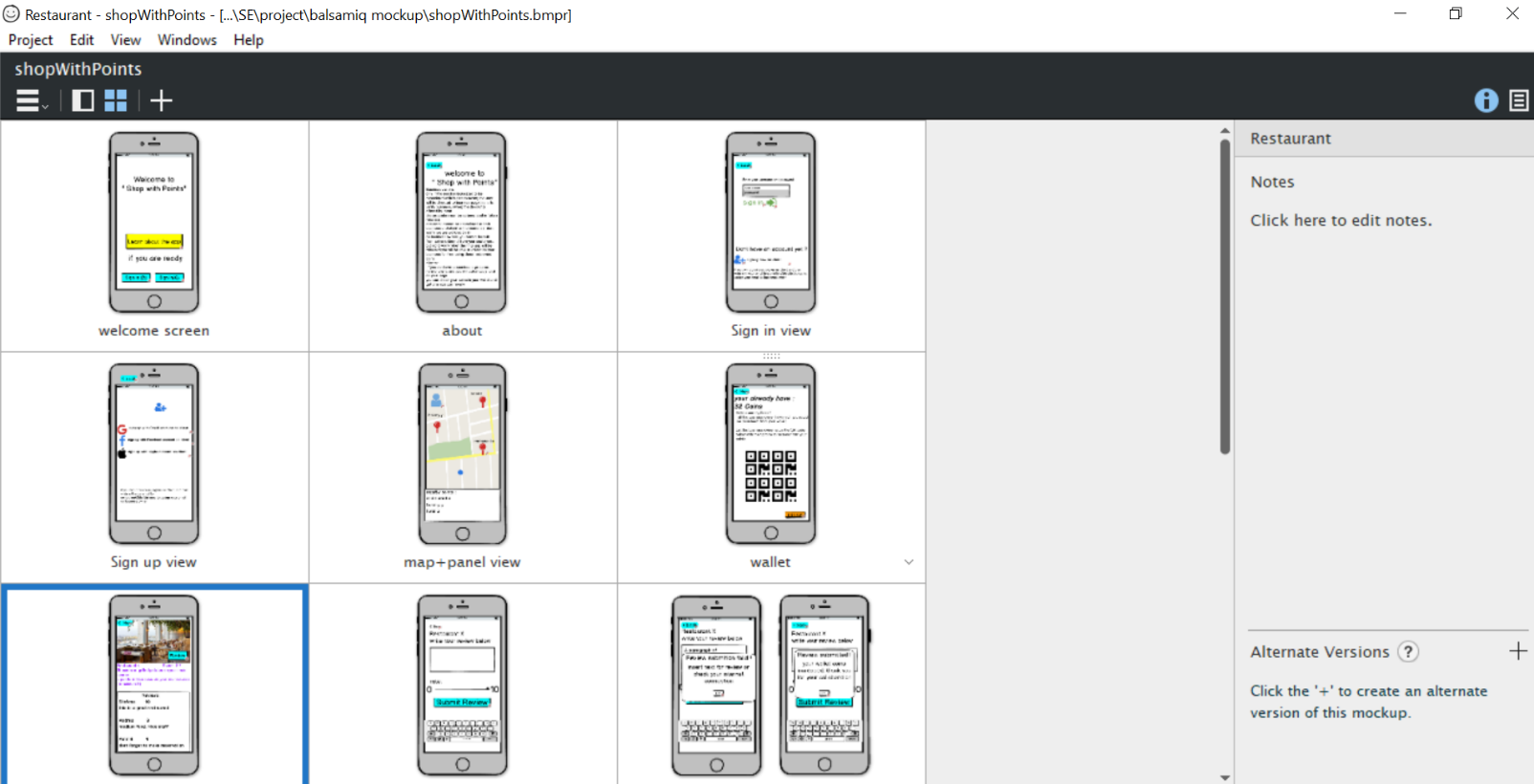
***Backend on AWS with KOTLIN***

The backend could be deployed on AWS lambda because it is powerful and free for 1 million invocations providing it the jar of backend code and that the code would be exposed to API calls. This approach is a serverless approach meaning that my code is not running on a specific server, but when it is exposed to a call, lambda will start our code, give the response to the user, and stop it.

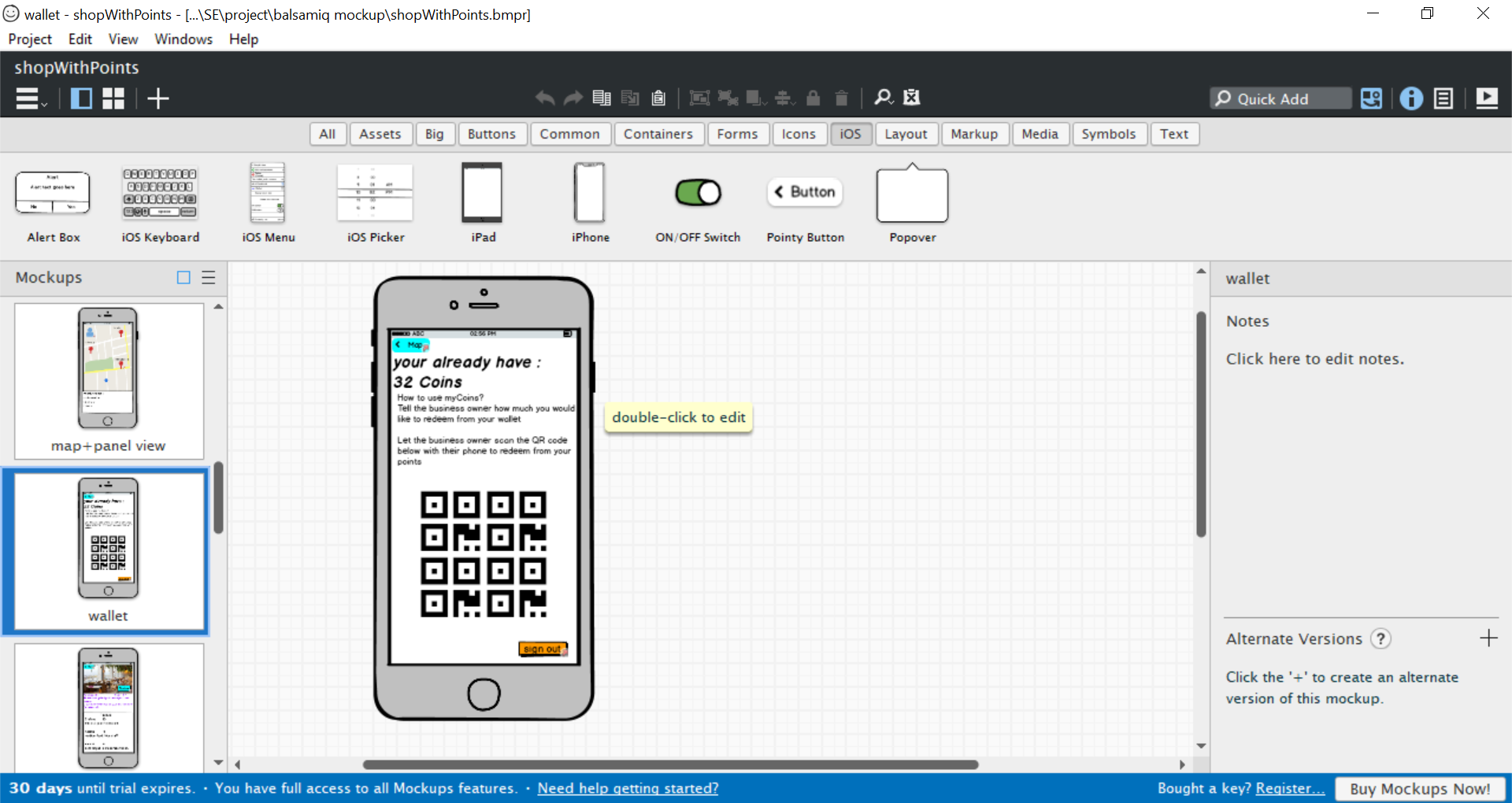
Obviously a database service (such as MongoDB) for storing data including comments, reviews, user ID, the image of the restaurant, location, credits of the user, etc.) is fundamental and we may need to retrieve filtered information over the data stored in the database.

# Mock ups:

After realizing the general idea, its time to scratch down general concept of what is in mind. To do so we used Balsamiq tool. **Balsamiq Wireframes** is a rapid low-fidelity UI wireframing tool that reproduces the experience of sketching on a notepad or whiteboard, but using a computer. It really forces you to **focus on structure and content**, avoiding lengthy discussions about colors and details that should come later in the process.  
UX of the project was created to check the functionality of the project and also to show the road of the stories. This file exists in Github repository .



Some views of UX in balsamiq environment



A view of clients wallet with generated QR code to be scanned by business owner to redeem from the wallet

# FUNCTION POINT COUNTING PROCESS

## **Introduction**

Systems continue to grow in size and complexity, becoming increasingly difficult to understand. As improvements in coding tools allow software developers to produce larger amounts of software to meet ever-expanding user requirements, a method to understand and communicate size must be used. A structured technique of problem solving, function point analysis is a method to break systems into smaller components, so they can be better understood and analyzed.

The overall objective is to determine adjusted function point count. There are several steps necessary to accomplish this. The actual sequence or order of steps is not necessary. Many counters will complete step 5 through out the entire count – gathering information as they go:

Objective of Section: Learn the necessary techniques to identify a RET, a DET and a FTR. Understanding how to identify DET’s and FTR’s is critical to distinguish one transaction from another.

## **Record Element Type (RET)**

A RET is user recognizable sub group of data elements within an ILF or an EIF.

## **File Type Referenced (FTR)**

A FTR is a file type referenced by a transaction. An FTR must also be an internal logical file or external interface file.

## **Data Element Type (DET)**

A DET is a unique user recognizable, non-recursive (non-repetitive) field. A DET is information that is dynamic and not static. A dynamic field is read from a file or created from DET’s contained in a FTR. Additionally, a DET can invoke transactions or can be additional information regarding transactions. If a DET is recursive then only the first occurrence of the DET is considered not every occurrence. Understanding the FTR’s and DET’s helped distinguish one transaction from another transactions.

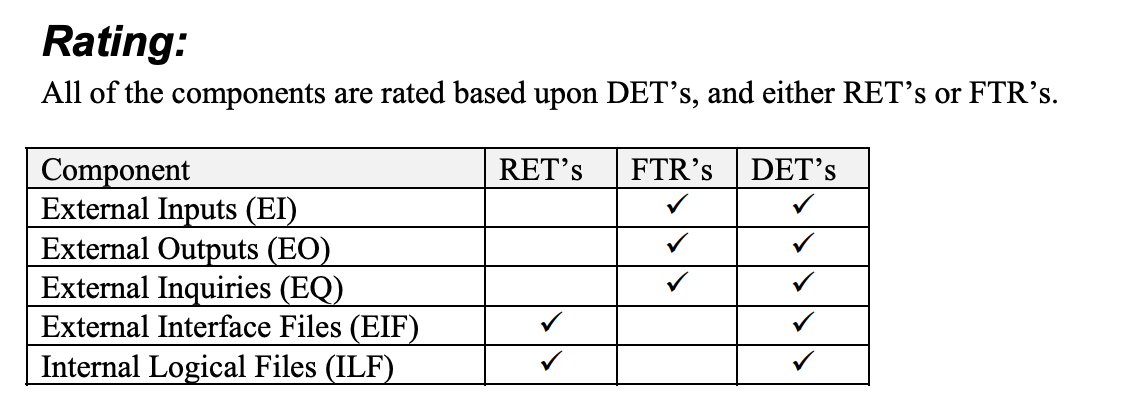
## **Rating**

All of the components are rated based upon DET’s, and either RET’s or FTR’s.

## **Transaction DET’s**

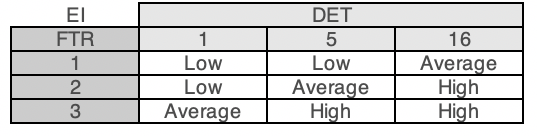
**External Inputs:** Data Input Fields, Error Messages, Calculated Values, Buttons

**External Outputs:** Data Fields on a Report, Calculated Values, Error Messages, and Column Headings that are read from an ILF. Like an EQ and EO can have an input side and output sides. External Inquiries: Input Side - field used to search by, the click of the mouse. Output side - displayed fields on a screen.



## **EXTERNAL INPUTS**

External Inputs (EI) - is an elementary process in which data crosses the boundary from outside to inside. This data may come from a data input screen or another application. The data may be used to maintain one or more internal logical files. The data can be either control information or business information. If the data is control information it does not have to maintain an internal logical file. If an external input adds, changes and deletes (maintains) information on an internal logical file, then this represents three external inputs. External inputs (especially change & delete) may be preceded by an external inquiry (see the section on external inquiries). Hence a full function screen is add, change, delete and inquiry.

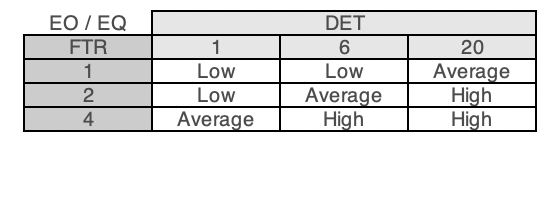


## **EXTERNAL OUTPUTS**

Describe and define the concepts necessary to identify and rate External Outputs. Definition: External Outputs (EO) - an elementary process in which derived data passes across the boundary from inside to outside. Additionally, an EO may update an ILF. The data creates reports or output files sent to other applications. These reports and files are created from information contained in one or more internal logical files and external interface files.

## **EXTERNAL INQUIRIES**

Describe and define the concepts necessary to identify and rate External Inquiries. Definition: External Inquiry (EQ) - an elementary process with both input and output components that result in data retrieval from one or more internal logical files and external interface files. The input process does not update or maintain any FTR’s (Internal Logical Files or External Interface Files) and the output side does not contain derived data.



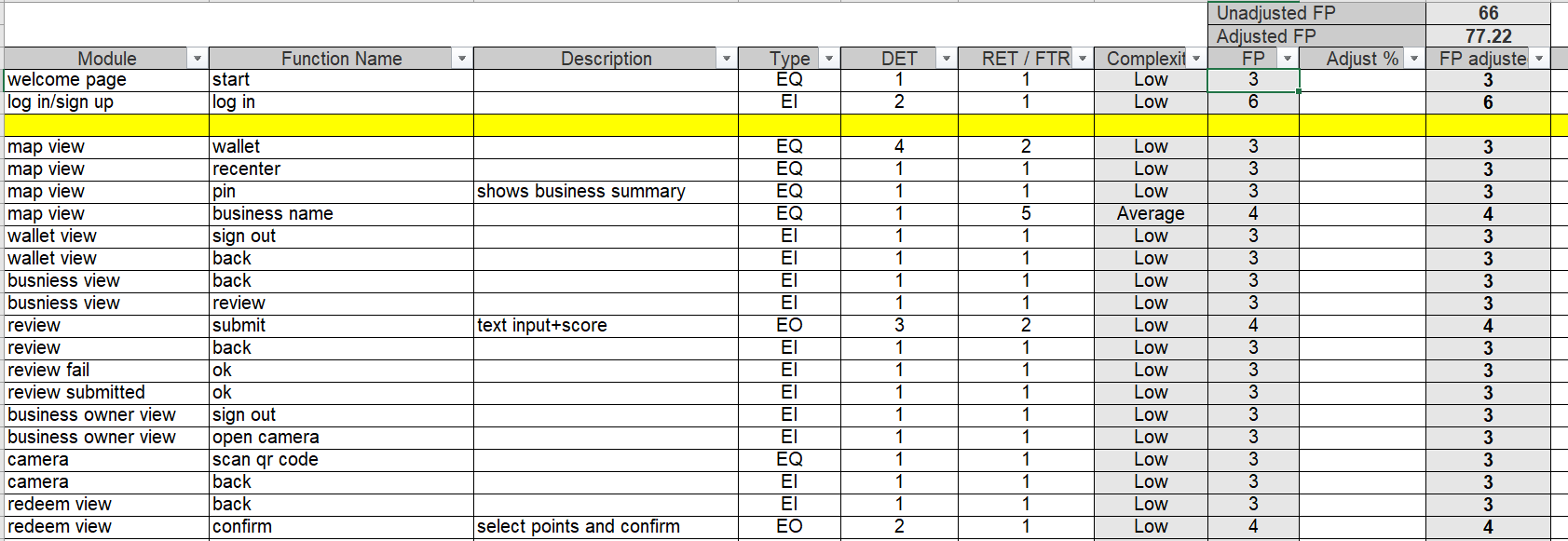
## **INTERNAL LOGICAL FILES:**

Describe and define the concepts necessary to identify and rate Internal Logical Files. Definition: Internal Logical Files (ILF) - a user identifiable group of logically related data that resides entirely within the application boundary and is maintained through External Inputs. Even though it is not a rule, an ILF should have at least one external output and/or external inquiry. That is, at least one external output and/or external inquiry should include the ILF as an FTR. Simply put, information is stored in an ILF, so it can be used later. The EO or EQ could be from another application. It is worth noting that an ILF may not be referenced by EO or EQ, but may be used by an EI (other than the EI that maintains it).

## **EXTERNAL INTERFACE FILES**

Describe and define the concepts necessary to identify and rate External Interface Files. Definition: External Interface Files (EIF) - a user identifiable group of logically related data that is used for reference purposes only. The data resides entirely outside the application boundary and is maintained by another application’s external inputs. The external interface file is an internal logical file for another application. Each EIF included in a function point count must have at least one external output or external interface file against it. At least one transaction, external input, external output or external inquiry should include the EIF as a FTR.

These calculations are provided using the delivered excel file



# CoCoMo II

The CO nstructive CO st MO del COCOMO cost estimation model is used by thousands of software project managers, and is based on a study of hundreds of software projects. In the **COCOMO II** model, some of the most important factors contributing to a project's duration and cost are the Scale Drivers. You set each Scale Driver to describe your project; these Scale Drivers determine the exponent used in the Effort Equation.

## **The 5 Scale Drivers are:**

* Precedentedness
* Development Flexibility
* Architecture / Risk Resolution
* Team Cohesion
* Process Maturity

## **Product Factors:**

* Required Software Reliability
* Database Size
* Software Product Complexity
* Required Reusability
* Documentation Match to Life-Cycle Needs

## **Platform Factors:**

* Execution Time Constraint
* Main Storage Constraint
* Platform Volatility

## **Project Factors:**

* Use of Software Tools
* Required Development Schedule
* Multisite Development

# COCOMO II Effort Equation

The **COCOMO II** model makes its estimates of required effort (measured in Person-Months **PM** ) based primarily on your estimate of software project's size (as measured in thousands of **SLOC, KSLOC** )):

Effort = 2.94 \* EAF \* (KSLOC) E

**Where:**

* EAF Is the Effort Adjustment Factor derived from the Cost Drivers
* E Is an exponent derived from the five Scale Drivers

## **Effort Adjustment Factor**

The Effort Adjustment Factor in the effort equation is simply the product of the effort multipliers corresponding to each of the cost drivers for your project.

## **COCOMO II Schedule Equation**

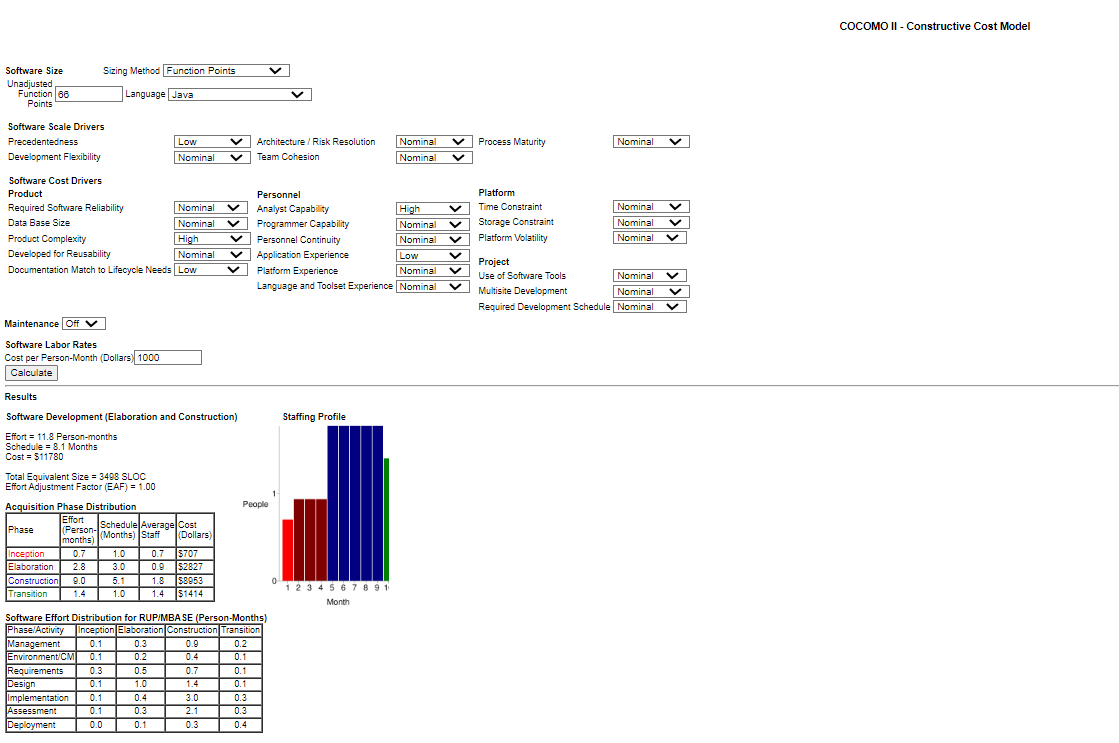
The **COCOMO II** schedule equation predicts the number of months required to complete your software project. The duration of a project is based on the effort predicted by the effort equation:

Duration = 3.67 \* (Effort) SE

## **Function Point Modeler And COCOMO**

Because **COCOMO** is well defined, and because it doesn't rely upon proprietary estimation algorithms, **Function Point Modeler** supports **COCOMO II Post Architecture** and offers these advantages to its users:

COCOMO estimates are more objective and repeatable than estimates made by methods relying on proprietary models.



# Scrum

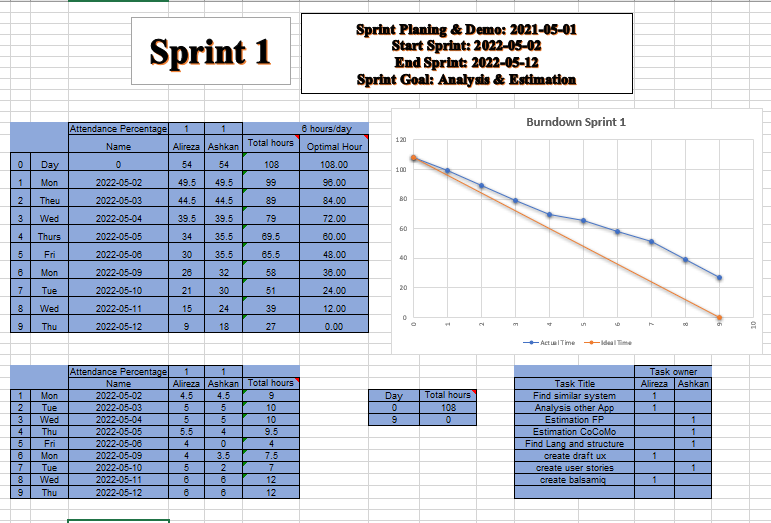
We had approximately 4 sprints. As you can see the sprint report in below table and also in proceeding detail of each sprint will be explained.

First for estimation and planning of the work time we used a excel that we created and formulated for estimated the time. According this excel based on optimal rate, availability rate and also off day estimated.

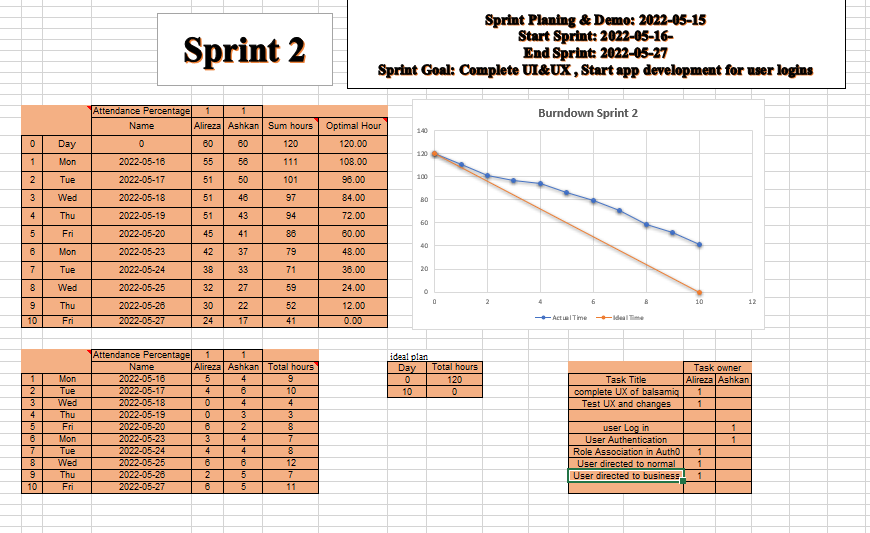
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| #Sprint | Sprint goal | Problems | Sprint status | Start Date | End Date |
| Sprint 1 | **1)Analysis**  **2)Estimation** | …. | Done | **2022-05-02** | **2022-05-12** |
| Sprint 2 | **Complete UI&UX , Start app development for user logins** | … | Done | **2022-05-16** | **2022-05-27** |
| Sprint 3 | **Complete UI&UX , Start app development for user logins** | The fundamental part of development | Done | **2022-05-30** | **2022-06-10** |
| Sprint 4 | **complete business view-complete app-complete documentation** | Overlapping with exams specially software engineering exam | Done | **2022-06-13** | **2022-06-22** |

Sprint1:

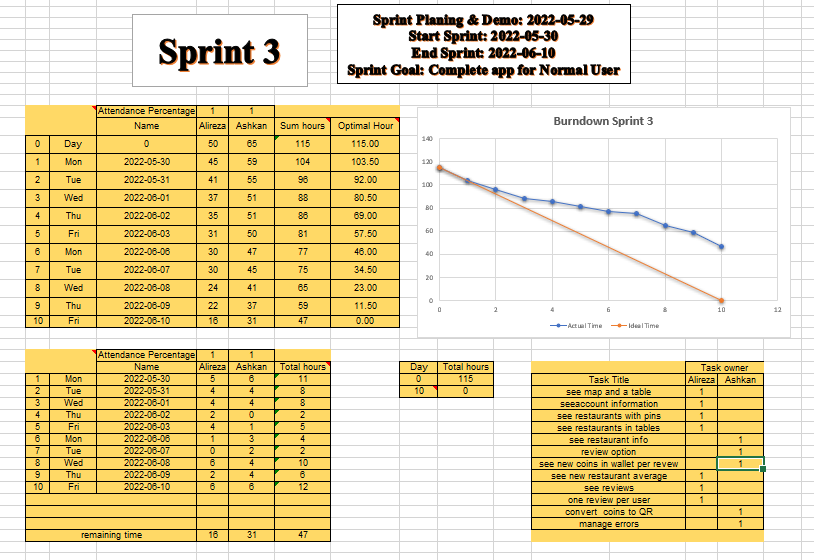
This spread sheet created for handling sprint. There are three tables in below picture. First thegeneral time of each person which is calculated by estimation excel. Second, the time working of each person in each day. And last, the general task and the owner task added to this table. In addition, you can see the sprint goal, Sprint Planning date and also start and end date of Sprint.



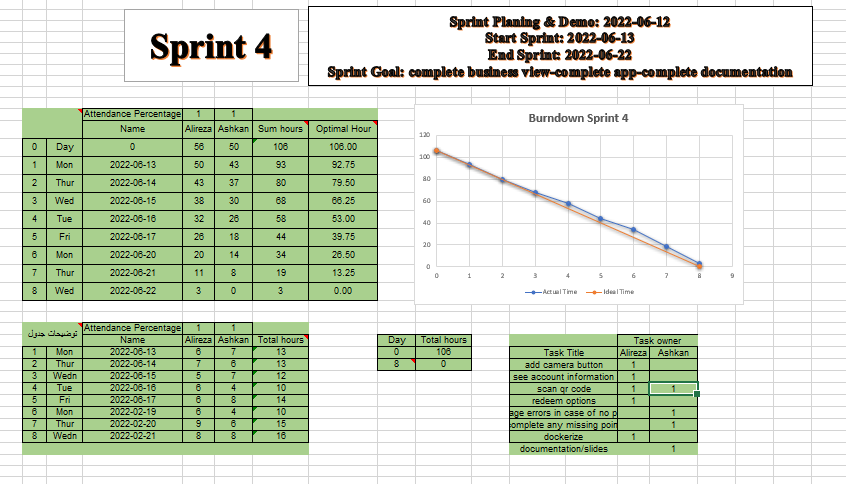
Sprint2



Sprint3



Sprint4



# Docker

Afet the code is developed, it is dockerized using docker engine . Docker **enables you to separate your applications from your infrastructure so you can deliver software quickly**. With Docker, you can manage your infrastructure in the same ways you manage your applications.

In this project we are first creating docker file to create a docker image for our backend(  java backend as a base image for our this image ) (open jdk image) and also mongo image for our backend createing two isolate containers one for backend and one for database.

To run this application on any infrastructure it is enough to run the following command.

docker compose up -d

exposing port on our local host to container is the port 8080

so if we want to see if for example Apis are working we can use postman to check them

# Final User Interface

**Some views of the final running application can be presented as below**



