# E-Farmers: documentation

Davide Bazzana, Florin Cuconasu, Antonio Grieco, Andrea Nardocci, Marco Settanni

April 2023

# Contents

# Chapter 1

# Introduction

E-Farmers is an e-commerce website that aims to connect local farmers with customers who are interested in buying fresh and locally sourced products. Our platform allows farmers to showcase their products by publishing insertions, while customers can easily browse and purchase these goods. Whether you are a farmer looking to sell your products or a customers seeking high-quality ”farm-to-table” items, E-Farmers has something for everyone. Moreover, if you cannot reach the farmer’s warehouse yourself, you can rely on riders who can ship the products to your home.

In this documentation, we will cover the key features of our application. Our application uses: React JS for the front-end; Django REST Framework and RabbitMQ for the back-end; PostgreSQL and SQLite as databases.

## 1.1 Architecture

Microservices architecture is a way of designing software applications as a collection of small, independent services that communicate with each other through APIs. Each microservice is responsible for a specific, well-defined task. Each microservice can be developed and deployed independently, which means that changes to one service will not necessarily impact the others. Additionally, if one service experiences increased demand, it can be scaled up independently of the other services.

## 1.2 Functionalities

- **Authentication**: user authentication with credentials and OAuth2 (e.g. Google sign-in).
- **Selling Products**: farmers can create insertions to sell their boxes with foodstuffs.
- **Buying Products**: customers can purchase products sold by farmers.
- **Reviews**: each user can submit a review regarding a farmer, after he has bought something from him.
- **Badges**: farmers can earn badges depending on their activities. For instance, if a farmer publishes so many insertions or has been active for a long time, he will obtain a particular badge, visible in this public page.
- **Subscription to Farmer**: customers can subscribe to receive notifications from farmers.
- **Calendar**: a calendar shows all seasonal foodstuffs.
- **Book Products**: customers can book grocery products that are only available during some seasons of the year.
- **Delivery Service**: riders deliver products to customers.

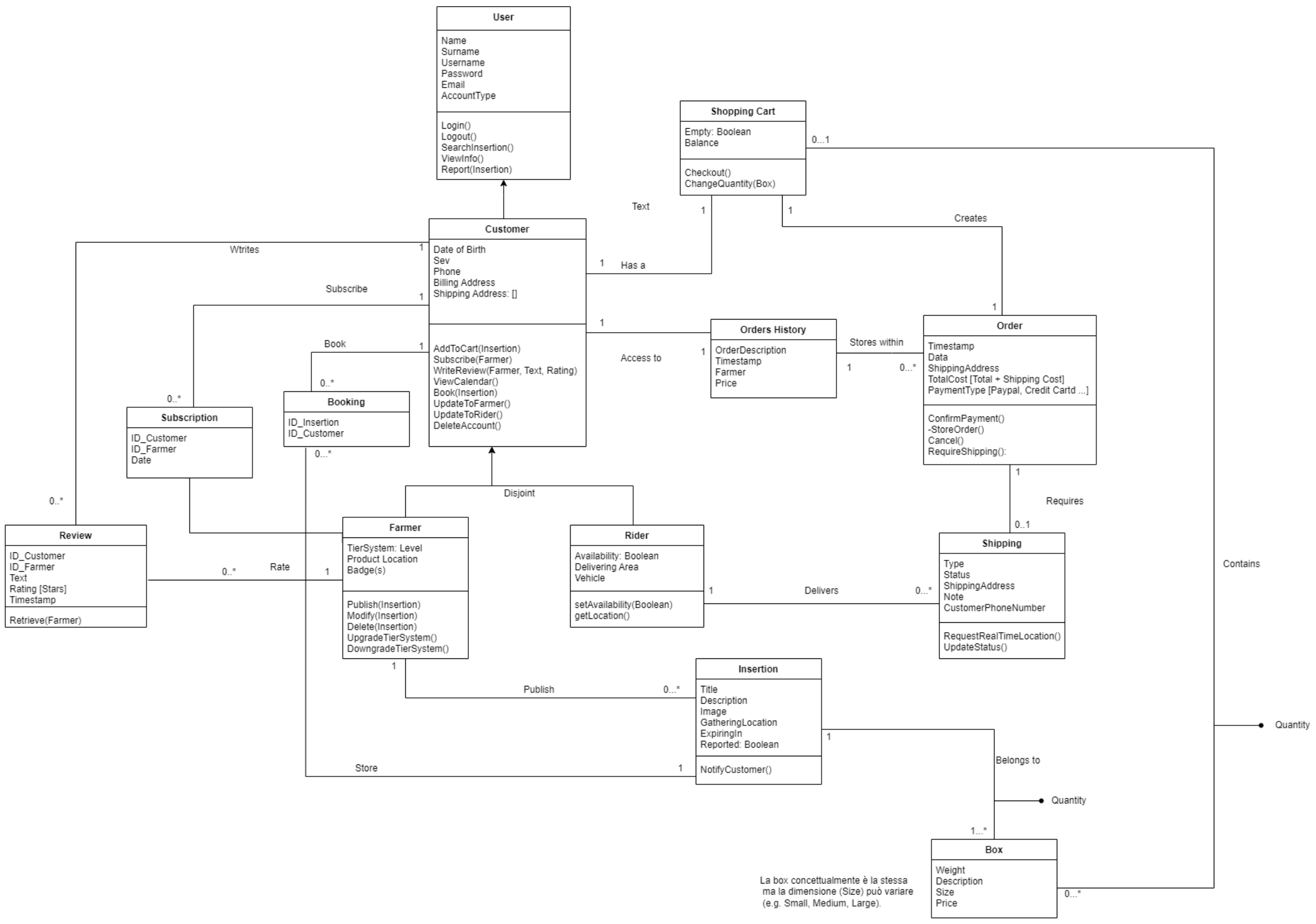


Figure 1.1: UML diagram.

# Chapter 2

# Scrum Framework

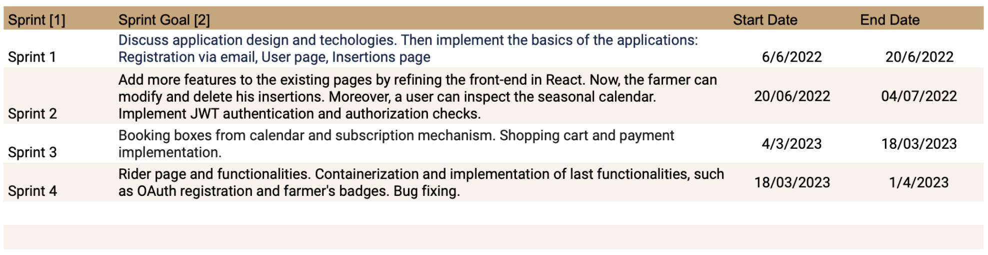
## 2.1 Description

Scrum is an agile framework for managing and completing complex projects. It is commonly used in software development to manage projects and teams, and it is based on iterative and incremental development.

## 2.2 Scrum sprint

In Scrum, the project is divided into small iterations, called sprints, typically lasting two to four weeks. Each sprint includes planning, development, testing, and review, and at the end of each sprint, the team delivers a working software increment that can be reviewed by stakeholders. During each sprint, the team holds several ceremonies:

- Sprint Planning: The team and the Product Owner review the backlog, discuss priorities, and select items to be included in the upcoming sprint.
- Daily Stand-up: A brief daily meeting where the team discusses progress, challenges, and plans for the day.
- Sprint Review: The team presents the working software increment to the stakeholders for feedback and review.
- Sprint Retrospective: The team reflects on the sprint, identifies areas for improvement, and creates a plan to address them.



| Sprint [1] | Sprint Goal [2] | Start Date | End Date |
|---|---|---|---|
| Sprint 1 | Discuss application design and techologies. Then implement the basics of the applications: Registration via email, User page, Insertions page | 6/6/2022 | 20/6/2022 |
| Sprint 2 | Add more features to the existing pages by refining the front-end in React. Now, the farmer can modify and delete his insertions. Moreover, a user can inspect the seasonal calendar. Implement JWT authentication and authorization checks. | 20/06/2022 | 04/07/2022 |
| Sprint 3 | Booking boxes from calendar and subscription mechanism. Shopping cart and payment implementation. | 4/3/2023 | 18/03/2023 |
| Sprint 4 | Rider page and functionalities. Containerization and implementation of last functionalities, such as OAuth registration and farmer's badges. Bug fixing. | 18/03/2023 | 1/4/2023 |

Figure 2.1: SCRUM sprint

## 2.3 Sprint backlog

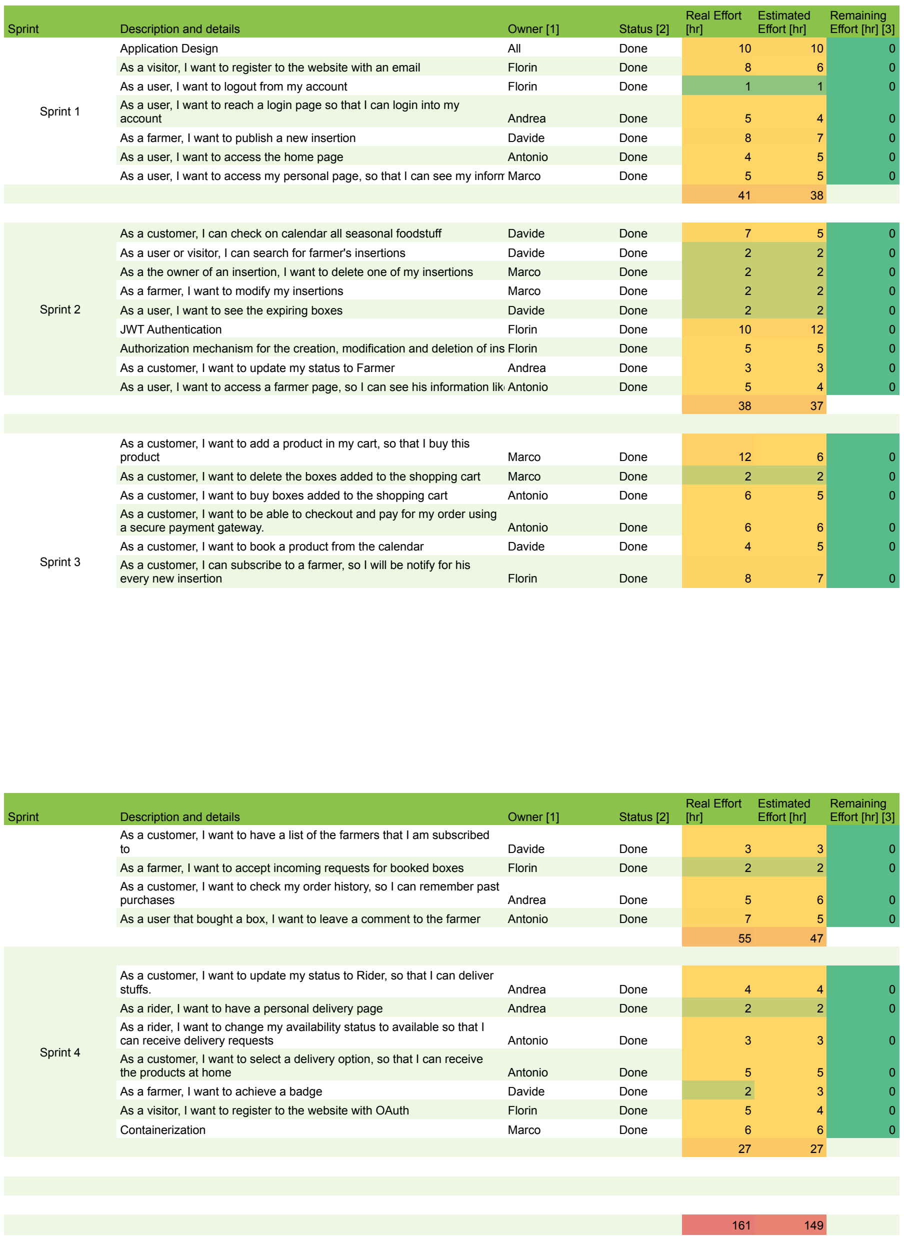
The Sprint Backlog is a list of items from the Product Backlog that the Development Team selects for completion during a Sprint. It contains the set of Product Backlog items that the Development Team has committed to delivering in the current sprint.

It provides a plan for the Development Team to complete the work required to achieve the sprint goal. It is a real-time view of the work that needs to be done during the Sprint and is used by the Development Team to manage their work during the Sprint.

The Sprint Backlog is used as a guide to monitor progress towards the sprint goal and to track any changes that may arise during the Sprint.

The Sprint Backlog is an essential tool for the Development Team to plan, track, and manage their work during a Sprint. It provides transparency into the work being done and helps the Development Team to focus on delivering the highest value features to the customer.

This is our Sprint Backlog :



| Sprint | Description and details | Owner [1] | Status [2] | Real Effort [hr] | Estimated Effort [hr] | Remaining Effort [hr] [3] |
|---|---|---|---|---|---|---|
| Sprint 1 | Application Design | All | Done | 10 | 10 | 0 |
| | As a visitor, I want to register to the website with an email | Florin | Done | 8 | 6 | 0 |
| | As a user, I want to logout from my account | Florin | Done | 1 | 1 | 0 |
| | As a user, I want to reach a login page so that I can login into my account | Andrea | Done | 5 | 4 | 0 |
| | As a farmer, I want to publish a new insertion | Davide | Done | 8 | 7 | 0 |
| | As a user, I want to access the home page | Antonio | Done | 4 | 5 | 0 |
| | As a user, I want to access my personal page, so that I can see my inform | Marco | Done | 5 | 5 | 0 |
| | | | | 41 | 38 | |
| | | | | | | |
| Sprint 2 | As a customer, I can check on calendar all seasonal foodstuff | Davide | Done | 7 | 5 | 0 |
| | As a user or visitor, I can search for farmer's insertions | Davide | Done | 2 | 2 | 0 |
| | As a the owner of an insertion, I want to delete one of my insertions | Marco | Done | 2 | 2 | 0 |
| | As a farmer, I want to modify my insertions | Marco | Done | 2 | 2 | 0 |
| | As a user, I want to see the expiring boxes | Davide | Done | 2 | 2 | 0 |
| | JWT Authentication | Florin | Done | 10 | 12 | 0 |
| | Authorization mechanism for the creation, modification and deletion of ins | Florin | Done | 5 | 5 | 0 |
| | As a customer, I want to update my status to Farmer | Andrea | Done | 3 | 3 | 0 |
| | As a user, I want to access a farmer page, so I can see his information lik | Antonio | Done | 5 | 4 | 0 |
| | | | | 38 | 37 | |
| | | | | | | |
| Sprint 3 | As a customer, I want to add a product in my cart, so that I buy this product | Marco | Done | 12 | 6 | 0 |
| | As a customer, I want to delete the boxes added to the shopping cart | Marco | Done | 2 | 2 | 0 |
| | As a customer, I want to buy boxes added to the shopping cart | Antonio | Done | 6 | 5 | 0 |
| | As a customer, I want to be able to checkout and pay for my order using a secure payment gateway. | Antonio | Done | 6 | 6 | 0 |
| | As a customer, I want to book a product from the calendar | Davide | Done | 4 | 5 | 0 |
| | As a customer, I can subscribe to a farmer, so I will be notify for his every new insertion | Florin | Done | 8 | 7 | 0 |
| | As a customer, I want to have a list of the farmers that I am subscribed to | Davide | Done | 3 | 3 | 0 |
| | As a farmer, I want to accept incoming requests for booked boxes | Florin | Done | 2 | 2 | 0 |
| | As a customer, I want to check my order history, so I can remember past purchases | Andrea | Done | 5 | 6 | 0 |
| | As a user that bought a box, I want to leave a comment to the farmer | Antonio | Done | 7 | 5 | 0 |
| | | | | 55 | 47 | |
| | | | | | | |
| Sprint 4 | As a customer, I want to update my status to Rider, so that I can deliver stuffs. | Andrea | Done | 4 | 4 | 0 |
| | As a rider, I want to have a personal delivery page | Andrea | Done | 2 | 2 | 0 |
| | As a rider, I want to change my availability status to available so that I can receive delivery requests | Antonio | Done | 3 | 3 | 0 |
| | As a customer, I want to select a delivery option, so that I can receive the products at home | Antonio | Done | 5 | 5 | 0 |
| | As a farmer, I want to achieve a badge | Davide | Done | 2 | 3 | 0 |
| | As a visitor, I want to register to the website with OAuth | Florin | Done | 5 | 4 | 0 |
| | Containerization | Marco | Done | 6 | 6 | 0 |
| | | | | 27 | 27 | |
| | | | | | | |
| | | | | 161 | 149 | |

Figure 2.2: Sprint backlog

## 2.4 BurndownChart

A burndown chart is a graphical representation of the progress of a Scrum team during a sprint. It shows the amount of work that remains to be completed versus the amount of time left in the sprint.

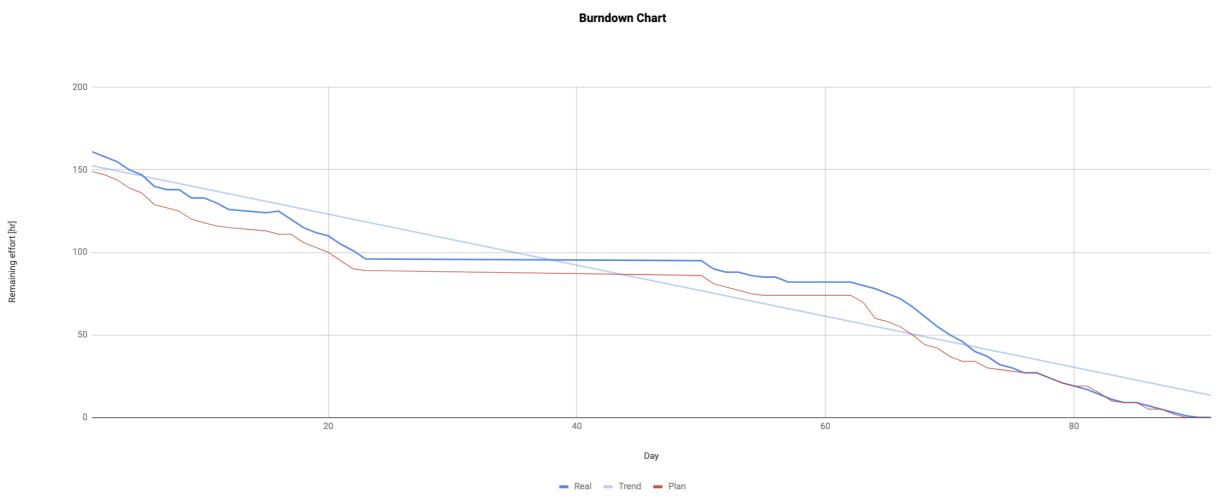
A burndown chart typically has two axes: the X-axis shows the time (usually in days or weeks) and the Y-axis shows the remaining work (usually in story points or hours). The chart starts at the top left corner, where the remaining work is equal to the total work, and it ends at the bottom right corner, where the remaining work is zero.

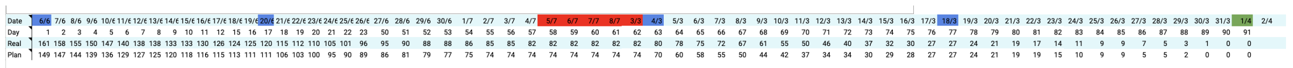
The burndown chart helps the team to track their progress during the sprint and identify if they are on track to complete the work by the end of the sprint. It also helps to identify if there are any issues or obstacles that are causing the team to fall behind schedule.

There are two types of burndown charts:

- Sprint burndown chart: This chart shows the progress of the team during a sprint. It helps the team to track their progress and make adjustments as needed to meet their sprint goal.
- Release burndown chart: This chart shows the progress of the team over multiple sprints towards a release goal. It helps the team to track their progress and make adjustments as needed to meet their release goal.

Burndown charts are a powerful tool for Scrum teams to visualize their progress and improve their productivity. They are also a useful communication tool for stakeholders to understand the progress of the project and make informed decisions based on that progress.





| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Date | 6/6 | 7/6 | 8/6 | 9/6 | 10/6 | 11/6 | 12/6 | 13/6 | 14/6 | 15/6 | 16/6 | 17/6 | 18/6 | 19/6 | 20/6 | 21/6 | 22/6 | 23/6 | 24/6 | 25/6 | 26/6 |
| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| Real | 161 | 158 | 155 | 150 | 147 | 140 | 138 | 138 | 133 | 133 | 130 | 126 | 124 | 125 | 120 | 115 | 112 | 110 | 105 | 101 | 96 |
| Plan | 149 | 147 | 144 | 139 | 136 | 129 | 127 | 125 | 120 | 118 | 116 | 115 | 113 | 111 | 111 | 106 | 103 | 100 | 95 | 90 | 89 |

| | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Date | 27/6 | 28/6 | 29/6 | 30/6 | 1/7 | 2/7 | 3/7 | 4/7 | 5/7 | 6/7 | 7/7 | 8/7 | 3/3 | 4/3 | 5/3 | 6/3 | 7/3 | 8/3 | 9/3 | 10/3 | 11/3 |
| Day | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| Real | 95 | 90 | 88 | 88 | 86 | 85 | 85 | 82 | 82 | 82 | 82 | 82 | 82 | 80 | 78 | 75 | 72 | 67 | 61 | 55 | 50 |
| Plan | 86 | 81 | 79 | 77 | 75 | 74 | 74 | 74 | 74 | 74 | 74 | 74 | 74 | 70 | 60 | 58 | 55 | 50 | 44 | 42 | 37 |

| | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Date | 12/3 | 13/3 | 14/3 | 15/3 | 16/3 | 17/3 | 18/3 | 19/3 | 20/3 | 21/3 | 22/3 | 23/3 | 24/3 | 25/3 | 26/3 | 27/3 | 28/3 | 29/3 | 30/3 | 31/3 | 1/4 | 2/4 |
| Day | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | |
| Real | 46 | 40 | 37 | 32 | 30 | 27 | 27 | 24 | 21 | 19 | 17 | 14 | 11 | 9 | 9 | 7 | 5 | 3 | 1 | 0 | 0 | |
| Plan | 34 | 34 | 30 | 29 | 28 | 27 | 27 | 24 | 21 | 19 | 19 | 15 | 10 | 9 | 9 | 5 | 5 | 2 | 0 | 0 | 0 | |

Figure 2.3: Burndown Chart

# Chapter 3

# Analysis

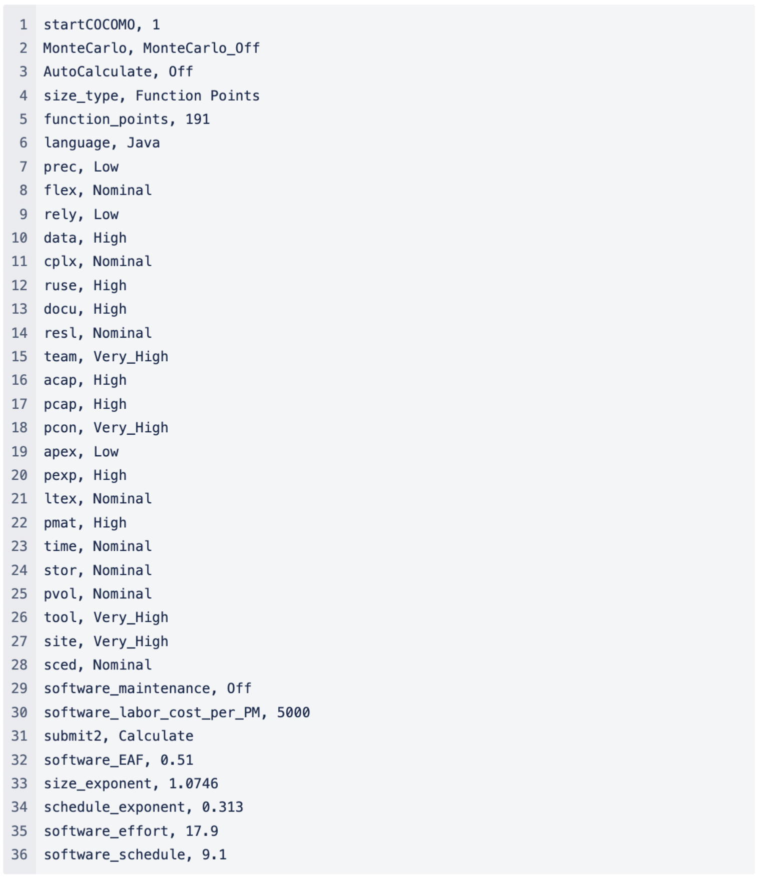
## 3.1 Cocomo II Model

The Cocomo II (Constructive Cost Model) is a software cost estimation model. Cocomo II is based on the assumption that there is a relationship between the size of a software system and the effort required to develop it. Cocomo II takes into account three different levels of software complexity: Basic, Intermediate, and Advanced. Each level is associated with a set of cost drivers, which are factors that can influence the cost and effort required to develop a software system. These cost drivers include factors such as the size of the development team, the experience level of the developers, the complexity of the system architecture, and the development environment.

Cocomo II also includes a set of equations that can be used to estimate the effort and cost required to develop a software system based on its size and complexity. The model provides estimates for various aspects of software development, such as design, coding, and testing.

One of the advantages of Cocomo II is its flexibility, as it can be customized to fit different types of software development projects. However, like any software cost estimation model, Cocomo II has its limitations, and its accuracy depends on the accuracy of the input data and assumptions used to create the estimates.

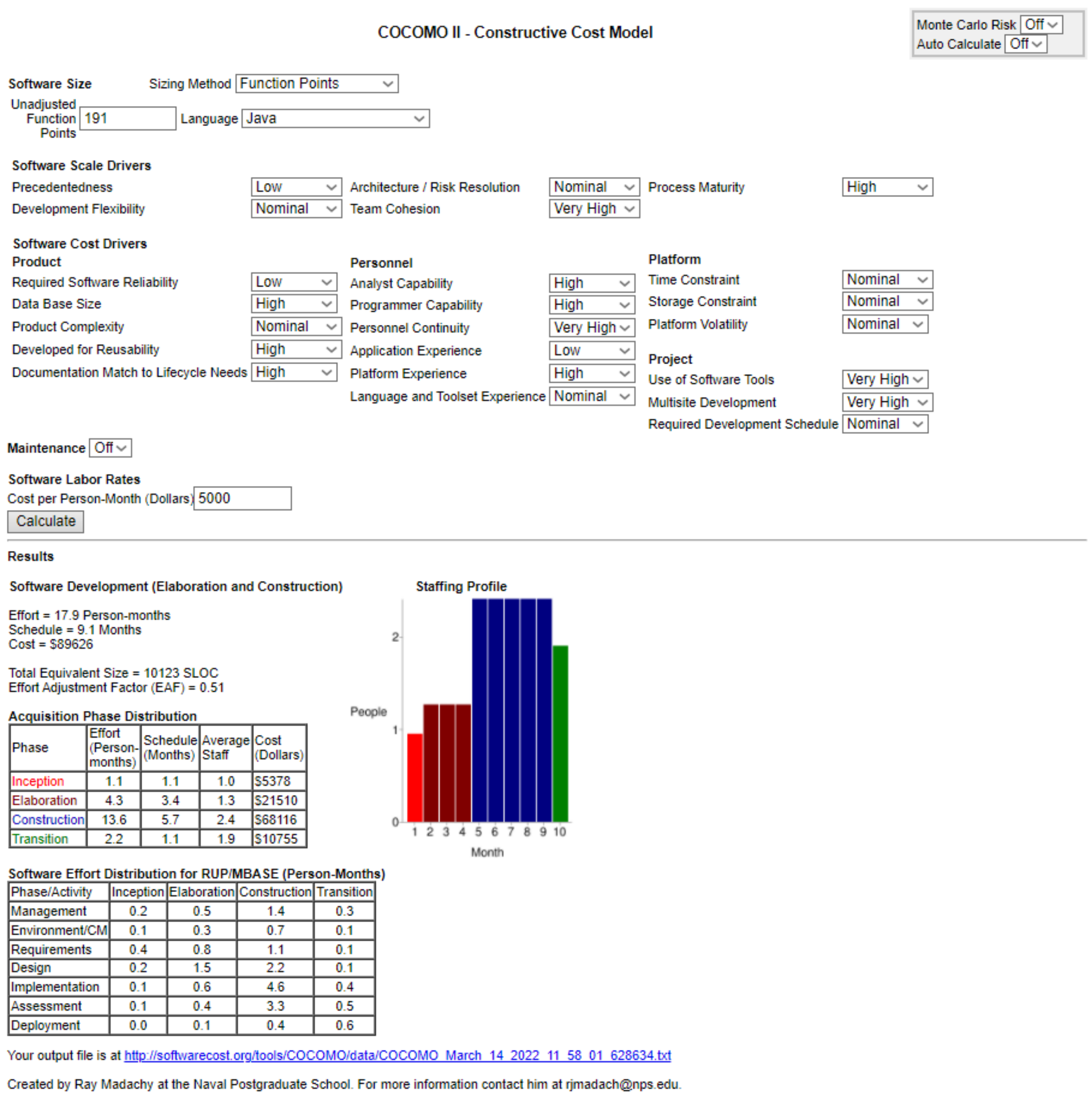
Here there is our parameter used in Cocomo II model.



```
startCOCOMO, 1
MonteCarlo, MonteCarlo_Off
AutoCalculate, Off
size_type, Function Points
function_points, 191
language, Java
prec, Low
flex, Nominal
rely, Low
data, High
cplx, Nominal
ruse, High
docu, High
resl, Nominal
team, Very_High
acap, High
pcap, High
pcon, Very_High
apex, Low
pexp, High
ltex, Nominal
pmat, High
time, Nominal
stor, Nominal
pvol, Nominal
tool, Very_High
site, Very_High
sced, Nominal
software_maintenance, Off
software_labor_cost_per_PM, 5000
submit2, Calculate
software_EAF, 0.51
size_exponent, 1.0746
schedule_exponent, 0.313
software_effort, 17.9
software_schedule, 9.1
```

Figure 3.1: Cocomo parameters

Then this is our model.



COCOMO II - Constructive Cost Model

Monte Carlo Risk: Off
Auto Calculate: Off

**Software Size** Sizing Method: Function Points
Unadjusted Function Points: 191 Language: Java

**Software Scale Drivers**

| | | | | | |
|---|---|---|---|---|---|
| Precedentedness | Low | Architecture / Risk Resolution | Nominal | Process Maturity | High |
| Development Flexibility | Nominal | Team Cohesion | Very High | | |

**Software Cost Drivers**

**Product**

| | |
|---|---|
| Required Software Reliability | Low |
| Data Base Size | High |
| Product Complexity | Nominal |
| Developed for Reusability | High |
| Documentation Match to Lifecycle Needs | High |

**Personnel**

| | |
|---|---|
| Analyst Capability | High |
| Programmer Capability | High |
| Personnel Continuity | Very High |
| Application Experience | Low |
| Platform Experience | High |
| Language and Toolset Experience | Nominal |

**Platform**

| | |
|---|---|
| Time Constraint | Nominal |
| Storage Constraint | Nominal |
| Platform Volatility | Nominal |

**Project**

| | |
|---|---|
| Use of Software Tools | Very High |
| Multisite Development | Very High |
| Required Development Schedule | Nominal |

**Maintenance** Off

**Software Labor Rates**
Cost per Person-Month (Dollars) 5000

Calculate

**Results**

**Software Development (Elaboration and Construction)**

Effort = 17.9 Person-months
Schedule = 9.1 Months
Cost = $89626

Total Equivalent Size = 10123 SLOC
Effort Adjustment Factor (EAF) = 0.51

**Acquisition Phase Distribution**

| Phase | Effort (Person-months) | Schedule (Months) | Average Staff | Cost (Dollars) |
|---|---|---|---|---|
| Inception | 1.1 | 1.1 | 1.0 | $5378 |
| Elaboration | 4.3 | 3.4 | 1.3 | $21510 |
| Construction | 13.6 | 5.7 | 2.4 | $68116 |
| Transition | 2.2 | 1.1 | 1.9 | $10755 |

**Staffing Profile**

**Software Effort Distribution for RUP/MBASE (Person-Months)**

| Phase/Activity | Inception | Elaboration | Construction | Transition |
|---|---|---|---|---|
| Management | 0.2 | 0.5 | 1.4 | 0.3 |
| Environment/CM | 0.1 | 0.3 | 0.7 | 0.1 |
| Requirements | 0.4 | 0.8 | 1.1 | 0.1 |
| Design | 0.2 | 1.5 | 2.2 | 0.1 |
| Implementation | 0.1 | 0.6 | 4.6 | 0.4 |
| Assessment | 0.1 | 0.4 | 3.3 | 0.5 |
| Deployment | 0.0 | 0.1 | 0.4 | 0.6 |

Your output file is at http://softwarecost.org/tools/COCOMO/data/COCOMO_March_14_2022_11_58_01_628634.txt

Created by Ray Madachy at the Naval Postgraduate School. For more information contact him at rjmadach@nps.edu.

Figure 3.2: Cocomo response

## 3.2 Function Point

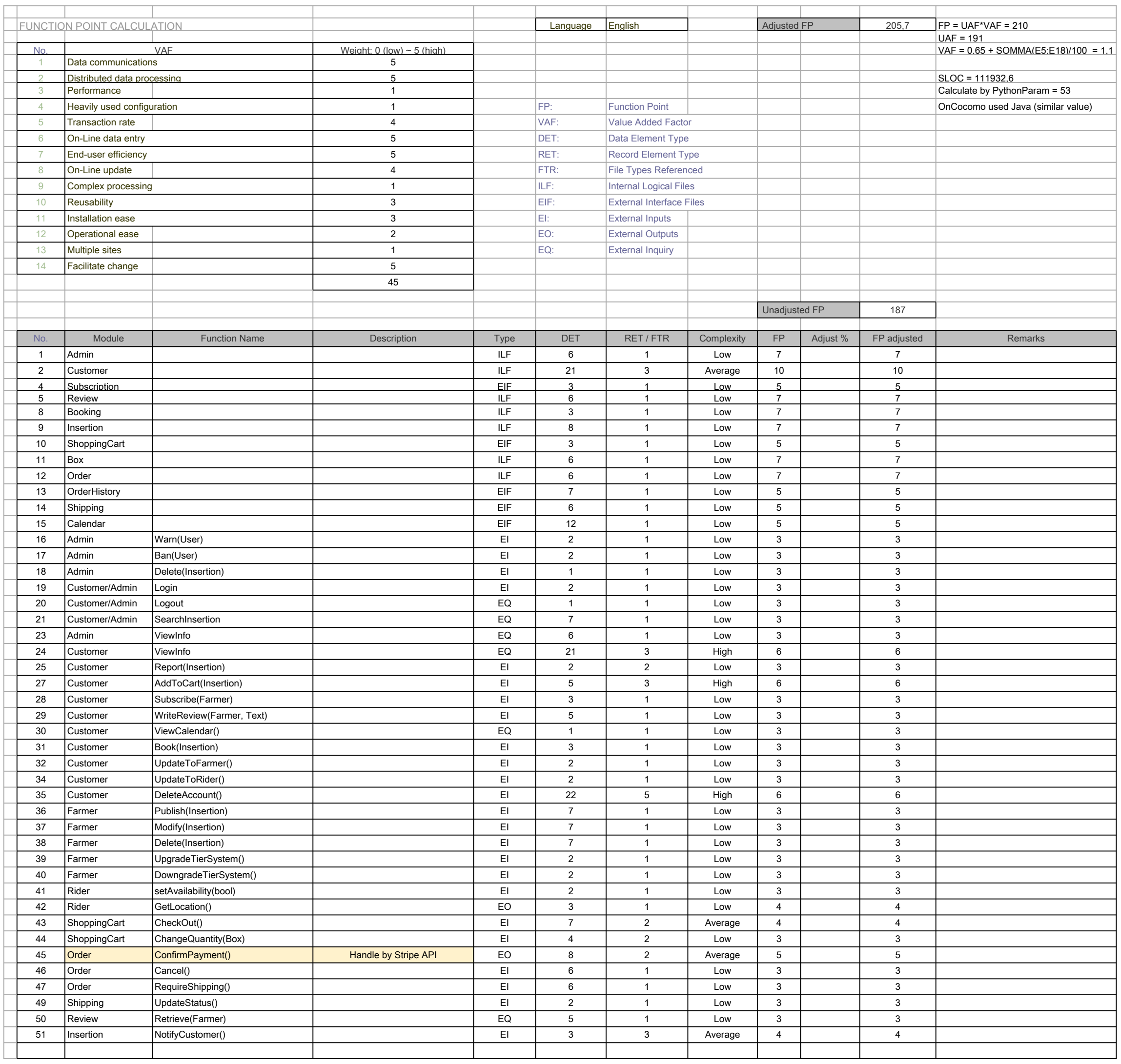
To estimating the size and complexity of software systems we use Function Points.

Function points are a software metric used to measure the size and complexity of a software system.They were introduced as a way to estimate the effort and cost of developing software applications.

Function points are based on the idea that software can be broken down into a series of functions or features, each of which contributes to the overall value of the system. These functions are categorized into different types, such as input, output, and processing functions, and assigned weights based on their complexity.

By calculating the total number of function points in a software system, developers and project managers can estimate the amount of time and resources that will be required to develop, test, and maintain the system. Function points can also be used to track the progress of a project and measure its overall quality.

So there there is our estimation about all the models and function available in our software application.



FUNCTION POINT CALCULATION

| | |
|---|---|
| Language | English |
| Adjusted FP | 205,7 |
| Unadjusted FP | 187 |

FP = UAF*VAF = 210
UAF = 191
VAF = 0,65 + SOMMA(E5:E18)/100 = 1.1

SLOC = 111932,6
Calculate by PythonParam = 53
OnCocomo used Java (similar value)

| No. | VAF | Weight: 0 (low) ~ 5 (high) |
|---|---|---|
| 1 | Data communications | 5 |
| 2 | Distributed data processing | 5 |
| 3 | Performance | 1 |
| 4 | Heavily used configuration | 1 |
| 5 | Transaction rate | 4 |
| 6 | On-Line data entry | 5 |
| 7 | End-user efficiency | 5 |
| 8 | On-Line update | 4 |
| 9 | Complex processing | 1 |
| 10 | Reusability | 3 |
| 11 | Installation ease | 3 |
| 12 | Operational ease | 2 |
| 13 | Multiple sites | 1 |
| 14 | Facilitate change | 5 |
| | | 45 |

| | |
|---|---|
| FP: | Function Point |
| VAF: | Value Added Factor |
| DET: | Data Element Type |
| RET: | Record Element Type |
| FTR: | File Types Referenced |
| ILF: | Internal Logical Files |
| EIF: | External Interface Files |
| EI: | External Inputs |
| EO: | External Outputs |
| EQ: | External Inquiry |

| No. | Module | Function Name | Description | Type | DET | RET / FTR | Complexity | FP | Adjust % | FP adjusted | Remarks |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Admin | | | ILF | 6 | 1 | Low | 7 | | 7 | |
| 2 | Customer | | | ILF | 21 | 3 | Average | 10 | | 10 | |
| 4 | Subscription | | | EIF | 3 | 1 | Low | 5 | | 5 | |
| 5 | Review | | | ILF | 6 | 1 | Low | 7 | | 7 | |
| 8 | Booking | | | ILF | 3 | 1 | Low | 7 | | 7 | |
| 9 | Insertion | | | ILF | 8 | 1 | Low | 7 | | 7 | |
| 10 | ShoppingCart | | | EIF | 3 | 1 | Low | 5 | | 5 | |
| 11 | Box | | | ILF | 6 | 1 | Low | 7 | | 7 | |
| 12 | Order | | | ILF | 6 | 1 | Low | 7 | | 7 | |
| 13 | OrderHistory | | | EIF | 7 | 1 | Low | 5 | | 5 | |
| 14 | Shipping | | | EIF | 6 | 1 | Low | 5 | | 5 | |
| 15 | Calendar | | | EIF | 12 | 1 | Low | 5 | | 5 | |
| 16 | Admin | Warn(User) | | EI | 2 | 1 | Low | 3 | | 3 | |
| 17 | Admin | Ban(User) | | EI | 2 | 1 | Low | 3 | | 3 | |
| 18 | Admin | Delete(Insertion) | | EI | 1 | 1 | Low | 3 | | 3 | |
| 19 | Customer/Admin | Login | | EI | 2 | 1 | Low | 3 | | 3 | |
| 20 | Customer/Admin | Logout | | EQ | 1 | 1 | Low | 3 | | 3 | |
| 21 | Customer/Admin | SearchInsertion | | EQ | 7 | 1 | Low | 3 | | 3 | |
| 23 | Admin | ViewInfo | | EQ | 6 | 1 | Low | 3 | | 3 | |
| 24 | Customer | ViewInfo | | EQ | 21 | 3 | High | 6 | | 6 | |
| 25 | Customer | Report(Insertion) | | EI | 2 | 2 | Low | 3 | | 3 | |
| 27 | Customer | AddToCart(Insertion) | | EI | 5 | 3 | High | 6 | | 6 | |
| 28 | Customer | Subscribe(Farmer) | | EI | 3 | 1 | Low | 3 | | 3 | |
| 29 | Customer | WriteReview(Farmer, Text) | | EI | 5 | 1 | Low | 3 | | 3 | |
| 30 | Customer | ViewCalendar() | | EQ | 1 | 1 | Low | 3 | | 3 | |
| 31 | Customer | Book(Insertion) | | EI | 3 | 1 | Low | 3 | | 3 | |
| 32 | Customer | UpdateToFarmer() | | EI | 2 | 1 | Low | 3 | | 3 | |
| 34 | Customer | UpdateToRider() | | EI | 2 | 1 | Low | 3 | | 3 | |
| 35 | Customer | DeleteAccount() | | EI | 22 | 5 | High | 6 | | 6 | |
| 36 | Farmer | Publish(Insertion) | | EI | 7 | 1 | Low | 3 | | 3 | |
| 37 | Farmer | Modify(Insertion) | | EI | 7 | 1 | Low | 3 | | 3 | |
| 38 | Farmer | Delete(Insertion) | | EI | 7 | 1 | Low | 3 | | 3 | |
| 39 | Farmer | UpgradeTierSystem() | | EI | 2 | 1 | Low | 3 | | 3 | |
| 40 | Farmer | DowngradeTierSystem() | | EI | 2 | 1 | Low | 3 | | 3 | |
| 41 | Rider | setAvailability(bool) | | EI | 2 | 1 | Low | 3 | | 3 | |
| 42 | Rider | GetLocation() | | EO | 3 | 1 | Low | 4 | | 4 | |
| 43 | ShoppingCart | CheckOut() | | EI | 7 | 2 | Average | 4 | | 4 | |
| 44 | ShoppingCart | ChangeQuantity(Box) | | EI | 4 | 2 | Low | 3 | | 3 | |
| 45 | Order | ConfirmPayment() | Handle by Stripe API | EO | 8 | 2 | Average | 5 | | 5 | |
| 46 | Order | Cancel() | | EI | 6 | 1 | Low | 3 | | 3 | |
| 47 | Order | RequireShipping() | | EI | 6 | 1 | Low | 3 | | 3 | |
| 49 | Shipping | UpdateStatus() | | EI | 2 | 1 | Low | 3 | | 3 | |
| 50 | Review | Retrieve(Farmer) | | EQ | 5 | 1 | Low | 3 | | 3 | |
| 51 | Insertion | NotifyCustomer() | | EI | 3 | 3 | Average | 4 | | 4 | |

Figure 3.3: Function Point

# Chapter 4

# Users microservice

## 4.1 Description

The microservice contains endpoints for user login, obtaining and refreshing JSON Web Tokens (JWT) for authenticated users, and revoking tokens on user logout. The endpoints for JWT generation and refresh utilize Django's built-in TokenObtainPairView and CustomTokenRefreshView classes, respectively. Additionally, there is an OAuthTokenObtainPairView for obtaining JWTs using Google authentication.

The microservice also includes several endpoints for user management, including registering new users, updating user accounts, retrieving user information, and adding reviews for farmers. These endpoints are grouped together under a single UsersView class and utilize various HTTP methods, such as GET, POST, and PATCH. There are also endpoints specifically for managing farmer and rider profiles, including updating the number of insertions for farmers and changing the status of riders.

Overall, this microservice appears to be a key component of a larger web application that allows for user authentication and authorization, as well as user-related operations such as user registration, profile management, and review posting.

## 4.2 JWT Authentication

JWT stands for *JSON Web Token*, which is a standard for securely transmitting information between parties as a JSON object. In the context of authentication, JWT is used to authenticate and authorize users, and it allows clients to obtain a token that can be used to access protected resources.

In our implementation, we use the third-party Django library *rest_framework_simplejwt*, as it provides a simple and powerful implementation of JWT authentication.
Here is how it works:

1. A user logs in using their username and password. The Django view authenticates the user and generates a JWT token.
2. The JWT token is then returned to the client as a response to the login request.
3. The client can then use this JWT token to access protected resources by including it in the Authorization header of subsequent requests (this is done automatically by the front-end).
4. When the server receives a request with a JWT token, it verifies the token's signature and decodes the token to extract the user's information.
5. If the token is valid and has not expired, the server allows the request to proceed and grants the user access to the protected resource.
6. If the token is invalid or has expired, the server denies the request and returns an error response.

In particular, there are generated two types of tokens the *access* token and the *refresh* token.

The *access* token is a short-lived token that is used to authenticate a user for a limited time period, usually for a few minutes (5 minutes in our case). This token contains the user's identity and permissions encoded within it. When a user logs in or authenticates for the first time, an access token is issued to them, and this token is sent along with every subsequent request to the server to verify their identity and permissions.

The *refresh* token, on the other hand, is a long-lived token that is used to obtain a new access token when the current one expires. When the *access* token expires, the user can send the refresh token to the server to obtain a new access token without having to log in again.

By using two types of tokens, the JWT authentication system can achieve better security and usability. Short-lived access tokens reduce the risk of token hijacking or misuse, while long-lived refresh tokens provide a seamless and user-friendly authentication experience. Additionally, the use of refresh tokens also helps reduce the load on the server, as the user can obtain a new access token without having to make a new login request.

In addition, the library provides features such as token refreshing, blacklisting, and custom claims that can be used to customize the authentication process.

### 4.2.1 Google OAuth

The system also allows to authenticated using a Google account. An OAuth is an open authorization service that allows websites or applications to share user information with other websites without being given a user's password. Users can sign in to multiple sites using the same account without creating other credentials. Therefore, this feature allows to register to our website without an explicit users' password.

In this case, the user has his own *access* token different from the JWT token, which is sent to the user once he has been authenticated to Google. The front-end sends that token to the back-end, which uses it to fetch user's information from the Google account. Now, it is verified whether there exists an account with that email in our system: if there is then is simply generated a JWT token using the *rest_framework_simplejwt* library; otherwise, it is first created an account to our website using a fake password and then it is genereted the JWT token.

The advantage of this method it that, in the end, the user will always use the JWT token, as if he accessed through credentials, so we do not need to handle particular cases.

## 4.3 API

### 4.3.1 Resource: Token

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| POST | token/ | JSON: { “email”: “xxx” “password”: “xxx” } | Generates the *access* and the *refresh* tokens associated to user with those email and password. |
| POST | token/refresh/ | JSON: { “refresh”: “xxx” } | It takes as input a *refresh* token and outputs the two new *access* and *refresh* tokens. It also returns a new *refresh* token since the first one is backlisted. |
| POST | token/verify/ | JSON: { “user_id”: “xxx” } | In this case the token is passed in the Authorization header. This API checks whether the user id passed as input is the one associated with the token. If it is not the case, the access is denied. |
| POST | oauth/token/ | JSON: { “access_token”: “xxx” } | The *access* token in input is the token generated by the Google OAuth service. The API then returns the usual *access* and *refresh* JWT tokens of our system. |
| POST | logout/blacklist/ | JSON: { “refresh_token”: “xxx” } | The *refresh* token is blacklisted. This will make sure that the refresh token cannot be used again to generate a new token. |

### 4.3.2 Resource: User

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| POST | login/ | JSON: {<br>“email”: “xxx”<br>“password”: “xxx”<br>} | It checks if there exist a user with that email and password in the DB. If it is not the case an exception is thrown. |
| GET | users/ | | Return all users that are stored in the database. |
| POST | users/ | JSON: {<br>”name”:”xxx”,<br>”email”:”xxx”<br>”password”:”xxx”<br>”account_type”:”xxx”<br>} | This creates a new instance of the user only when you satisfied all the requirements needed in order to register to this application. If you don’t respect some of this requirements: like unique email and so on, u can’t able to register to the site. |
| GET | users/<int:user_id>/ | JSON : {<br>”user_id” : ”xxx”<br>} | Get the specific user which match the user_id passed through the request. If the user has some special extra information, that comes from special kind of account, the request retrieve either the user information and the user extra info. |
| GET | users/<int:user_id>/name/ | | Get the name of the user with that id. |
| PATCH | users/<int:user_id>/ | JSON: {<br>”account_type”:”xxx”,<br>} | This request modify the field of the user in the db by updating the type of the users account. This is a special operation that is handled if cause some error by administrator. |
| PATCH | users/<int:user_id>/changes/ | JSON: {<br>”name”:”xxx”,<br>”email”:”xxx”<br>”account_type”:”xxx”<br>”billing_address”:”xxx”<br>”shipping_address”:”xxx”<br>”phone_number”:”xxx”<br>”bio”:”xxx”<br>”billing_address”:”xxx”<br>} | Updates the user’s account information, such as name, billing, shipping address and so other usefully information. It doesn’t require that all field are filled up with some context. |
| POST | users/<int:user_id>/<int:type>/ | Farmer<br>JSON: {<br>”bio”:”xxx”,<br>”farm_location”:”xxx”<br>}<br>Rider<br>JSON: {<br>”available”:”xxx”,<br>”bio”:”xxx”<br>} | Updates user extra information based on type, by adding some new field in the database. In this case gives two additional information for the specific case of rider and farmer. |

### 4.3.3 Resource: User.Farmer

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| GET | farmers/<int:user_id>/ | | Get the instance of farmer which has the sames user_id, then retrieve all the extra information about it. |
| PATCH | farmers/<int:user_id>/ id | JSON: { "farmer_id":"xxx" } | Once a farmer publish his insertion, this request is called in order to increase the global counter of published insertion for the specific farmer |

### 4.3.4 Resource: User.Rider

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| GET | riders/ | | Search through the database to find the first raider available in order to assign him to a new delivery. |
| GET | riders/<int:user_id>/ | | Get the instance of rider which has the same user_id, then retrieve all the extra information about it. |
| PATCH | riders/<int:user_id>/ | JSON: { "available":"xxx" } | Special operation for the rider. It change only the status of the rider by putting him available or not available,which means being able to handle a delivery and not being able to handle a delivery, respectively |

### 4.3.5 Resource: Review

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| GET | review/<int:user_id>/ | | Return the last review written for the specific farmer and display it when u visit the Farmer profile page. |
| POST | review/<int:user_id>/ | JSON: { "rating":"xxx", "comment":"xxx", "farmer_user":"xxx", } | User can create a review about the order. This review is applied both to the shipping and to the quality of the product selled by the farmer. Once the user create a review it can be available on the specific farmer's page. |

# Chapter 5

# Insertion microservice

## 5.1 Description

This microservice implements all the functionalities related to insertions. Creation, deletion and update of insertions and boxes are all treated here. The booking of products is also treated by this microservice.

## 5.2 API

### 5.2.1 Resource:Insertion

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| GET | insertions/ | JSON: {"search":"xxx", "expiring":"xxx", "farmer":"xxx"} | List all or a subset of the insertions. |
| POST | insertions/ | JSON: {"title":"xxx", "description":"xxx", "expiration_date":"xxx", "gathering_location":"xxx", "image":"xxx", "reported":"xxx", "farmer":"xxx", "related_name":"xxx", "private":"xxx", "request":"xxx"} | Creates a new insertion. |
| GET | insertions/<int:id>/ | | Retrieve a specific insertion. If the insertion has expired, it is deleted from the database. |
| GET | insertions/<int:id>/image/ | | Retrieve the image of the specified insertion. |
| PUT | insertions/<int:id>/ | JSON: {"title":"xxx", "description":"xxx", "gathering_location":"xxx", "image":"xxx", "farmer":"xxx", "related_name":"xxx", "private":"xxx", "request":"xxx"} | Create new insertion |
| DELETE | insertions/<int:id>/ | | Delete the specified insertion. |

### 5.2.2 Resoure: Box

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| GET | insertions/<int:id>/boxes/ | | Retrieve all the boxes related to an insertion. |
| POST | insertions/<int:id>/boxes/ | JSON: {"insertion":"xxx", "weight":"xxx", "validators":"xxx", "size":"xxx", "price":"xxx", "number_of_available_boxes":"xxx", } | Create a new box for an insertion. |
| PATCH | boxes/<int:box_id>/decrease | | Decrease the number of boxes of a given type. |

### 5.2.3 Resoure: Booking

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| GET | booking/<int:request_id>/ | | Returns the specified request |
| PUT | booking/<int:request_id>/ | JSON: {"insertion":"xxx"} | As a farmer, accept a request by publishing a private insertion. |
| GET | booking/requests/<int:user_id>/ | | Returns the list of a user's requests (booked products). |
| GET | booking/inbox/<int:farmer_id>/ | | Returns the list of a farmer's requests received by users. |
| POST | booking/ | JSON: {"user":"xxx", "farmer":"xxx", "title":"xxx", "comment":"xxx", "weight":"xxx", "deadline":"xxx", "insertion":"xxx"} | Creates a new request. |
| DELETE | booking/<int:id>/ | | Deletes the specified request. |

# Chapter 6

# Subscriptions microservice

## 6.1 Description

This service implements the subscription functionality: a customer can subscribe to a farmer in order to be notified when that farmer publishes new insertions.

## 6.2 API

### 6.2.1 Resource: Queue

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| PUT | customer/<int:user_id>/ | JSON: {"farmer_id":"xxx"} | Creates a new binding between the customer's queue and the farmer's exchange. If the queue or the exchange do not exist, they are created. Each customer has its own queue. |
| PATCH | customer/<int:user_id>/ | JSON: {"farmer_id":"xxx"} | Deletes a binding between a customer's queue and a farmer's exchange. |
| GET | customer/<int:user_id>/ | | Reads all the messages in the queue. |
| DELETE | customer/<int:user_id>/ | | Deletes the queue. |

### 6.2.2 Resource: Exchange

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| POST | farmer/<int:farmer_id>/ | JSON: {"message":"XXX"} | Passes a message to the exchange relative to the farmer. If the exchange does not exists, it is created. The exchange will deliver the message to all the queues it is bound to. |
| DELETE | farmer/<int:farmer_id>/ | | Deletes the exchange. |

# Chapter 7

# Payments & Orders microservice

## 7.1 Description

The microservice is responsible for handling requests related to order and payment management in this application. The payment are handled by the API given by Stripe, a payment processing company.

It provides a set of RESTful endpoints for processing payments, saving stripe information, getting specific orders, updating orders, updating the status of riders, and retrieving orders by email. The NewView class is used as the view for handling incoming requests, and it has various methods associated with different HTTP verbs and actions, including test_payment for processing payments, in particulary save_stripe_info for saving stripe information that is handled by STRIPE API, and getSpecificOrder for retrieving specific orders.

As a microservice, it provides an independent and modular solution for order and payment management, which can be easily scaled and updated without affecting other components of the application. It encapsulates the order and payment logic, providing a clear separation of concerns between different parts of the system. This allows for better maintainability, security, and flexibility in managing orders and payments within the application. Thanks to stripe it also provides robust security features, including data encryption and tokenization, to help protect sensitive payment information.

## 7.2 API

### 7.2.1 Resource: Orders

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| GET | getSpecificOrder/<str:payment_method_id>/ | | Reeturn the information about the specific order with the same unique payment_method_id. |
| GET | getSpecificOrderByRider/<int:rider_id>/ | | Get the instance of the order that is handled by the rider which has the same value of the parameter rider_id. |
| PATCH | update-order/ | JSON: { "payment_method_id": "riderId":"xxx" , } | Update the status of the order when is in the process of delivering. The shipping can be handled by a rider or the costumer can choose to pickup at the warehouse. |
| PATCH | update-status-rider/ | JSON: { "riderUserId":"xxx" } | Special operation of the rider and change the status of a specific order,which has the same rideerUserId, once the delivery is completed. It can be generated by the rider itself directly from it's account. |
| GET | get-orders/<str:email>/ | | Retrieve all the orders by passing the email address of the users as a parameter. |

### 7.2.2 Resource: Payments

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| POST | save-stripe-info/ | JSON: { "email":"xxx", "payment_method_id": "price":"xxx", "box_names":"xxx", "farmer":"xxx" } | Creates an Intent of the payment with the total price of the items in the cart and the user, then the intent is handled in order to create the payment. Futhermore it create and save the information about the order early created. |

# Chapter 8

# Shopping Cart microservice

## 8.1 Description

The Shopping Cart microservice is responsible for handling the requests that regards the shopping cart management, is linked to the checkout process and take care of checking the information passed from the Insertions microservice.
Inside the microservice two main models have been defined to manage the requests and to define the single, definitive source of information about the data: the Cart and the CartItem. Each of these model maps to a single database table and the attributes defined inside the model outline the column in the table.
The microservice provides a fixed amount of RESTful endpoints for the following functionalities:

- Create the shopping cart
- Retrieve the cart
- Delete the cart
- List the cart items
- Add the insertion boxes to the shopping cart
- Remove the insertion boxes
- Checkout the shopping cart

The main concept of the Cart model is that each shopping cart is linked to a user and both to a farmer, so it's possible to only add products to the shopping cart that have been created by a single farmer. For each product that doesn't belong to the same farmer, upon request made to the user, the shopping cart is deleted and a new one is created with the products of the new farmer.

## 8.2 API

Here it's possible to see a review of all the APIs built for the Shopping Cart microservice. These APIs cover both the Cart and the CartItem logic.

### 8.2.1 Resource: Cart

| Method | Endpoint | Request body | Description |
|---|---|---|---|
| GET | users/<int:user_id>/cart/ | The user id is extracted from the endpoint | Obtains the shopping cart given the user ID |
| POST | users/<int:user_id>/cart/ | JSON:{ "user": user_id "current_farmer": farmer_id } | Create a new shopping cart that is linked to a user by its user id and a farmer by its farmer id |
| DELETE | users/<int:user_id>/cart/ | The user id is extracted from the endpoint | Delete the shopping cart linked to the user by its user id |

### 8.2.2 Resource: CartItem

| | | | |
|---|---|---|---|
| GET | users/<int:user_id>/cart/items/ | The user id is extracted from the endpoint | Obtains the boxes saved in the shopping cart given the user ID |
| PUT | users/<int:user_id>/cart/items/ | JSON:{<br>"cart": cart_id<br>"box_id": box_id<br>"name": boxName<br>"size": boxSize<br>"weight": boxWeight<br>"price": boxPrice<br>} | Create a new CartItem based on the info of the insertion box and it's added to the shopping cart |
| DELETE | users/<int:user_id>/cart/items/ | JSON:{<br>"box_id": box_id<br>} | Delete the CartItem linked to the box id passed in the request |

# Chapter 9

# WebApp Tour through mockups

## 9.1 Home Page

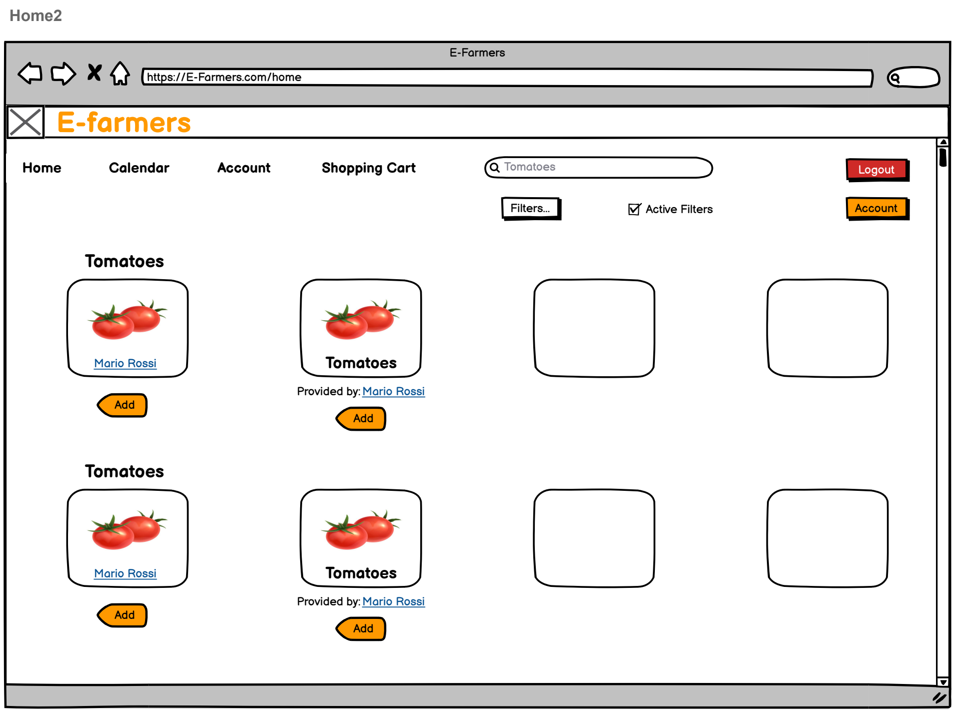
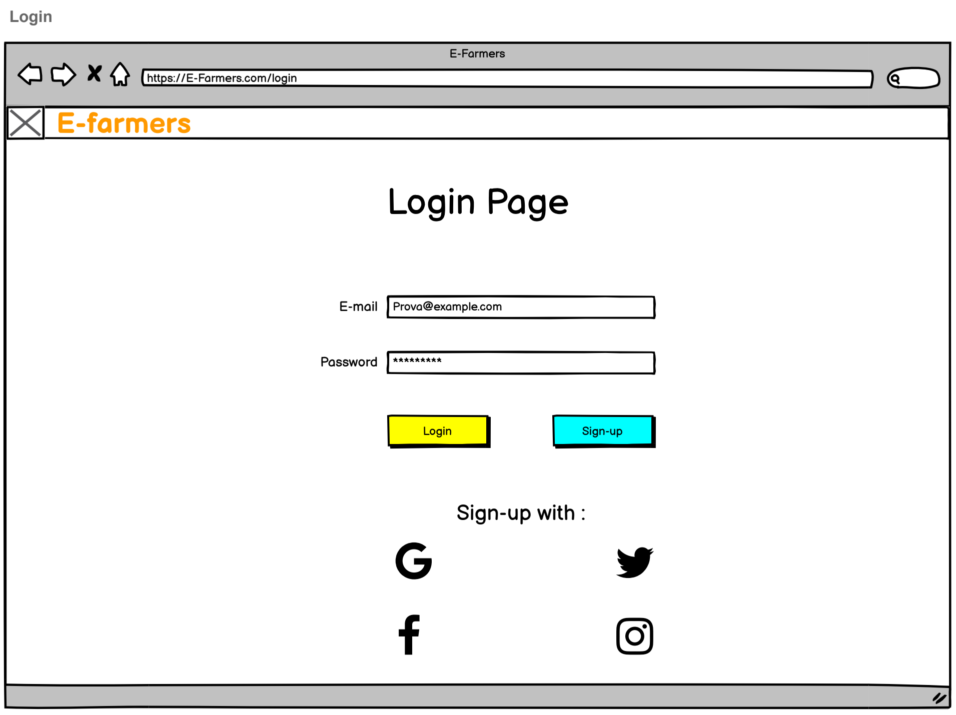


Figure 9.1: Home Page

## 9.2 User API

### 9.2.1 Login



https://E-Farmers.com/login

Login Page

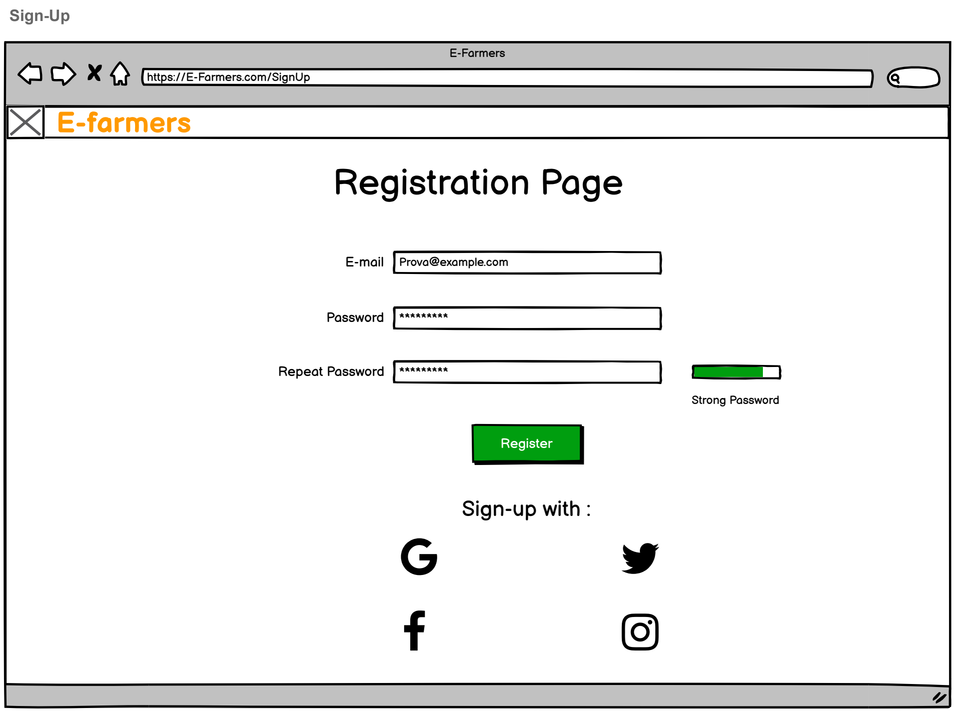
E-mail: Prova@example.com

Password: *********

Sign-up with :

Figure 9.2: User Login

### 9.2.2 Registration



https://E-Farmers.com/SignUp

Registration Page

E-mail: Prova@example.com

Password: *********

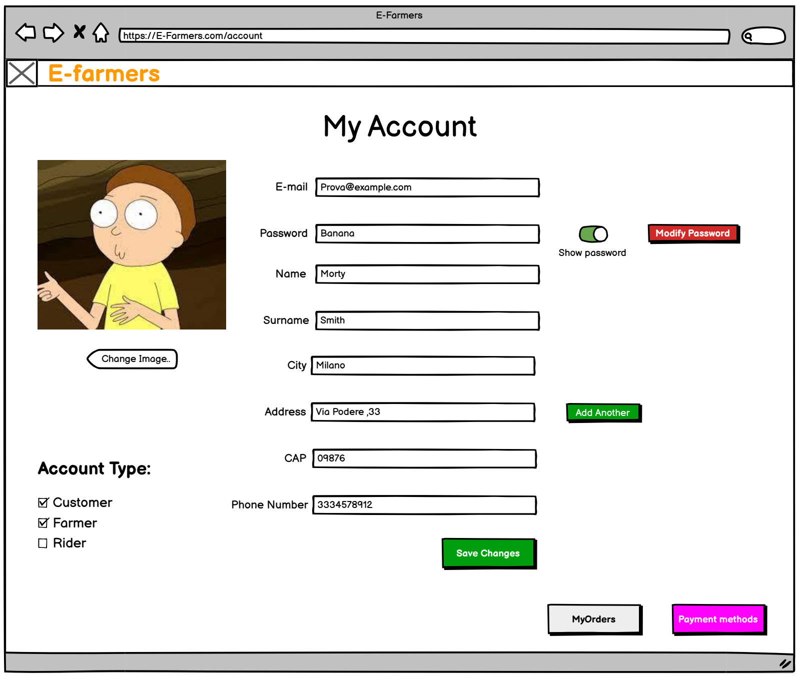
Repeat Password: *********

Sign-up with :

Figure 9.3: User Registration

### 9.2.3 Profile

Account



E-Farmers

https://E-Farmers.com/account

E-farmers

My Account

| | |
|---|---|
| E-mail | Prova@example.com |
| Password | Banana |
| Name | Morty |
| Surname | Smith |
| City | Milano |
| Address | Via Podere ,33 |
| CAP | 09876 |
| Phone Number | 3334578912 |

Show password

Account Type:

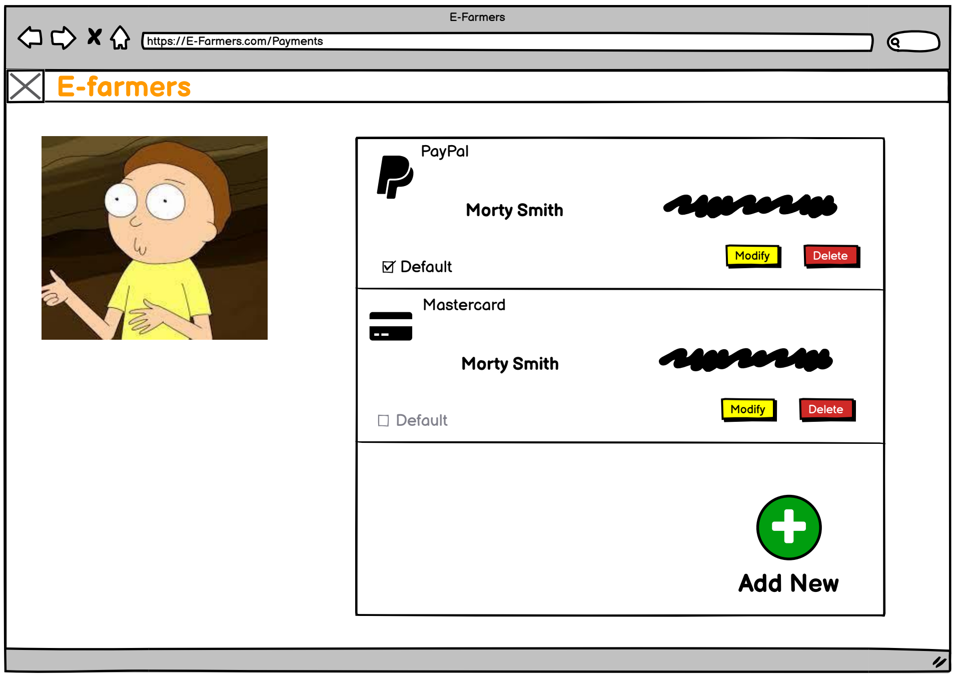
- [✓ Customer]
- [✓ Farmer]
- Rider

Figure 9.4: User Profile

## 9.3 Payments & Orders API

### 9.3.1 Payment Method

Payment methods



E-Farmers

https://E-Farmers.com/Payments

E-farmers

PayPal

Morty Smith [redacted]

[✓ Default]

Mastercard

Morty Smith [redacted]

Default

Figure 9.5: Payment Method

### 9.3.2 Order Visualization

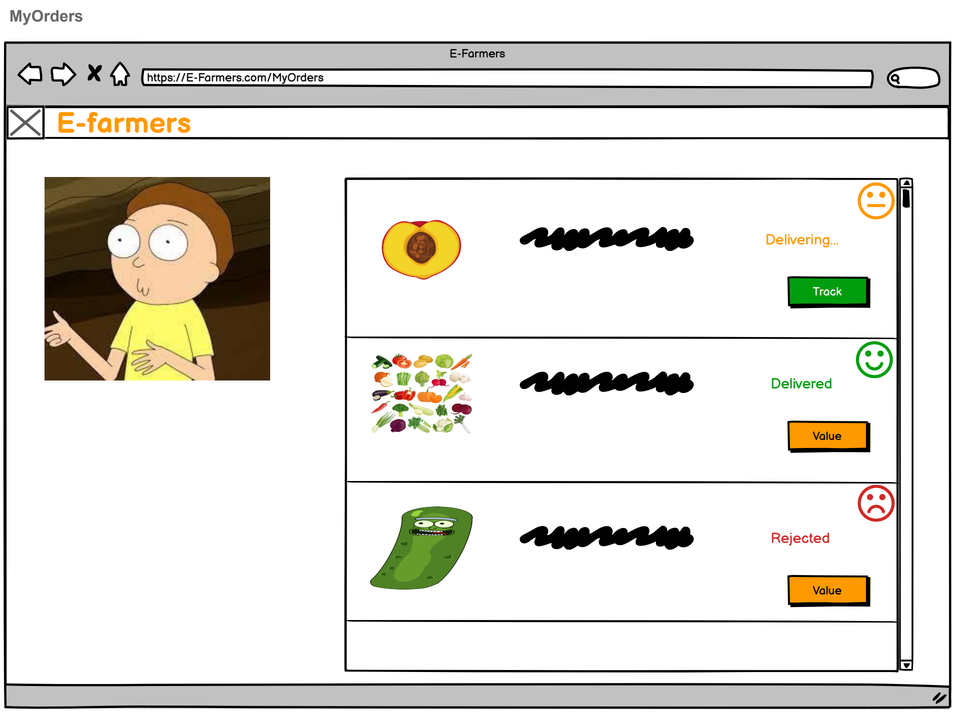


Figure 9.6: Order Visualization

## 9.4 ShoppingCart APIs

### 9.4.1 Cart Visualization

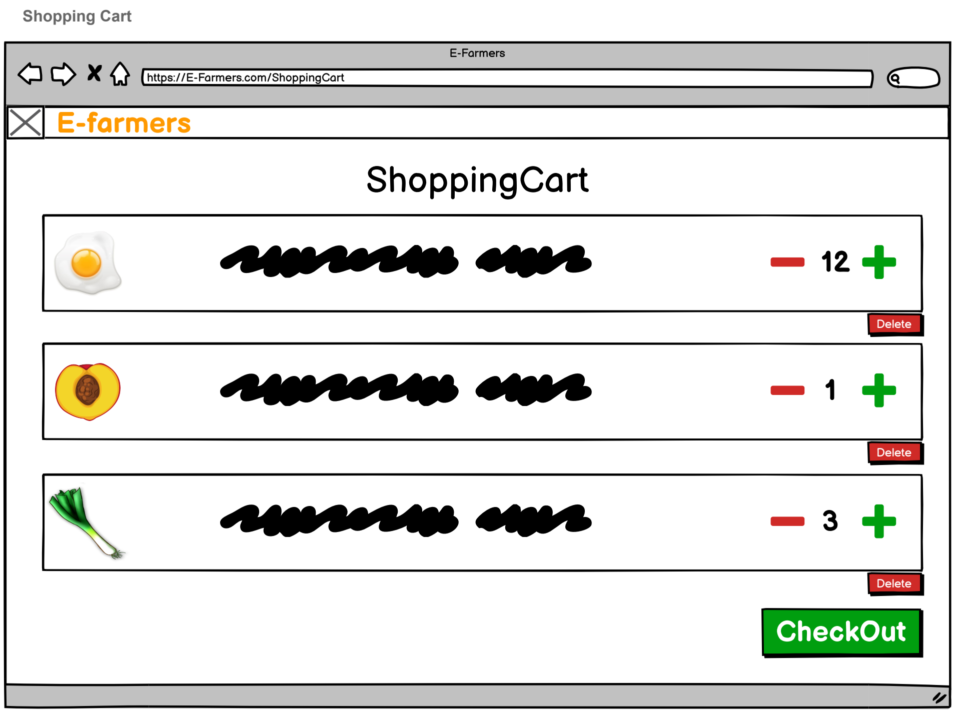
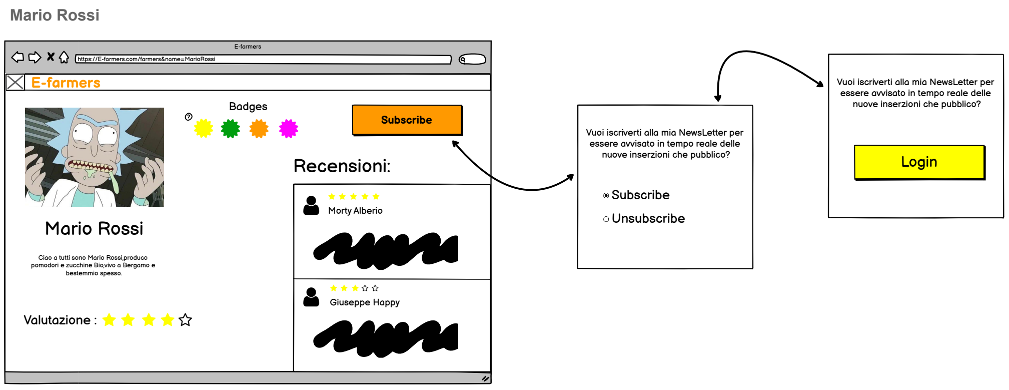


Figure 9.7: Cart Visualization

## 9.5 Subscription Services



Ciao a tutti sono Mario Rossi,produco pomodori e zucchine Bio,vivo a Bergamo e bestemmio spesso.

Vuoi iscriverti alla mia NewsLetter per essere avvisato in tempo reale delle nuove inserzioni che pubblico?

[✓ Subscribe] Unsubscribe

Vuoi iscriverti alla mia NewsLetter per essere avvisato in tempo reale delle nuove inserzioni che pubblico?

Figure 9.8: Subscription Operation