Upgrade Sprint 2 :

Content:

Summary of general upgrades made in Sprint 2:

- More visual presentation
- In this sprint we have meet more times and did the proper documentation for every decision made
- Back-end added for both applications
- Pro App front end developed
- Changes in economical strategy in order to reflect the received feedback

1. Technological part.

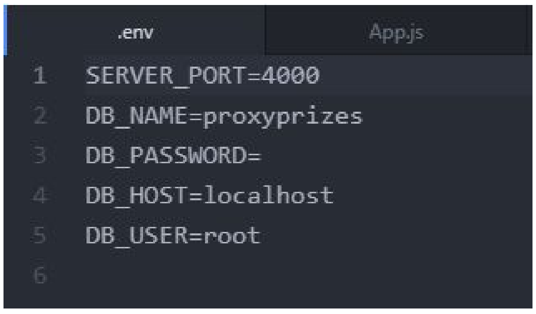
The main changes that occurred during this second sprint were related to the back-end's settlement.

As we previously explained it in the architecture diagram, we chose to use node.js in order to implement the back end server so that we could use javascript for both client and server sides.

Server configuration

The first part will explain the configuration of this server. To initiate our local server, we added the express library to the project.

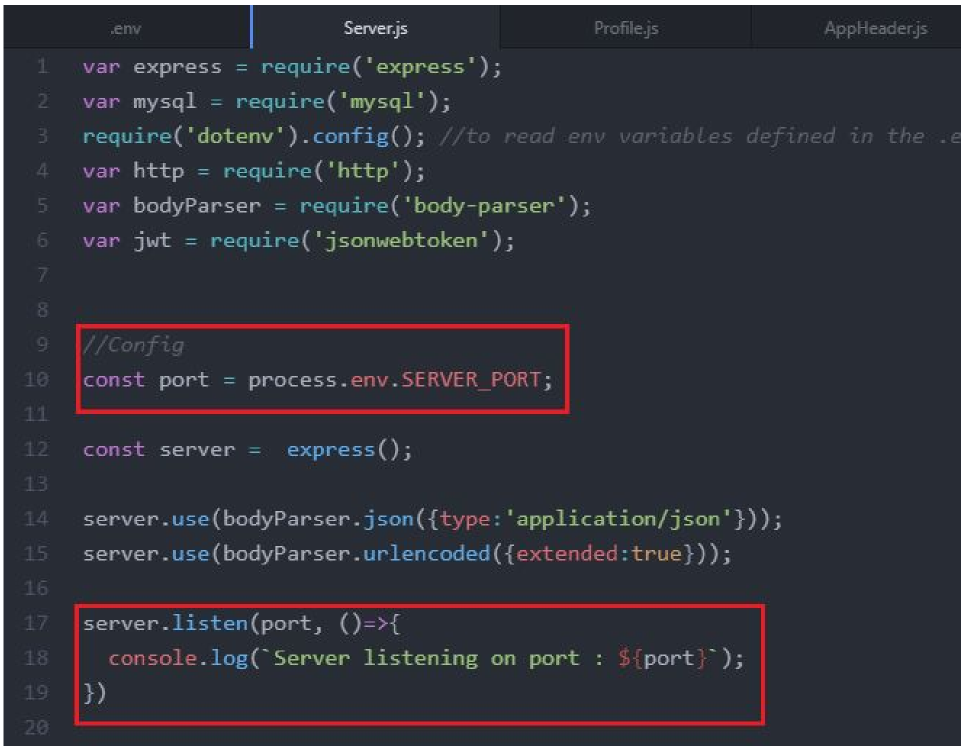
As it is crucial not exposing key information such as server port, or database information, we created a .env to cluster all those information.



```
SERVER_PORT=4000
DB_NAME=proxyprizes
DB_PASSWORD=
DB_HOST=localhost
DB_USER=root

```

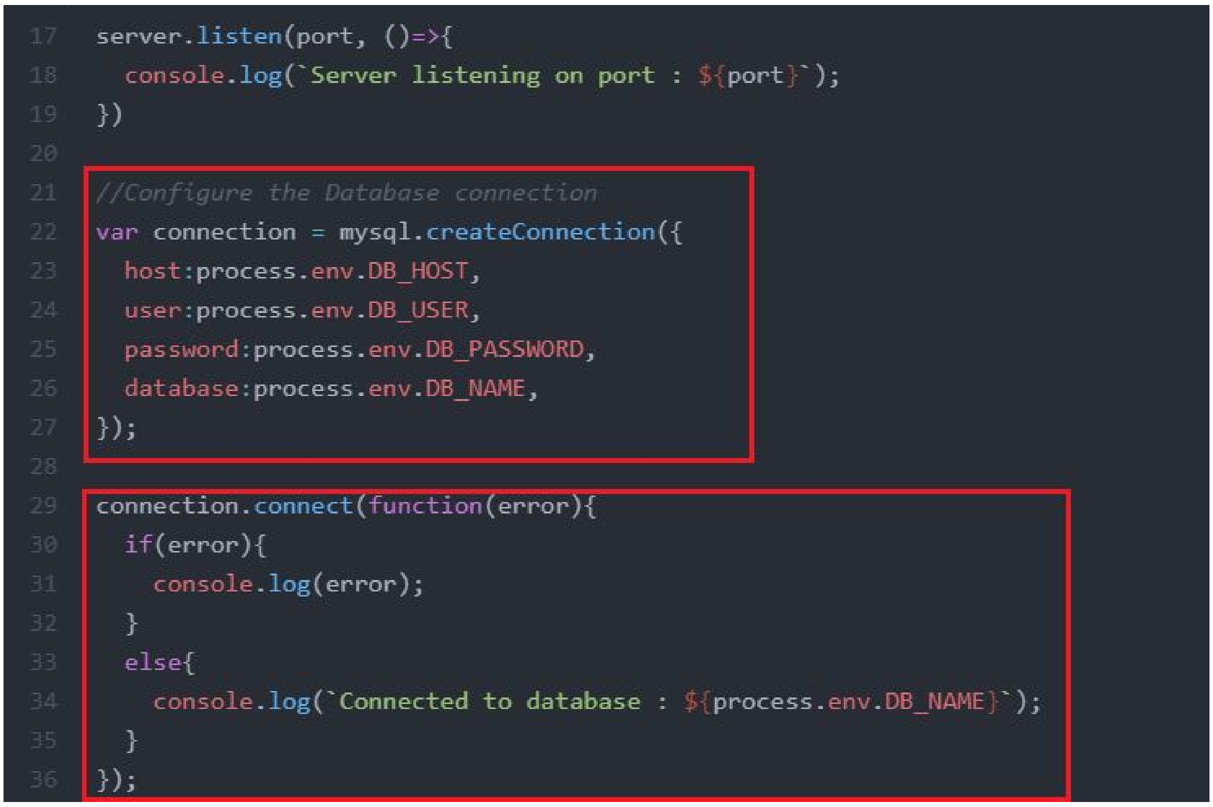
The .env file is never committed to the repository so that the sensitive information are not exposed. The required information are then extracted from this .env file to be injected within the server's code. In this way, it prevents sensitive information to be displayed in the code.

The following screenshot illustrates the injection of the server's port data from the .env file.



```
1   var express = require('express');
2   var mysql = require('mysql');
3   require('dotenv').config(); //to read env variables defined in the .e
4   var http = require('http');
5   var bodyParser = require('body-parser');
6   var jwt = require('jsonwebtoken');
7
8
9   //Config
10  const port = process.env.SERVER_PORT;
11
12  const server =  express();
13
14  server.use(bodyParser.json({type:'application/json'}));
15  server.use(bodyParser.urlencoded({extended:true}));
16
17  server.listen(port, ()=>{
18    console.log(`Server listening on port : ${port}`);
19  })
20
```

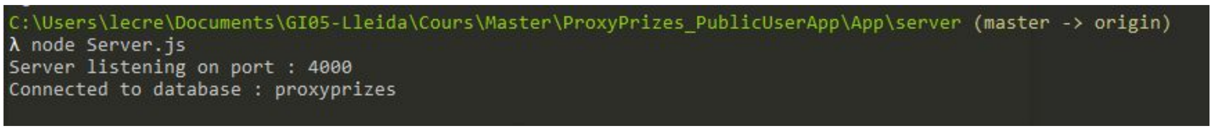
The same system was used to generate kind of "datasources" to connect the server to the database. As we were more used to using relational databases, we decided to choose a mysql database.
First , we need to import the mysql driver (which corresponds to line 2).



```javascript
server.listen(port, ()=>{
  console.log(`Server listening on port : ${port}`);
})

//Configure the Database connection
var connection = mysql.createConnection({
  host:process.env.DB_HOST,
  user:process.env.DB_USER,
  password:process.env.DB_PASSWORD,
  database:process.env.DB_NAME,
});

connection.connect(function(error){
  if(error){
    console.log(error);
  }
  else{
    console.log(`Connected to database : ${process.env.DB_NAME}`);
  }
});
```

Then we configure the database connection by importing all the relevant information from the .env file.
Finally, we initiate the connection between the server and the database.

The following screenshot shows the message displayed in the server console when the connection is successful.

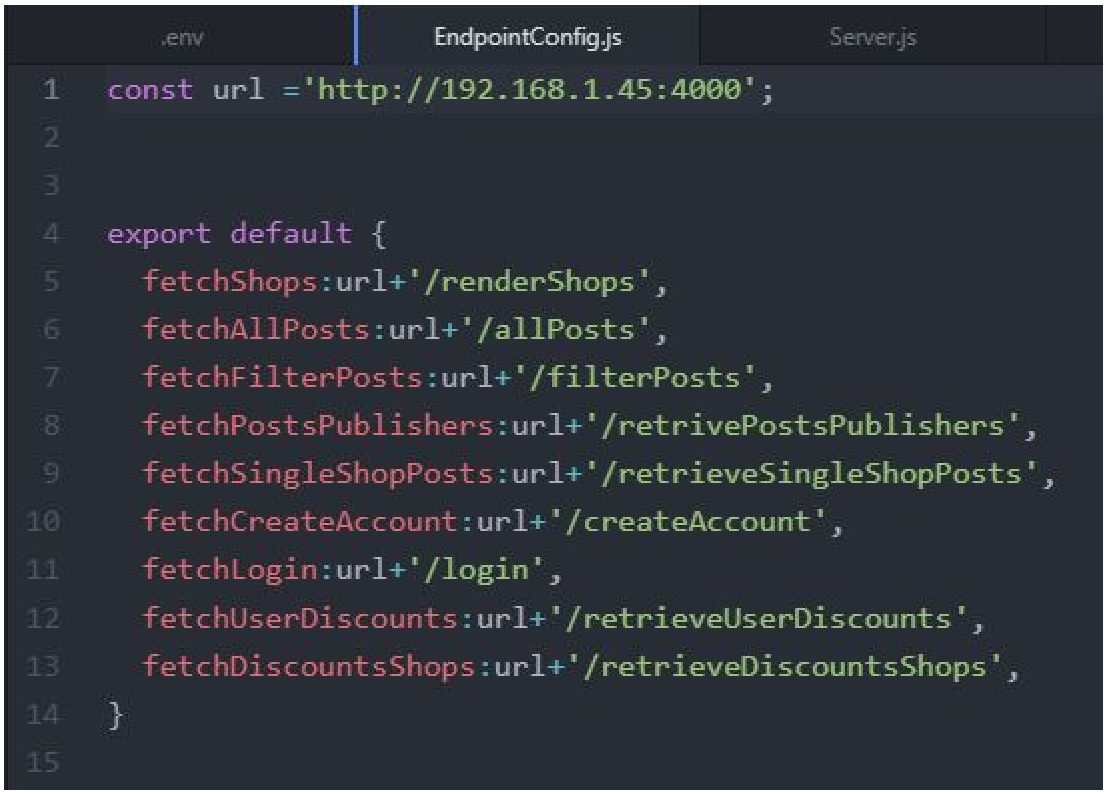


```
C:\Users\lecre\Documents\GI05-Lleida\Cours\Master\ProxyPrizes_PublicUserApp\App\server (master -> origin)
λ node Server.js
Server listening on port : 4000
Connected to database : proxyprizes
```

Route and endpoints settlement

All along the server file, we created many different endpoints so that our client applications can send requests to the server.

In order to prevent those routes from being exposed in our client applications, we also created an EndpointConfig file to cluster all the routes. This file has also been added to the gitignore (as the .env file) to prevent the routes from being uploaded.



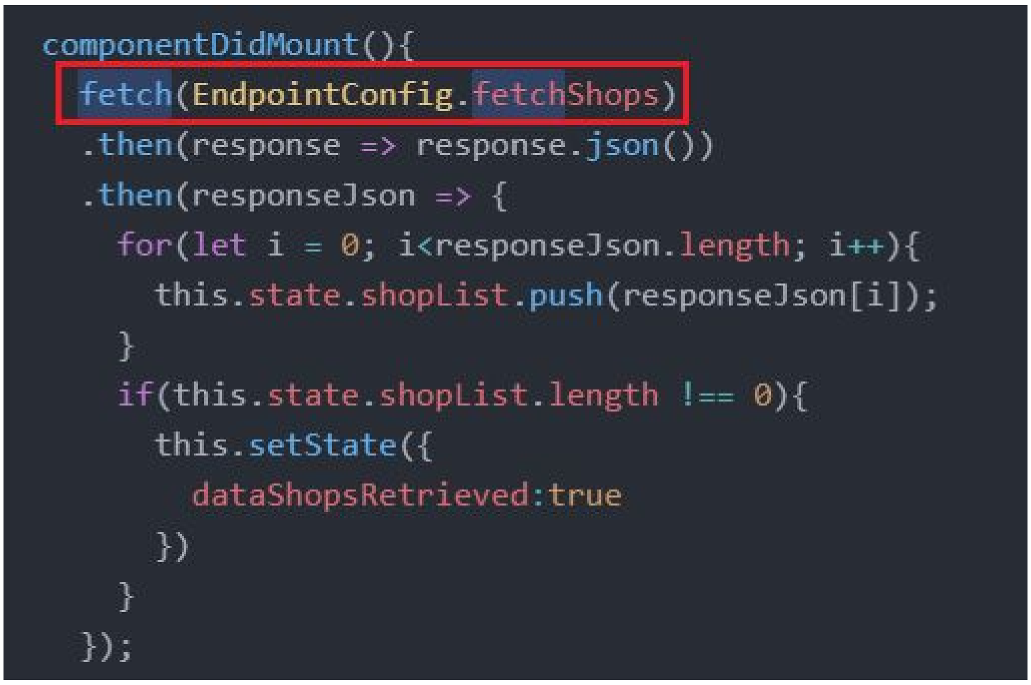
```
const url ='http://192.168.1.45:4000';


export default {
  fetchShops:url+'/renderShops',
  fetchAllPosts:url+'/allPosts',
  fetchFilterPosts:url+'/filterPosts',
  fetchPostsPublishers:url+'/retrivePostsPublishers',
  fetchSingleShopPosts:url+'/retrieveSingleShopPosts',
  fetchCreateAccount:url+'/createAccount',
  fetchLogin:url+'/login',
  fetchUserDiscounts:url+'/retrieveUserDiscounts',
  fetchDiscountsShops:url+'/retrieveDiscountsShops',
}
```

Request example

The following paragraph will briefly demonstrate how a GET and POST request to the server with the related interactions to the database.

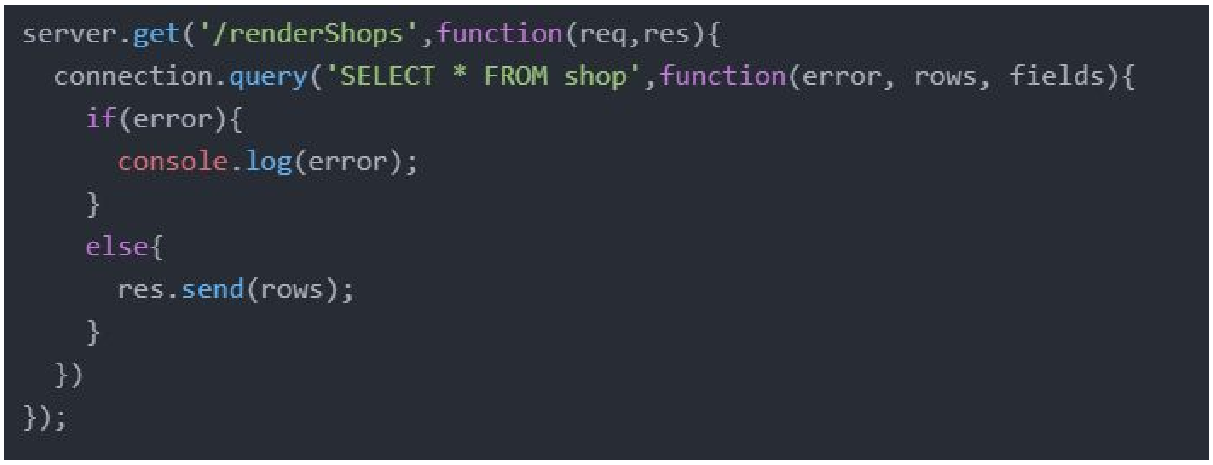
We will fisrt have a look at a GET request to render the shops on the application’s map.

The first part of the process occurs in the client side by using the fetch API to call the related endpoint :



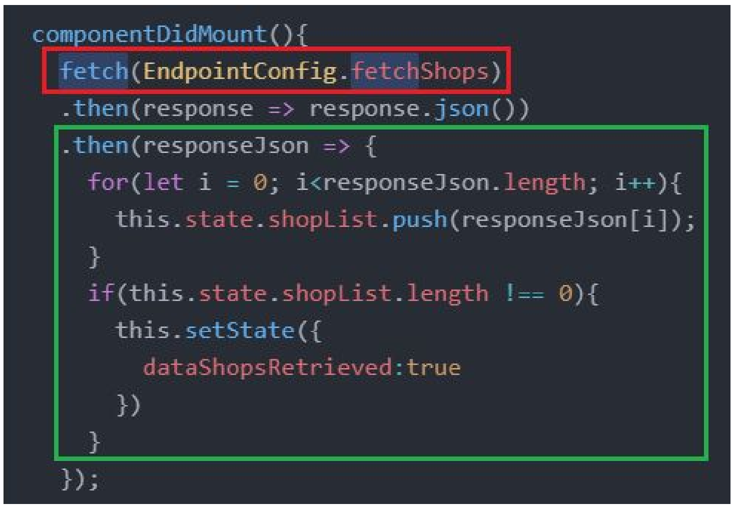
```
componentDidMount(){
  fetch(EndpointConfig.fetchShops)
   .then(response => response.json())
   .then(responseJson => {
     for(let i = 0; i<responseJson.length; i++){
       this.state.shopList.push(responseJson[i]);
     }
     if(this.state.shopList.length !== 0){
       this.setState({
         dataShopsRetrieved:true
       })
     }
   });
```

Once the endpoint is hitted, ther server sends an SQL request to the database throught the **connection** object.
The result is then sended back to the client.



```
server.get('/renderShops',function(req,res){
  connection.query('SELECT * FROM shop',function(error, rows, fields){
    if(error){
      console.log(error);
    }
    else{
      res.send(rows);
    }
  })
});
```

In the following screenshot, the client browses the content of the json response to push each shop object within the shopList state of the ShopMap component.



```
componentDidMount(){
  fetch(EndpointConfig.fetchShops)
  .then(response => response.json())
  .then(responseJson => {
    for(let i = 0; i<responseJson.length; i++){
      this.state.shopList.push(responseJson[i]);
    }
    if(this.state.shopList.length !== 0){
      this.setState({
        dataShopsRetrieved:true
      })
    }
  });
```

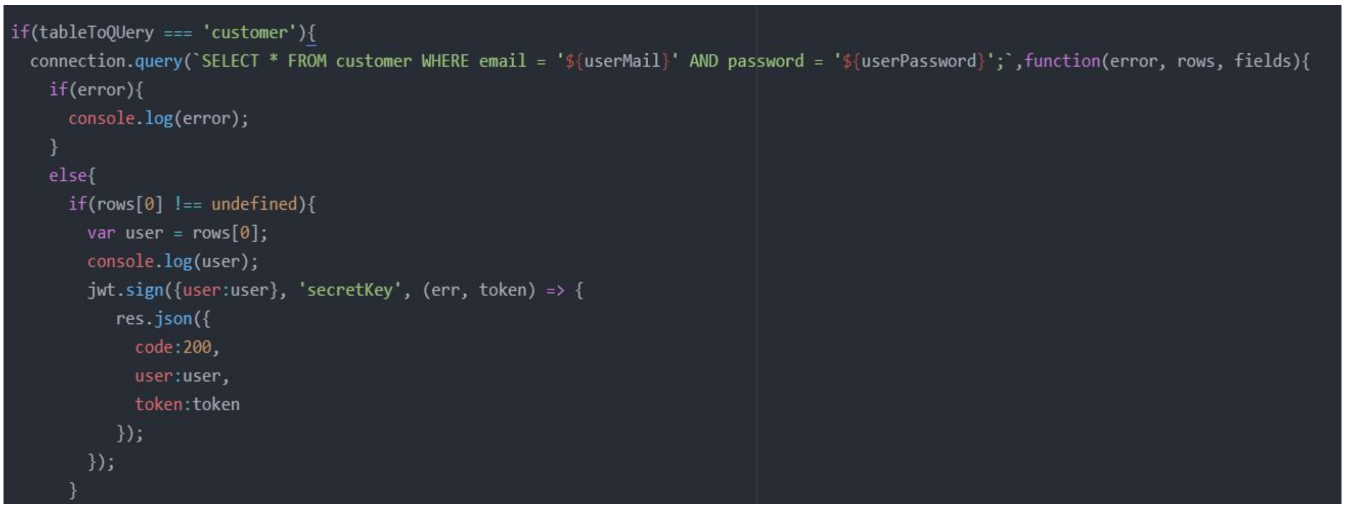
Once all the data has been loaded, a boolean value is set to “true” in order to trigger the display of those datas in the render method.

JWT tokenization

As we previously explained during the last sprint, we wanted to use the JWT Api in order to create tokens for the connected users in our application.

The tokenization aims to embody a connected user through the application. It is also useful for security purposes. In this way, it is possible to protect some of our backend endpoints by granting access to those specific endpoints only through a valid token. This feature has not been implemented yet but we could provide it in the final sprint to protect some sensitive routes such as user profile information or post publishing.

The tokenization process intervenes during the login step.



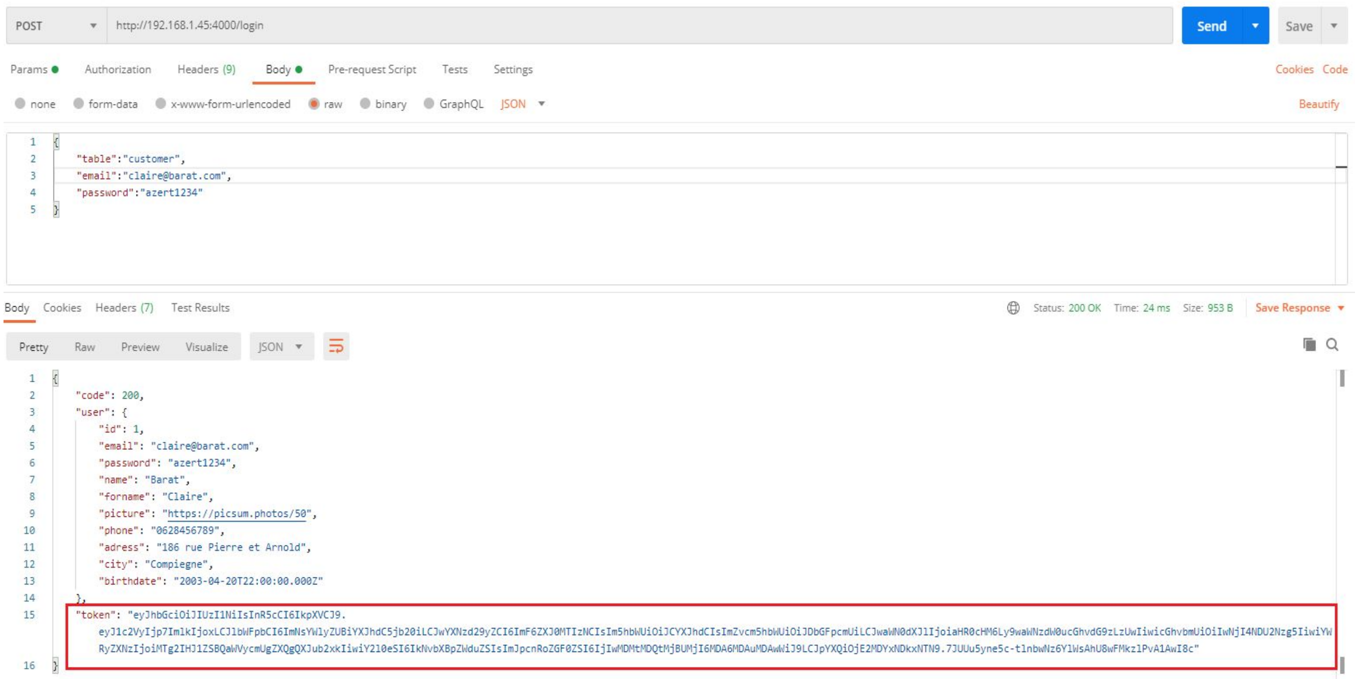
```
if(tableToQUery === 'customer'){
  connection.query(`SELECT * FROM customer WHERE email = '${userMail}' AND password = '${userPassword}';`,function(error, rows, fields){
    if(error){
      console.log(error);
    }
    else{
      if(rows[0] !== undefined){
        var user = rows[0];
        console.log(user);
        jwt.sign({user:user}, 'secretKey', (err, token) => {
          res.json({
            code:200,
            user:user,
            token:token
          });
        });
      }
```

First, the server sends a request to the database to retrieve the user, passing the user’s mail and password coming from the login form.

Once the user has been retrieved, we use the sign method of the jwt object to generate the token and we then send it back to the client.

In a reel situation, we would generate the secret key thanks to a key-generator.

Here is a postman example of the token sended to the client by the server after a successful connection :



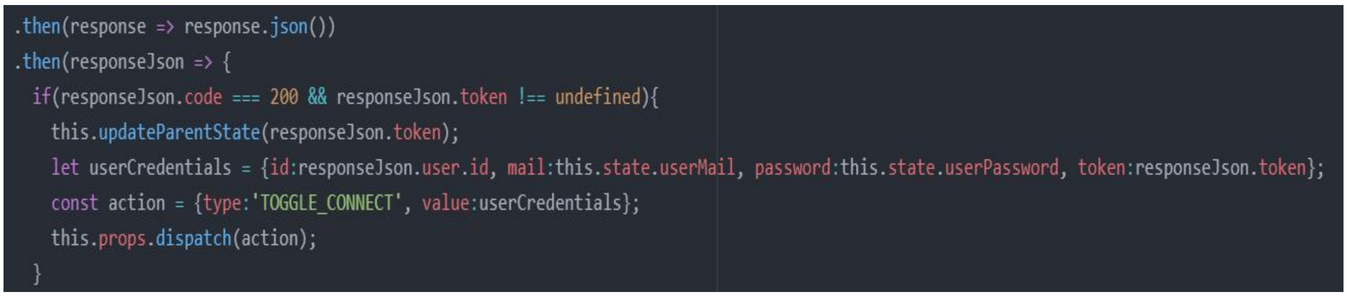
```
http://192.168.1.45:4000/login
```

```
{
    "table":"customer",
    "email":"claire@barat.com",
    "password":"azert1234"
}
```

Status: 200 OK Time: 24 ms Size: 953 B

```
{
    "code": 200,
    "user": {
        "id": 1,
        "email": "claire@barat.com",
        "password": "azert1234",
        "name": "Barat",
        "forname": "Claire",
        "picture": "https://picsum.photos/50",
        "phone": "0628456789",
        "adress": "186 rue Pierre et Arnold",
        "city": "Compiegne",
        "birthdate": "2003-04-20T22:00:00.000Z"
    },
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ1c2VyIjp7ImlkIjoxLCJlbWFpbCI6ImNsYWlyZUBiYXJhdC5jb20iLCJwYXNzd29yZCI6ImF6ZXJ0MTIzNCIsIm5hbWUiOiJCYXJhdCIsImZvcm5hbWUiOiJDbGFpcmUiLCJwaWN0dXJlIjoiaHR0cHM6Ly9waWNzdW0ucGhvdG9zLzUwIiwicGhvbmUiOiIwNjI4NDU2Nzg5IiwiYW
        RyZXNzIjoiMTg2IHJ1ZSBQaWVycmUgZXQgQXJub2xkIiwiY2l0eSI6IkNvbXBpZWduZSIsImJpcnRoZGF0ZSI6IjIwMDMtMDQtMjBUMjI6MDA6MDAuMDAwWiJ9LCJpYXQiOjE2MDYxNDkxNTN9.7JUUu5yne5c-tlnbwNz6YlWsAhU8wFMkz1PvA1AwI8c"
}
```

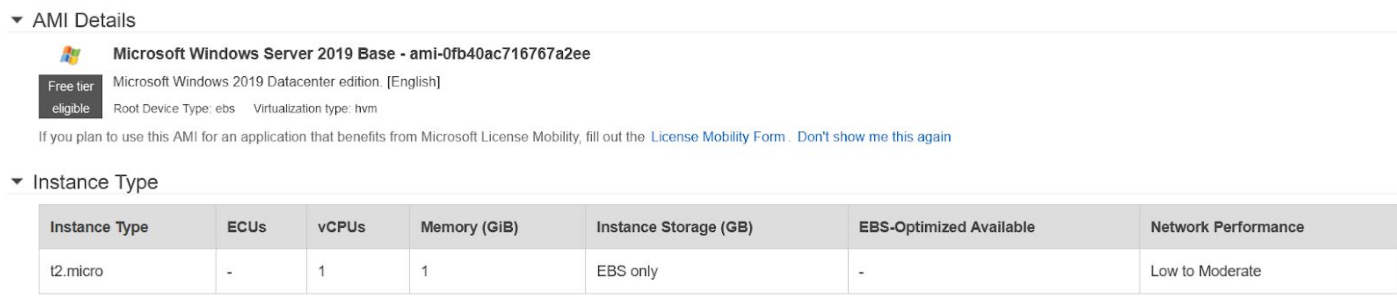
Once the response of the server is sended to the client, the token will be store in the client side thanks to the redux cache local storage.



```
.then(response => response.json())
.then(responseJson => {
  if(responseJson.code === 200 && responseJson.token !== undefined){
    this.updateParentState(responseJson.token);
    let userCredentials = {id:responseJson.user.id, mail:this.state.userMail, password:this.state.userPassword, token:responseJson.token};
    const action = {type:'TOGGLE_CONNECT', value:userCredentials};
    this.props.dispatch(action);
  }
```

Solution to deploy the server

One of the proposed solutions for deploying the server, is using Amazon AWS EC2. Amazon EC2 is basically a virtual machine hosted on Amazon Web Services, it can be Windows or Linux, for fastest implementation we would utilize a Windows-based host on EC2.



▾ AMI Details

Microsoft Windows Server 2019 Base - ami-0fb40ac716767a2ee

Free tier eligible

Microsoft Windows 2019 Datacenter edition. [English]

Root Device Type: ebs Virtualization type: hvm

If you plan to use this AMI for an application that benefits from Microsoft License Mobility, fill out the License Mobility Form. Don't show me this again

▾ Instance Type

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	-	1	1	EBS only	-	Low to Moderate

The process would go as follow:

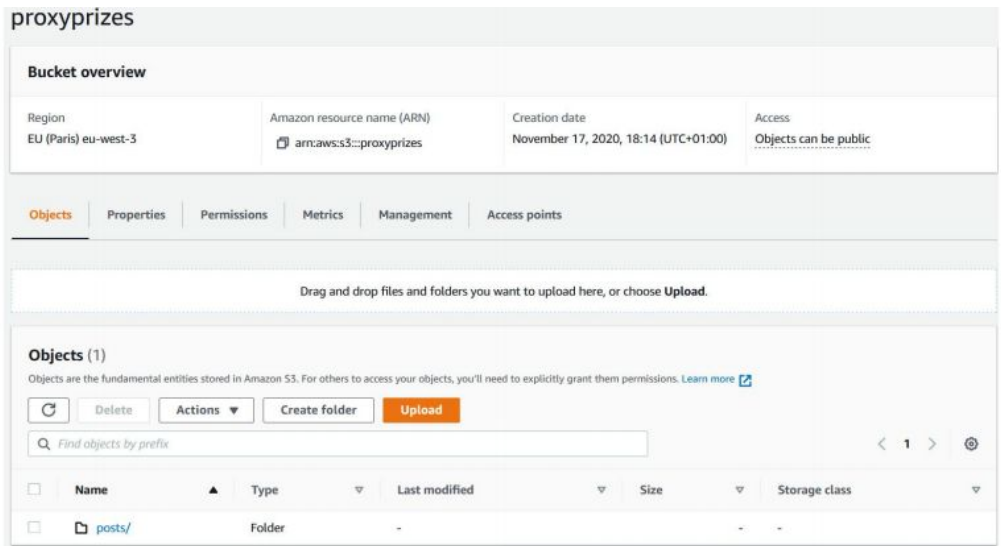
- Create and pre-configure a Windows-based host, installing Node.js and other server dependencies.
- We would clone our servers and then start them (Server for pro app and one for the public app) on different ports.
- After this, the host would be ready to act as a server.

Access and Security considerations:

- We can configure and control who have access to the server, this can be done through IP routing in AWS EC2 using Security groups
- We can block or allow public access
- The access for fetching different functions would be controlled by endpoints pre-configured

AWS S3 Image upload

We have decided to adopt the AWS S3 as a storage solution for our application, this way, in the database we only have to store the image link.



proxyprizes

Bucket overview

Region	Amazon resource name (ARN)	Creation date	Access
EU (Paris) eu-west-3	arn:aws:s3:::proxyprizes	November 17, 2020, 18:14 (UTC+01:00)	Objects can be public

Objects | Properties | Permissions | Metrics | Management | Access points

Drag and drop files and folders you want to upload here, or choose **Upload**.

Objects (1)

Objects are the fundamental entities stored in Amazon S3. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Delete | Actions ▾ | Create folder | Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
posts/	Folder	-	-	-

Amazon S3 is a simple service of storage, you can upload and retrieve files within it.

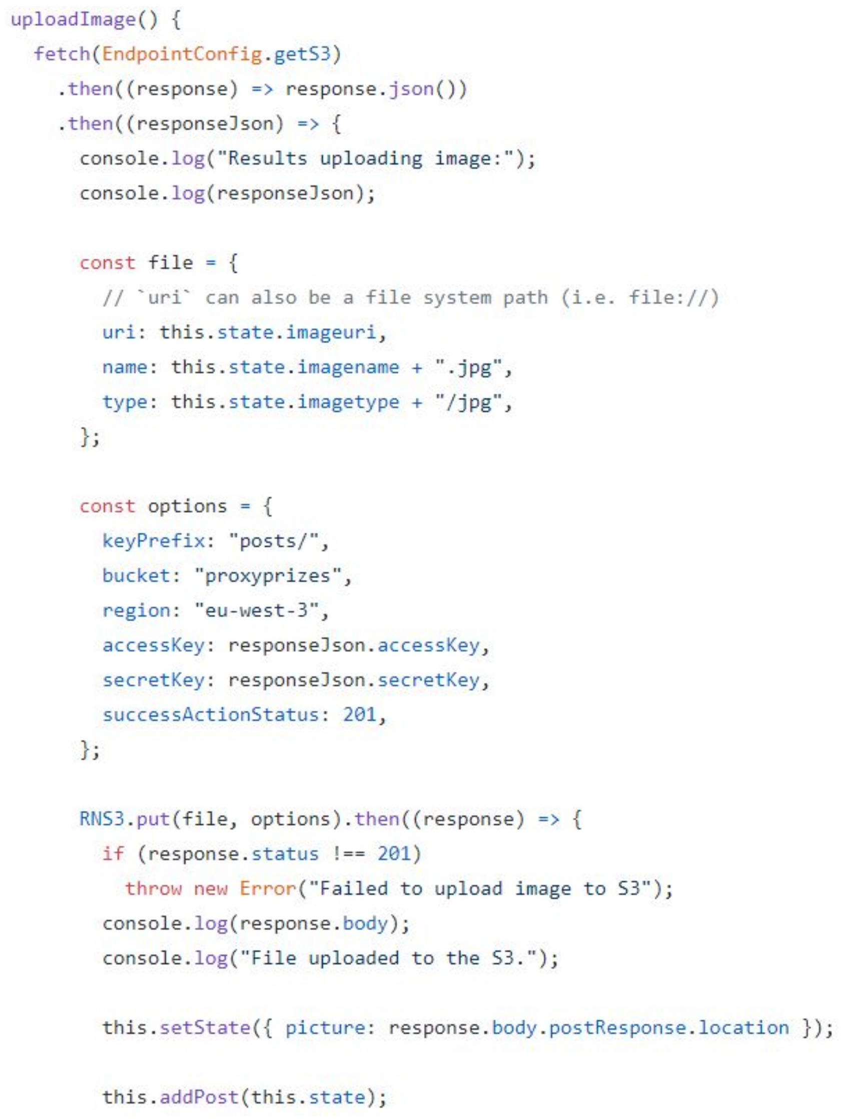
In order to access the S3, you must use this credentials which are in the .env (like other credentials, this is not visible on the code).

- Access Key ID
- Secret Access Key

We are using the following library in order to communicate with the S3

import { RNS3 } from "react-native-aws3";

Next, we have the function to do the upload

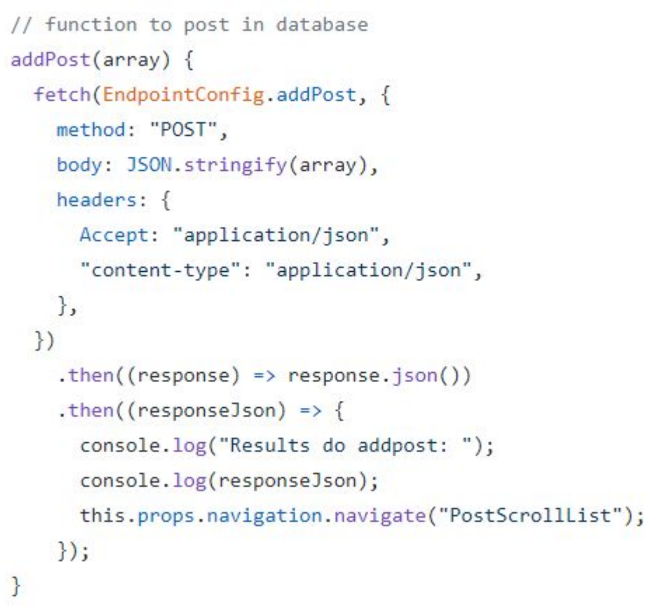


```
uploadImage() {
  fetch(EndpointConfig.getS3)
    .then((response) => response.json())
    .then((responseJson) => {
      console.log("Results uploading image:");
      console.log(responseJson);

      const file = {
        // `uri` can also be a file system path (i.e. file://)
        uri: this.state.imageuri,
        name: this.state.imagename + ".jpg",
        type: this.state.imagetype + "/jpg",
      };

      const options = {
        keyPrefix: "posts/",
        bucket: "proxyprizes",
        region: "eu-west-3",
        accessKey: responseJson.accessKey,
        secretKey: responseJson.secretKey,
        successActionStatus: 201,
      };

      RNS3.put(file, options).then((response) => {
        if (response.status !== 201)
          throw new Error("Failed to upload image to S3");
        console.log(response.body);
        console.log("File uploaded to the S3.");

        this.setState({ picture: response.body.postResponse.location });

        this.addPost(this.state);
```

What it does:

- Get the image information from the file that the user have selected in their device (path, name, type, etc)
- Gather that information, also get the credentials from the server to the S3
- Do the upload and then it returns the imageurl ready to insert into the database.

Function to add post to the database:



```
// function to post in database
addPost(array) {
  fetch(EndpointConfig.addPost, {
    method: "POST",
    body: JSON.stringify(array),
    headers: {
      Accept: "application/json",
      "content-type": "application/json",
    },
  })
    .then((response) => response.json())
    .then((responseJson) => {
      console.log("Results do addpost: ");
      console.log(responseJson);
      this.props.navigation.navigate("PostScrollList");
    });
}
```

Pro App Back-end

The back-end for the pro application is also finished, all current screens have functional server-side requests and functions, the structure of them follows the same as the public application, please refer to the Pro App Github for more details.

2.Financial part upgrades

Here we will list all changes made regarding the previous sprint.
Please refer to the financial document for extra information, all values are already updated to reflect the decisions made in this sprint.

Changes made:

1. Changed market and expansion strategy
2. Added explanations for each what-if scenario
3. Added a way to deal with the negative cashflow

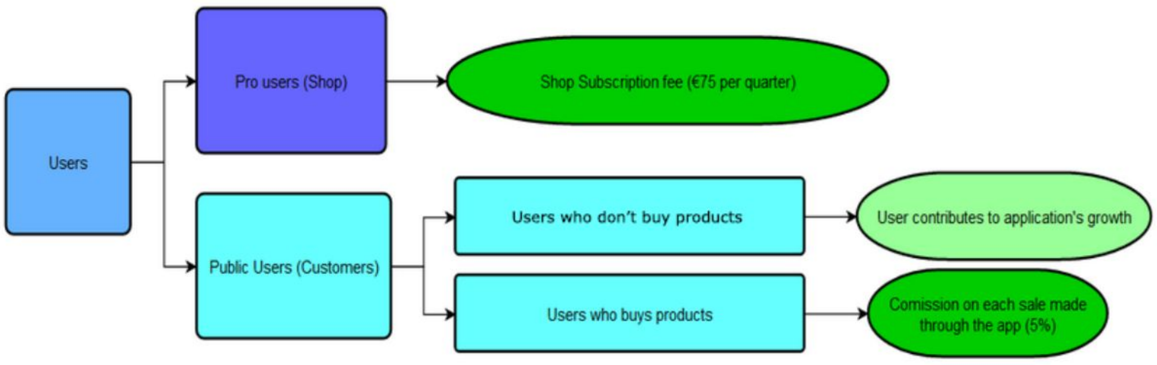
- Main excel document with all the data: https://udlcat-my.sharepoint.com/:x:/r/personal/dls5_alumnes_udl_cat/_layouts/15/doc2.aspx?sourcedoc=%7Bec14aa05-f822-44cb-bda3-8cfadf771bcd%7D&action=edit&activeCell=%27Planilha1%27!N21&wdrcid=5fbe3ecc-e4a0-4bd7-8e97-a61f70e7a74c&wdrldc=1&cid=6797ceaf-c490-4e01-81e8-237e4bdbadec
- Main financial document: available on github under /documentation/ folder

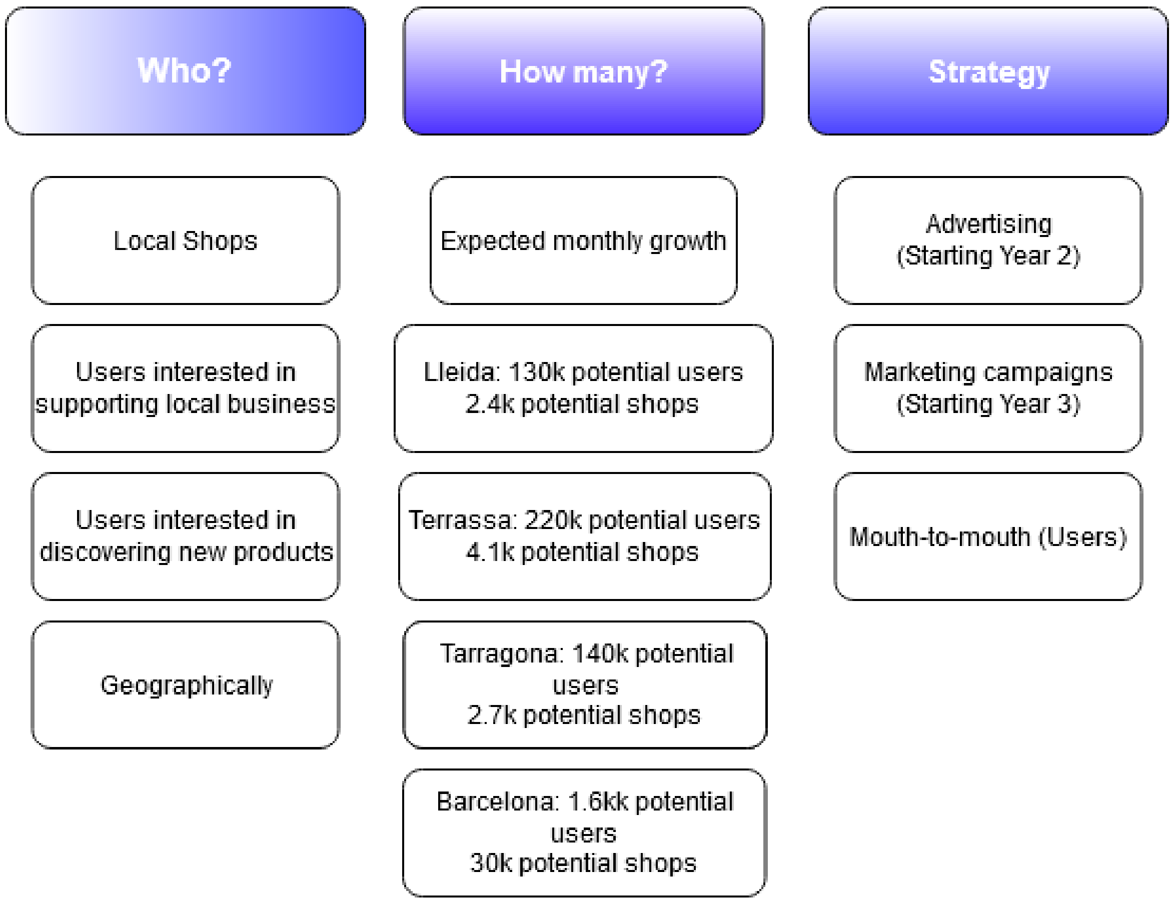
1. Market and expansion strategy

1. Year 2:
   a) Lleida (130k potential users, 2.4k potential shops)

2. Year 3:
   a) + Terrasa (220k potential users, 4.1k potential shops)
   b) + Tarragona (140k potential users, 2.7k potential shops)

3. Year 4:
   a) + Barcelona (1,6kk potential users, 30k potential shops)


Users
Pro users (Shop)
Shop Subscription fee (€75 per quarter)
Public Users (Customers)
Users who don't buy products
User contributes to application's growth
Users who buys products
Comission on each sale made through the app (5%)


Who?
How many?
Strategy
Local Shops
Expected monthly growth
Advertising (Starting Year 2)
Users interested in supporting local business
Lleida: 130k potential users 2.4k potential shops
Marketing campaigns (Starting Year 3)
Users interested in discovering new products
Terrassa: 220k potential users 4.1k potential shops
Mouth-to-mouth (Users)
Geographically
Tarragona: 140k potential users 2.7k potential shops
Barcelona: 1.6kk potential users 30k potential shops

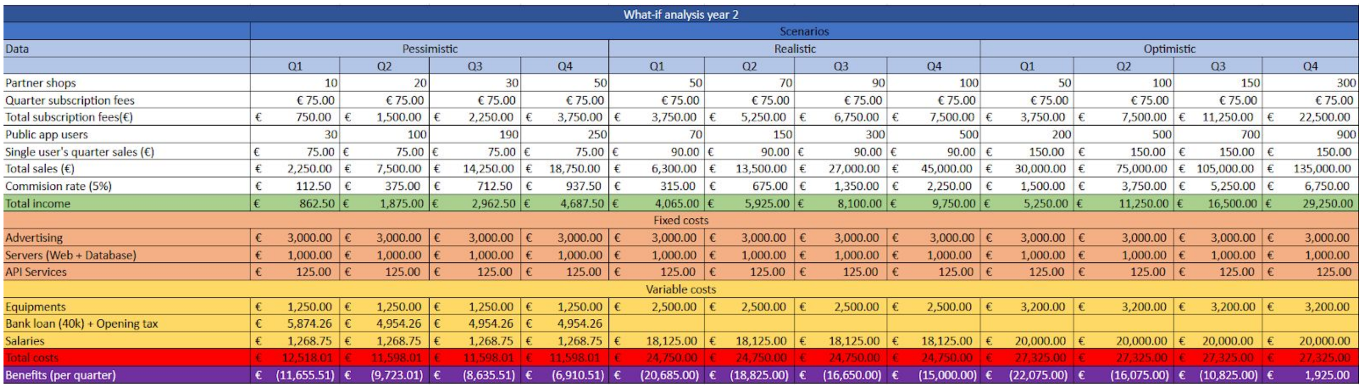
Considerations:

- Company is opened as a "Limited Society" without offices.
- The idea is to start the application in the local shops of Lleida. (Year 2)
- In Year 3 we expanded business to Terrassa and Tarragona.
- In Year 4 we expanded business to Barcelona, in this first year, we are on a "adaptation" process so the numbers do not reflect yet all the potential customers of the city.

2. Added explanations about each what-if scenarios

2.1 What if analysis

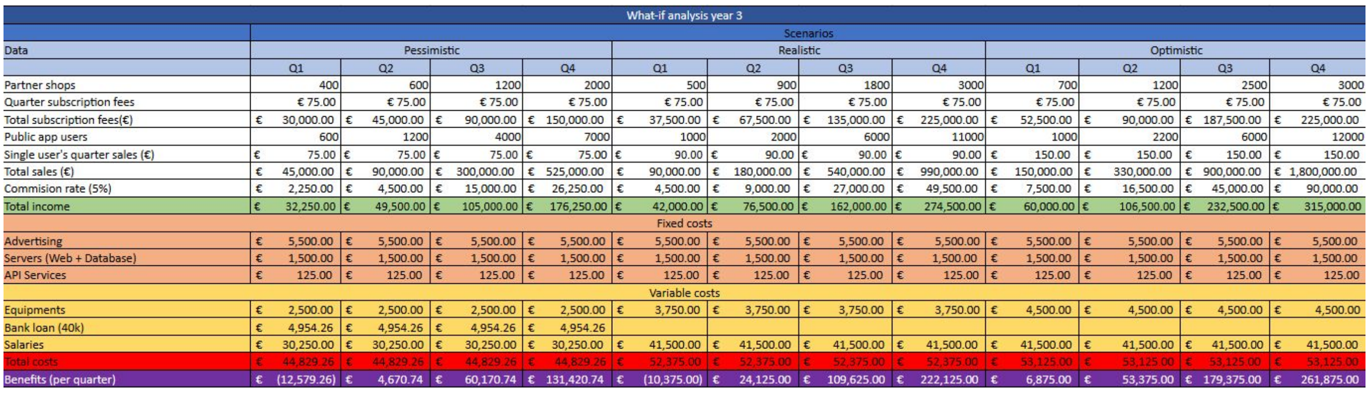
- **Year 2**



What-if analysis year 2												
	Scenarios											
Data	Pessimistic				Realistic				Optimistic			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Partner shops	10	20	30	50	50	70	90	100	50	100	150	300
Quarter subscription fees	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00
Total subscription fees(€)	€ 750.00	€ 1,500.00	€ 2,250.00	€ 3,750.00	€ 3,750.00	€ 5,250.00	€ 6,750.00	€ 7,500.00	€ 3,750.00	€ 7,500.00	€ 11,250.00	€ 22,500.00
Public app users	30	100	190	250	70	150	300	500	200	500	700	900
Single user's quarter sales (€)	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 150.00	€ 150.00	€ 150.00	€ 150.00
Total sales (€)	€ 2,250.00	€ 7,500.00	€ 14,250.00	€ 18,750.00	€ 6,300.00	€ 13,500.00	€ 27,000.00	€ 45,000.00	€ 30,000.00	€ 75,000.00	€ 105,000.00	€ 135,000.00
Commision rate (5%)	€ 112.50	€ 375.00	€ 712.50	€ 937.50	€ 315.00	€ 675.00	€ 1,350.00	€ 2,250.00	€ 1,500.00	€ 3,750.00	€ 5,250.00	€ 6,750.00
Total income	€ 862.50	€ 1,875.00	€ 2,962.50	€ 4,687.50	€ 4,065.00	€ 5,925.00	€ 8,100.00	€ 9,750.00	€ 5,250.00	€ 11,250.00	€ 16,500.00	€ 29,250.00
					Fixed costs							
Advertising	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00
Servers (Web + Database)	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00	€ 1,000.00
API Services	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00
					Variable costs							
Equipments	€ 1,250.00	€ 1,250.00	€ 1,250.00	€ 1,250.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 3,200.00	€ 3,200.00	€ 3,200.00	€ 3,200.00
Bank loan (40k) + Opening tax	€ 5,874.26	€ 4,954.26	€ 4,954.26	€ 4,954.26								
Salaries	€ 1,268.75	€ 1,268.75	€ 1,268.75	€ 1,268.75	€ 18,125.00	€ 18,125.00	€ 18,125.00	€ 18,125.00	€ 20,000.00	€ 20,000.00	€ 20,000.00	€ 20,000.00
Total costs	€ 12,518.01	€ 11,598.01	€ 11,598.01	€ 11,598.01	€ 24,750.00	€ 24,750.00	€ 24,750.00	€ 24,750.00	€ 27,325.00	€ 27,325.00	€ 27,325.00	€ 27,325.00
Benefits (per quarter)	€ (11,655.51)	€ (9,723.01)	€ (8,635.51)	€ (6,910.51)	€ (20,685.00)	€ (18,825.00)	€ (16,650.00)	€ (15,000.00)	€ (22,075.00)	€ (16,075.00)	€ (10,825.00)	€ 1,925.00

- Explanations:
    - In case of pessimistic scenario we are paying the Bank loan + taxes;
    - In case of pessimistic scenario we are spending less on equipments;
    - In case of pessimistic scenario our salaries are cut (just the fullstack developer is working half-time);
    - In case of realistic scenario all variables are the same as in the cashflow;
    - In case of Optimistic scenario, we are spending more on equipment, and on salaries (our salaries fulltime + fullstack fulltime);

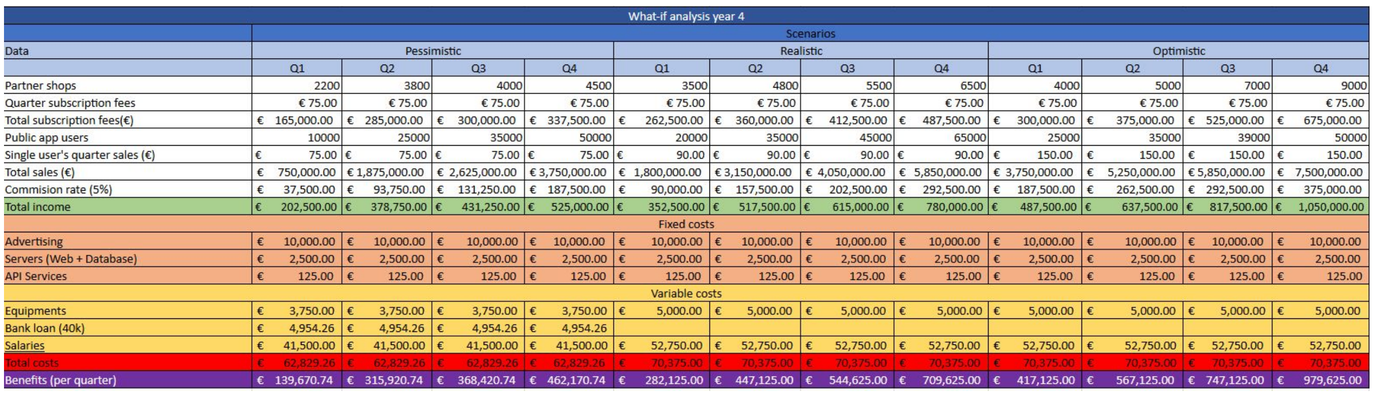
- **Year 3**



What-if analysis year 3												
	Scenarios											
Data	Pessimistic				Realistic				Optimistic			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Partner shops	400	600	1200	2000	500	900	1800	3000	700	1200	2500	3000
Quarter subscription fees	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00
Total subscription fees(€)	€ 30,000.00	€ 45,000.00	€ 90,000.00	€ 150,000.00	€ 37,500.00	€ 67,500.00	€ 135,000.00	€ 225,000.00	€ 52,500.00	€ 90,000.00	€ 187,500.00	€ 225,000.00
Public app users	600	1200	4000	7000	1000	2000	6000	11000	1000	2200	6000	12000
Single user's quarter sales (€)	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 150.00	€ 150.00	€ 150.00	€ 150.00
Total sales (€)	€ 45,000.00	€ 90,000.00	€ 300,000.00	€ 525,000.00	€ 90,000.00	€ 180,000.00	€ 540,000.00	€ 990,000.00	€ 150,000.00	€ 330,000.00	€ 900,000.00	€ 1,800,000.00
Commision rate (5%)	€ 2,250.00	€ 4,500.00	€ 15,000.00	€ 26,250.00	€ 4,500.00	€ 9,000.00	€ 27,000.00	€ 49,500.00	€ 7,500.00	€ 16,500.00	€ 45,000.00	€ 90,000.00
Total income	€ 32,250.00	€ 49,500.00	€ 105,000.00	€ 176,250.00	€ 42,000.00	€ 76,500.00	€ 162,000.00	€ 274,500.00	€ 60,000.00	€ 106,500.00	€ 232,500.00	€ 315,000.00
					Fixed costs							
Advertising	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00	€ 5,500.00
Servers (Web + Database)	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00	€ 1,500.00
API Services	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00
					Variable costs							
Equipments	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 3,750.00	€ 3,750.00	€ 3,750.00	€ 3,750.00	€ 4,500.00	€ 4,500.00	€ 4,500.00	€ 4,500.00
Bank loan (40k)	€ 4,954.26	€ 4,954.26	€ 4,954.26	€ 4,954.26								
Salaries	€ 30,250.00	€ 30,250.00	€ 30,250.00	€ 30,250.00	€ 41,500.00	€ 41,500.00	€ 41,500.00	€ 41,500.00	€ 41,500.00	€ 41,500.00	€ 41,500.00	€ 41,500.00
Total costs	€ 44,829.26	€ 44,829.26	€ 44,829.26	€ 44,829.26	€ 52,375.00	€ 52,375.00	€ 52,375.00	€ 52,375.00	€ 53,125.00	€ 53,125.00	€ 53,125.00	€ 53,125.00
Benefits (per quarter)	€ (12,579.26)	€ 4,670.74	€ 60,170.74	€ 131,420.74	€ (10,375.00)	€ 24,125.00	€ 109,625.00	€ 222,125.00	€ 6,875.00	€ 53,375.00	€ 179,375.00	€ 261,875.00

- Explanations:
    - In case of pessimistic scenario we are paying the Bank loan + taxes;
    - In case of pessimistic scenario we are spending less on equipments;
    - In case of pessimistic scenario we are spending less on Salaries (just 3 of us + full stack);
    - In case of realistic scenario all variables are the same as in the cashflow;
    - In the optimistic scenario and realistic, we are spending a little more on salaries (3 of us + fullstack + sales manager, everyone fulltime)

- **Year 4**



What-if analysis year 4												
	Scenarios											
Data	Pessimistic				Realistic				Optimistic			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Partner shops	2200	3800	4000	4500	3500	4800	5500	6500	4000	5000	7000	9000
Quarter subscription fees	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00
Total subscription fees(€)	€ 165,000.00	€ 285,000.00	€ 300,000.00	€ 337,500.00	€ 262,500.00	€ 360,000.00	€ 412,500.00	€ 487,500.00	€ 300,000.00	€ 375,000.00	€ 525,000.00	€ 675,000.00
Public app users	10000	25000	35000	50000	20000	35000	45000	65000	25000	35000	39000	50000
Single user's quarter sales (€)	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 150.00	€ 150.00	€ 150.00	€ 150.00
Total sales (€)	€ 750,000.00	€ 1,875,000.00	€ 2,625,000.00	€ 3,750,000.00	€ 1,800,000.00	€ 3,150,000.00	€ 4,050,000.00	€ 5,850,000.00	€ 3,750,000.00	€ 5,250,000.00	€ 5,850,000.00	€ 7,500,000.00
Commision rate (5%)	€ 37,500.00	€ 93,750.00	€ 131,250.00	€ 187,500.00	€ 90,000.00	€ 157,500.00	€ 202,500.00	€ 292,500.00	€ 187,500.00	€ 262,500.00	€ 292,500.00	€ 375,000.00
Total income	€ 202,500.00	€ 378,750.00	€ 431,250.00	€ 525,000.00	€ 352,500.00	€ 517,500.00	€ 615,000.00	€ 780,000.00	€ 487,500.00	€ 637,500.00	€ 817,500.00	€ 1,050,000.00
					Fixed costs							
Advertising	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00
Servers (Web + Database)	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00	€ 2,500.00
API Services	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00	€ 125.00
					Variable costs							
Equipments	€ 3,750.00	€ 3,750.00	€ 3,750.00	€ 3,750.00	€ 5,000.00	€ 5,000.00	€ 5,000.00	€ 5,000.00	€ 5,000.00	€ 5,000.00	€ 5,000.00	€ 5,000.00
Bank loan (40k)	€ 4,954.26	€ 4,954.26	€ 4,954.26	€ 4,954.26								
Salaries	€ 41,500.00	€ 41,500.00	€ 41,500.00	€ 41,500.00	€ 52,750.00	€ 52,750.00	€ 52,750.00	€ 52,750.00	€ 52,750.00	€ 52,750.00	€ 52,750.00	€ 52,750.00
Total costs	€ 62,829.26	€ 62,829.26	€ 62,829.26	€ 62,829.26	€ 70,375.00	€ 70,375.00	€ 70,375.00	€ 70,375.00	€ 70,375.00	€ 70,375.00	€ 70,375.00	€ 70,375.00
Benefits (per quarter)	€ 139,670.74	€ 315,920.74	€ 368,420.74	€ 462,170.74	€ 282,125.00	€ 447,125.00	€ 544,625.00	€ 709,625.00	€ 417,125.00	€ 567,125.00	€ 747,125.00	€ 979,625.00

- Explanations:
    - In case of pessimistic scenario we are paying the Bank loan + taxes;
    - In case of pessimistic scenario we are spending less on equipments;
    - In case of pessimistic scenario we are spending less on Salaries (just 3 of us + full stack + 1 sales manager);
    - In case of realistic scenario all variables are the same as in the cashflow;
    - In the optimistic scenario and realistic, we are spending a little more on salaries (3 of us + fullstack + 2 sales manager, everyone fulltime)

3. Way to deal with the negative cashflow

Following the feedback that we needed a way to deal with the negative cashflow, we have include a Crowd Funding project, in which we accomplished to gather **60,000 Euros**. We have made research about some crowd funding platforms like Kickstarter, Indiegogo, etc and decided that our project fits the target amount expectation successfully.

All values are already updated in the cashflow and in all financial factors (ROI, NPV, IRR, etc, you can find them in the main financial document on github).

3. Management part upgrades

Public App tasks:

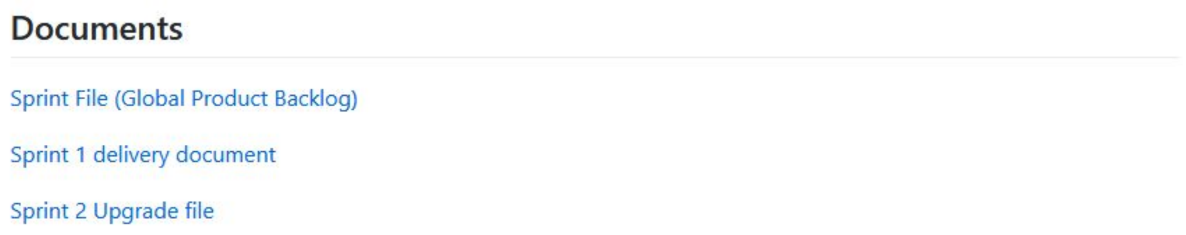
Adapt existing screens to work with the backend	Hard	Tom	Sprint 2
Retrieve shops posts from DB and display them	Medium	Tom	Sprint 2
Adapt financial document	Medium	Danillo/Tom	Sprint 2
Redux storage for language selection	Medium	Danillo/Tom	Sprint 2
Reduction list from backend	Medium	Tom	Sprint 2
Develop back-end and connection with the database	Easy	Danillo/Tom/Marcel	Sprint 2
Create local database	Easy	Danillo/Tom/Marcel	Sprint 2
Filter products through categories	Medium	Tom	Sprint 2
Retrieve images from the AWS bucket	Medium	Danillo	Sprint 2
Adapt front form for account creation	Easy	Tom	Sprint 2
Display error message if login fails	Easy	Tom	Sprint 2
Store JWT tokenisation within redux Store	Hard	Tom	Sprint 2
Retrieve User from db	Medium	Tom	Sprint 2
Create the AWS Bucket	Easy	Danillo	Sprint 2
Upload images to the aws bucket	Medium	Danillo	Sprint 2
Retrieve all posts from DB	Easy	Tom	Sprint 2
Retrieve shops from real Database datas	Medium	Tom	Sprint 2

Pro App Tasks:

Task	Difficult	Asignee	Sprint
Create Register / login Screen	Easy	Marcel	Sprint 2
Provide a smooth navigation through the app	Hard	Marcel	Sprint 2
List of all the posts related to the shop	Medium	Marcel	Sprint 2
List of pending discounts that the owner of a shop can monitor	Easy	Marcel	Sprint 2
List of all discounts that had already been used with usage date	Easy	Marcel	Sprint 2
Post a product to advertise it.	Medium	Marcel	Sprint 2
Provide different kind of roles and permissions (owner / employee)	Easy	Danillo	Sprint 2
Back-end data within the professional shop profile	Hard	Danillo	Sprint 2

3. Management Part

Regarding the management part off the work, a few changes have been made during this second sprint.
At first we decided to implement a more accurate Readme file for both public and pro applications.
Those Readme now cluster the main documents related with the different sprints.



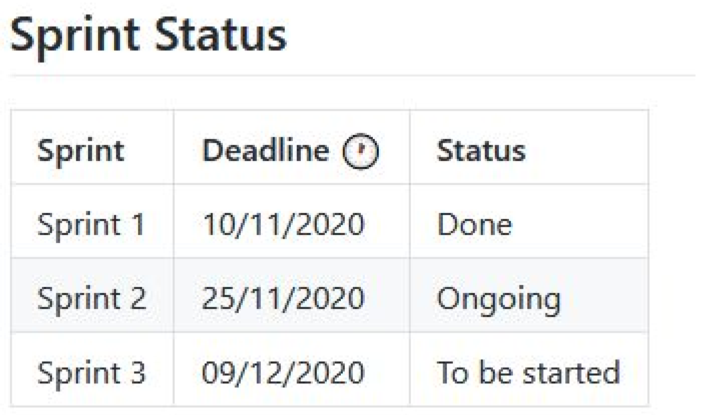
Documents

Sprint File (Global Product Backlog)

Sprint 1 delivery document

Sprint 2 Upgrade file

We also added a schedule to remind easily the deadlines of the three sprints :

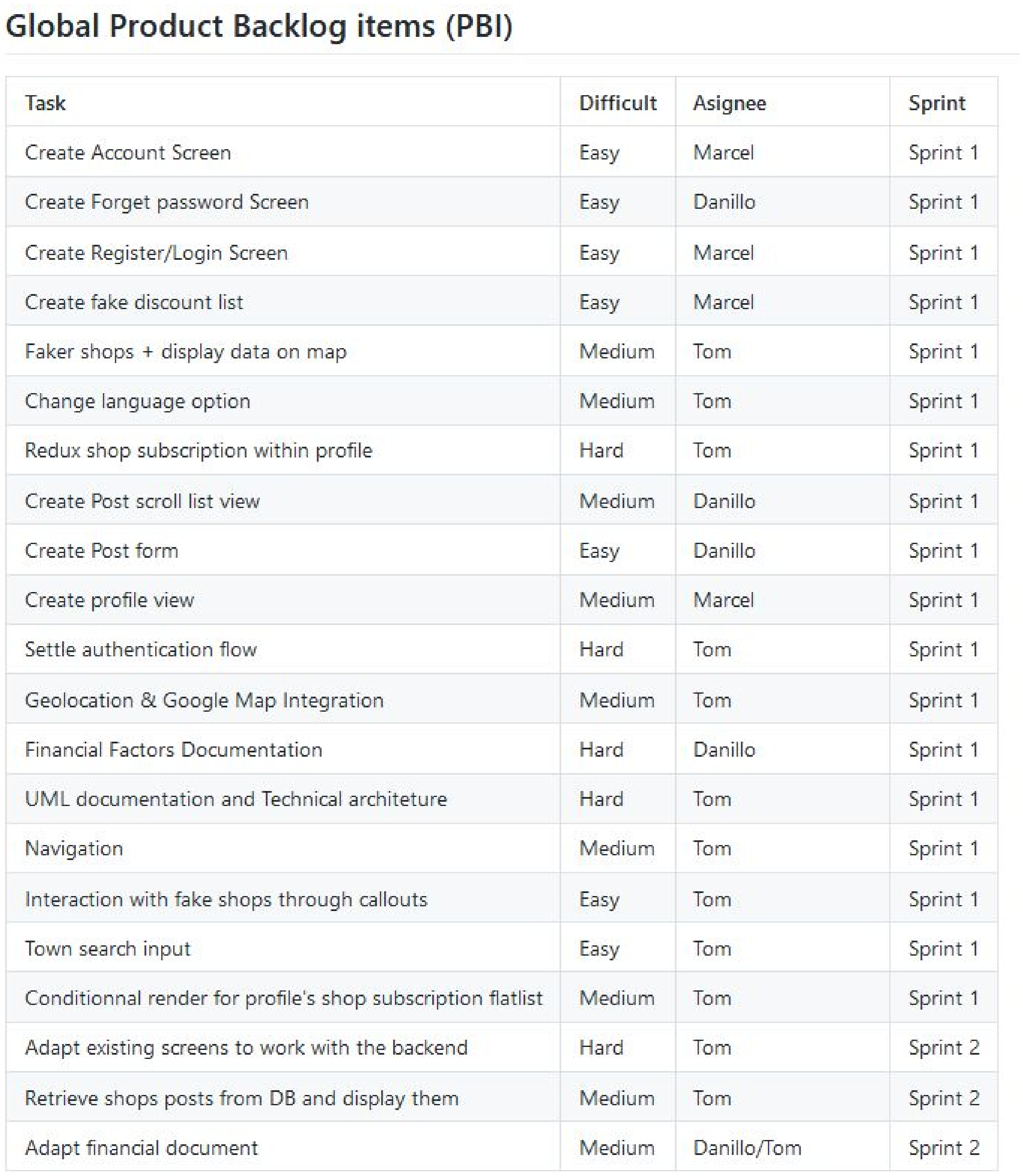


Sprint Status

Sprint	Deadline	Status
Sprint 1	10/11/2020	Done
Sprint 2	25/11/2020	Ongoing
Sprint 3	09/12/2020	To be started

Last but not least, we also wanted to include the global product backlog with all the tasks we have to implement during each sprint. This array also mentions the assignee and the difficulty of those tasks.

Here is a non-exhaustive part of the global product backlog for the public app



Global Product Backlog items (PBI)

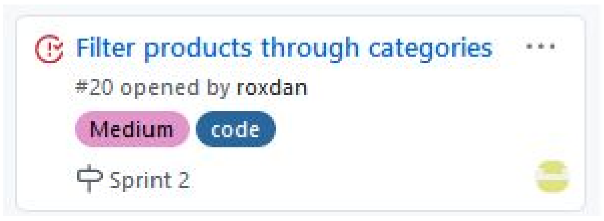
Task	Difficult	Asignee	Sprint
Create Account Screen	Easy	Marcel	Sprint 1
Create Forget password Screen	Easy	Danillo	Sprint 1
Create Register/Login Screen	Easy	Marcel	Sprint 1
Create fake discount list	Easy	Marcel	Sprint 1
Faker shops + display data on map	Medium	Tom	Sprint 1
Change language option	Medium	Tom	Sprint 1
Redux shop subscription within profile	Hard	Tom	Sprint 1
Create Post scroll list view	Medium	Danillo	Sprint 1
Create Post form	Easy	Danillo	Sprint 1
Create profile view	Medium	Marcel	Sprint 1
Settle authentication flow	Hard	Tom	Sprint 1
Geolocation & Google Map Integration	Medium	Tom	Sprint 1
Financial Factors Documentation	Hard	Danillo	Sprint 1
UML documentation and Technical architeture	Hard	Tom	Sprint 1
Navigation	Medium	Tom	Sprint 1
Interaction with fake shops through callouts	Easy	Tom	Sprint 1
Town search input	Easy	Tom	Sprint 1
Conditionnal render for profile's shop subscription flatlist	Medium	Tom	Sprint 1
Adapt existing screens to work with the backend	Hard	Tom	Sprint 2
Retrieve shops posts from DB and display them	Medium	Tom	Sprint 2
Adapt financial document	Medium	Danillo/Tom	Sprint 2

Regarding to the kanban's organization, we decided to keep

In the beginning of the sprint, we decide together which are the tasks that should be implemented regarding the teachers / clients expectation. During this second sprint, the main concern was linking the app to the back end part in order to inject reel data coming from the database inside the client application.

For each task we previously decided to implement, we create a related issue. This issue specifies the title and the description of the task. It also assigns the task to a member of the group. Labels are also affected to the issue to describe which category the task belongs to (code, bug, documentation) but also to give an expected level of difficulty / effort dedication to each task (easy - less than 3 hours | medium - between 3 and 5 hours | hard - more than 5 hours).

Here is a closer look at one of the issues we have created for this second sprint.

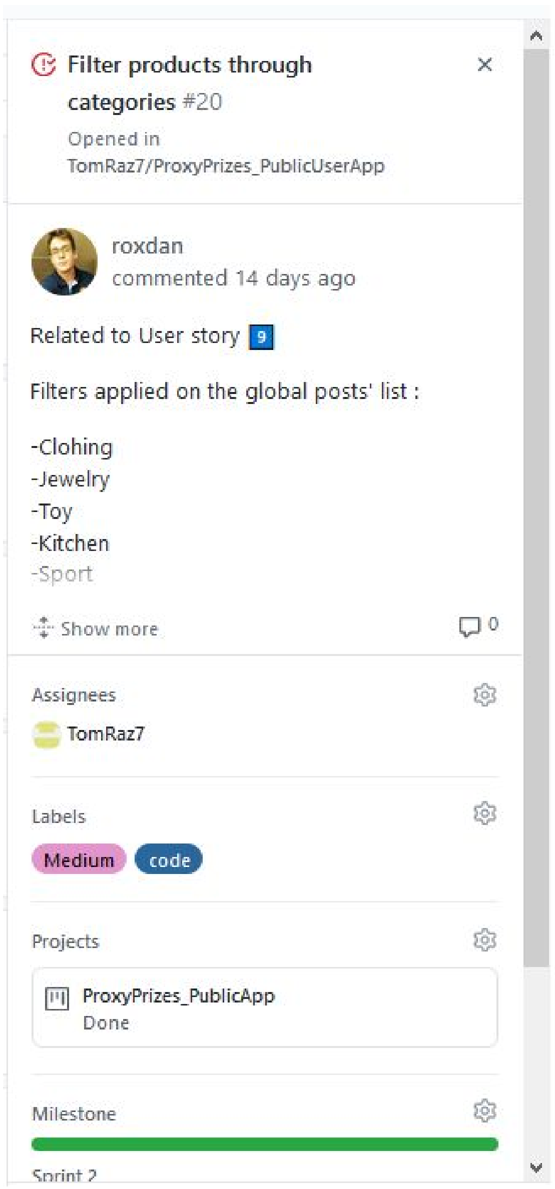


Filter products through categories

#20 opened by roxdan

Medium code

Sprint 2



Filter products through categories #20

Opened in TomRaz7/ProxyPrizes_PublicUserApp

roxdan commented 14 days ago

Related to User story 9

Filters applied on the global posts' list :

-Clohing
-Jewelry
-Toy
-Kitchen
-Sport

Assignees: TomRaz7

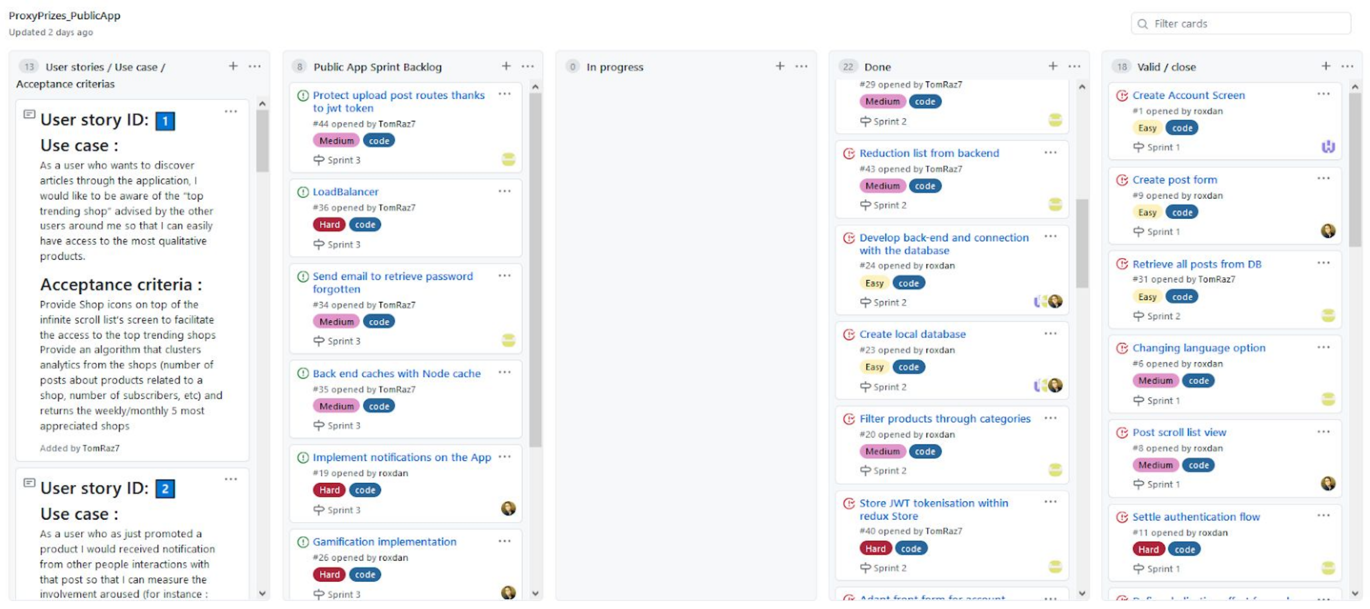
Labels: Medium code

Projects: ProxyPrizes_PublicApp Done

Milestone: Sprint 2

We also linked each issue with a specific user story as it's possible to see on the upper screenshot.

We then drag and drop those different issues into the columns of the kanban to track and monitor the evolution of the project during the sprint.



ProxyPrizes_PublicApp
Updated 2 days ago

Filter cards

13 User stories / Use case / Acceptance criterias

User story ID: 1

Use case :

As a user who wants to discover articles through the application, I would like to be aware of the "top trending shop" advised by the other users around me so that I can easily have access to the most qualitative products.

Acceptance criteria :

Provide Shop icons on top of the infinite scroll list's screen to facilitate the access to the top trending shops
Provide an algorithm that clusters analytics from the shops (number of posts about products related to a shop, number of subscribers, etc) and returns the weekly/monthly 5 most appreciated shops

Added by TomRaz7

User story ID: 2

Use case :

As a user who as just promoted a product I would received notification from other people interactions with that post so that I can measure the involvement aroused (for instance :

8 Public App Sprint Backlog

Protect upload post routes thanks to jwt token
#44 opened by TomRaz7
Medium code
Sprint 3

LoadBalancer
#36 opened by TomRaz7
Hard code
Sprint 3

Send email to retrieve password forgotten
#34 opened by TomRaz7
Medium code
Sprint 3

Back end caches with Node cache
#35 opened by TomRaz7
Medium code
Sprint 3

Implement notifications on the App
#19 opened by roxdan
Hard code
Sprint 3

Gamification implementation
#26 opened by roxdan
Hard code
Sprint 3

0 In progress

22 Done

#29 opened by TomRaz7
Medium code
Sprint 2

Reduction list from backend
#43 opened by TomRaz7
Medium code
Sprint 2

Develop back-end and connection with the database
#24 opened by roxdan
Easy code
Sprint 2

Create local database
#23 opened by roxdan
Easy code
Sprint 2

Filter products through categories
#20 opened by roxdan
Medium code
Sprint 2

Store JWT tokenisation within redux Store
#40 opened by TomRaz7
Hard code
Sprint 2

18 Valid / close

Create Account Screen
#1 opened by roxdan
Easy code
Sprint 1

Create post form
#9 opened by roxdan
Easy code
Sprint 1

Retrieve all posts from DB
#31 opened by TomRaz7
Easy code
Sprint 2

Changing language option
#6 opened by roxdan
Medium code
Sprint 1

Post scroll list view
#8 opened by roxdan
Medium code
Sprint 1

Settle authentication flow
#11 opened by roxdan
Hard code
Sprint 1

Each time a specific task is terminated and pushed on the repository, we examine it and, if it fits the task's expectations, we close the related issue and move it to the "Done" part.

After presenting the work done during the sprint, we wait for the teachers' remarks and feedback to move the issues to the "Valid / close" column or to reopen the issue if it needs to be modified.