Universitat de Lleida

Sprint 1

MINF UDL 20-21

TI Project Management

Team ProxiPrizes

This document clusters the whole product backlog but also technical and economical explanations about the project.

ProxyPrizes - Product Backlog

User types :

- Shops
    - Owners / managers of the shop
    - Employees of the shops

- Clients
    - Clients who wants to promote
    - Clients who wants to discover products

Users stories :

Customers

As a user of the application, I would like to monitor all the discounts from which I benefit so that I can organize my future purchases.

- Acceptance criteria :
    - Create a discounts' list screen within the user's profile part to show all his discounts
    - For each discount, display the expiration date

As a user of the application, I would like to be notified when I received a new discount from a shop so that I don't miss opportunities to save money.

- Acceptance criteria :
    - Include push notification within the app

As a user of the application I would like to use my discounts directly within the shops so that I can perceive my discount

- Acceptance criteria :
    - Provide a "use discount" button within the client app
    - Provide a notification system from the client app to the customer app / the web app
    - Provide a "accept discount usage " button within the pro app
    - Change the status of the discount from pending to accepted / used

As a customer, I would be able to rely on other users' recommendations on the platform so that I can discover new articles through the application.

- Acceptance criteria :
    - Provide an infinite scroll list of users' posts that promotes articles

As a customer who wants to discover articles through the application, I would be able to filter products thanks to categories so that I can search accurately and save time.

- Acceptance criteria :
    - Provide filter buttons on top of the infinite scroll list’s screen

As a user of the application, I would like to register / log in easily thanks to my facebook / instagram account, so that I can access the application.

- Acceptance criteria :
    - Provide a password input
    - Provide a email/username input
    - Provide a “password forgotten” button to recover a password
    - Provide a login button
    - Provide a create account button

As a customer who wants to discover articles through the application, I would be able to subscribe to my favourite stores so that I won’t miss new products/posts from this shop.

- Acceptance criteria :
    - Provide a subscribe button
    - Provide a list of the shops a user has subscribed to within the profile screen

As a customer, I would like to be able to promote a product that I like easily so that I can perceive a discount from the shop.

- Acceptance criteria :
    - Generate a post form
        - an input for the title
        - a way to upload picture
        - an input for the description
        - an input for the price of the product
        - name/ location of the shop the article is coming from
        - the logo of the shop

As a customer whose post has been rejected, I would like the system to give me the reason and to advise me so that my next posts will be accepted.

- Acceptance criteria :
    - Display a clear message to explain to the user the reason why his/her post has been rejected (for instance bad language, poor resolution picture , too short description of the product, etc)

As a user who wants to discover articles through the application, I would be able to save / add to my favourite article from other users’ posts so that I can retrieve it easily.

- Acceptance criteria :
    - Provide a “add to favourite”/ heart button to add a post to the user’s favourite
    - Provide a list of the posts a user has added to his favourite within the profile screen

As a user who wants to discover articles through the application, I would like to know if an article I saw on a post is still available at the shop so that I would know the products I can purchase.

- Acceptance criteria :

    - Provide a “I Ask for availability” button which would send a notification to the related shop
    - Provide an answer from the shop (not available / available + quantity)

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 : somebody has just added your post to his favorites)

- Acceptance criteria :
    - Provide a notification systems related with the posts

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

Shops

As the owner of a shop I would like to monitor pending discounts easily (= discounts that have not been used already) so that I can keep track of the pending discounts I will need to offer.

- Acceptance criteria :
    - Provide a list of pending discounts related to a shop with the expiration date for each discount

As the owner of a shop, I would like to have a history of discount vouchers used so that I can keep track of the vouchers I offered.

- Acceptance criteria :
    - Provide a list of pending discounts related to a shop with the date of usage for each discount

As the owner of a shop I would like to advertise my products (create a post) and have statistics about people reached so that I can feed my profile.

- Acceptance criteria :
    - Generate a post form
        - an input for the title
        - a way to upload picture
        - an input for the description
        - an input for the price of the product
        - name/logo of the shop the article is coming from

As a shop owner I would be able to federate my community by organizing events so that I can gather a community around my activities.

- Acceptance criteria :
  - Provide an event form to organize a private event for subscribers
    - a date
    - a title
    - a description
    - the value of an additional discount during this day
    - retrieve all the subscribers

Features to implement

Client :

- Register / log in (with facebook maybe latter)
- Provide a smooth navigation through the app
- Interactive map (with callouts) with geolocated shops
- Store user's credential with local storage
- Display a discounts' list to cluster all the discounts
- Infinite scroll list of the other users posts
- Firebase notification
- Subscription to a shop
- Create a post
- Navigate through the application
- Filter the products through categories
- Have access to a profile
- Add another user's post to your favorite (to cluster all your favourite products)
- Purchase products within the app
- Advise daily/weekly or monthly top trending shops
- A shop profil would cluster all the post of the other users related to this shop and the shop's promotional posts
- Gamification part : base the discounts on a gamification process. The users would enhance their discount rate by winning trophies (for instance gathering a 100 likes above one of their posts, gathering a certain number of "add to favourites" on their post, etc.) The trophies that allow to enhance the user's percentage discount would be related to a specific shop.

Optional feature : Provide a tutorial for a brand new user to explain him/her how the app works

Shops:

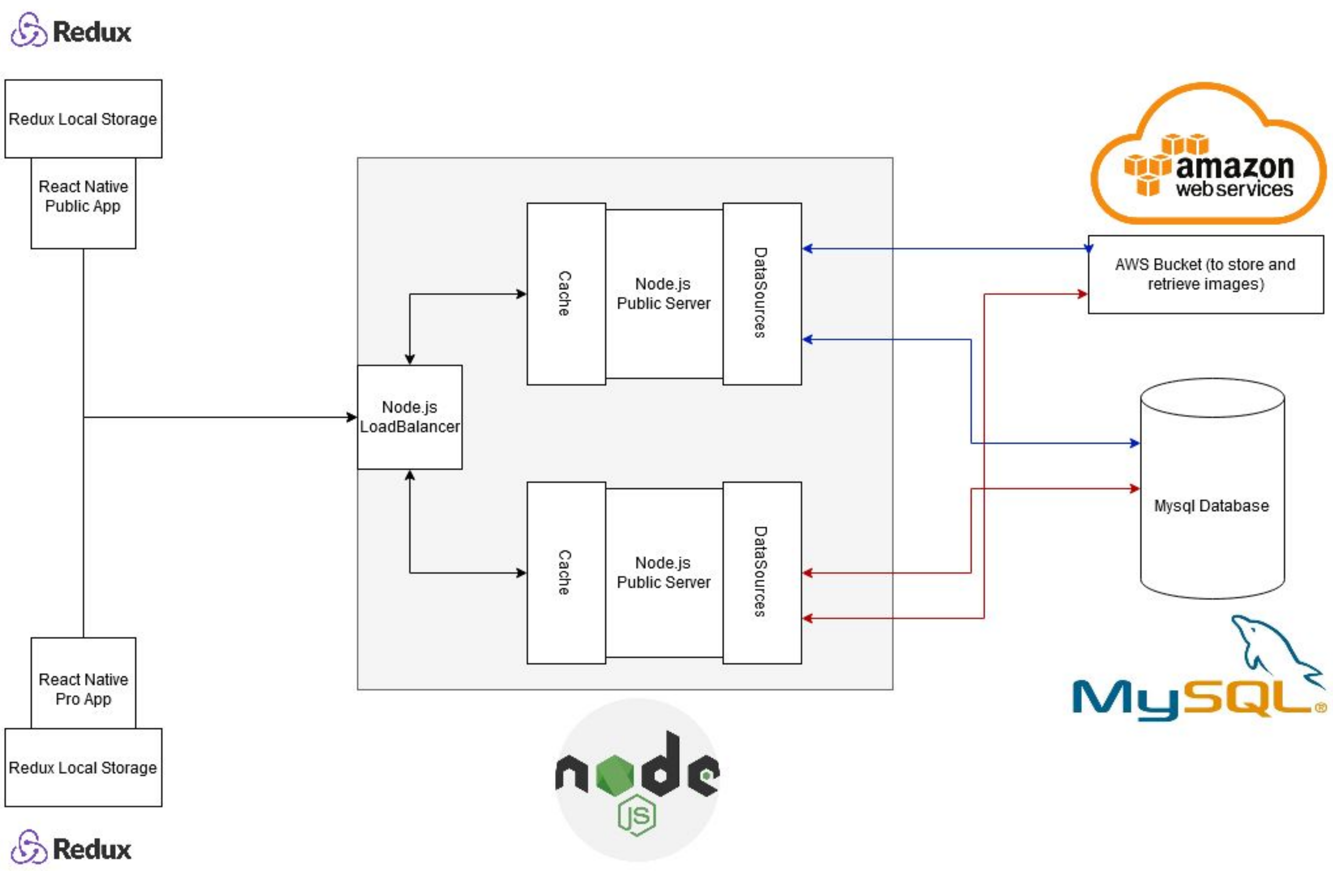
- Register / log in (with facebook maybe latter)
- Provide a smooth navigation through the app
- Set a default "two weeks" expiration time for the discounts allow the shops to customize their expiration date
- List of all the posts related to the shop
- List of pending discounts that the owner of a shop can monitor
- List of all discounts that had already been used with usage date

- Have access to accurate figures to measure involvement around a post related with a product of the shop
- Post a product to advertise it.
- Back end datas within the professional shop profile (subscribers number, etc.)
- Organize contests
- Receive notifications about a product's availability and answer to the user with an optional quantity
- Provide different kind of roles and permissions (owner / employee)

Technological part :

System's architecture

This part of the document aims to present and justify the technical aspects of our App.
We will first introduce the general architecture of our App and also present the UML before diving further into some coding explanation that seems relevant for a better understanding of the app (navigation, local storage, etc.).



The previous picture introduces the general architecture of the system.
The system is divided in three main categories :

1. The front end part
2. The server side
3. The storage part

The front end part :

As we explained during the meetings, we decided to select the framework **react native** to handle the front part of our App. We coupled this framework with an **“Expo” solution**. Expo allows developers to host their React Native project on their own Expo servers in order to facilitate the development of the app. In fact, Expo solution provides the developers and application to have real time rendering on a physical device which helps to fasten the development of the app.

As seen on the diagram, our model relies on a double sided project. On the one hand, we must provide a customer App so that the users can post articles, get discounts and benefits from the shops, but also interact with the others' posts. On the other hand, we will have to provide a “Pro” app, dedicated to the owners of the partner shops in order to provide them monitoring functionalities as explained in our use cases.

Within this front end part of the architecture, we also wanted to provide local storage to retrieve easily some useful datas. For instance, it allows the client app to cache some relevant information such as the user's credential, so that the user doesn't need to log in each time he closes the application. That is when redux comes into play. Redux could be seen as a session object that enables to store data which would later be accessible from every part of the app. Coupled with the redux-persist library, it ensures a local persistence even after the app has been shutdown.

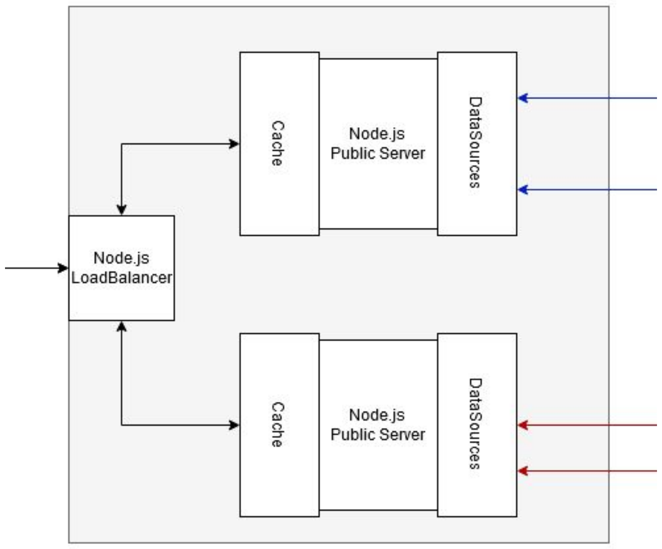
To summarize the front end part :

React Native

Expo

Redux

The server / back end part :



Our backend server part would rely on Node.js. This enables us to use javascript for booth front-end and back-end parts which can really be time saving.
We also would like to provide a scalable back-end architecture that could support requests load. We will may not have enough time to implement all this back-end architecture but we wanted to provide a diagram that tries to fit with professional expectations. As we have a two sided client part, we thought it could be interesting to implement a load balancer to redirect the requests coming from a customer app to a dedicated “Public” server whereas the requests coming coming from a pro app would be redirected to a “Pro” server. This solution could be interesting to improve our system’s throughput. To reduce the system’s latency it could also be interesting to provide server caches to decrease the number of database requests. Not only implementing a load balancer could smooth the back-end traffic but it could also be useful for security purposes as it can prevent the system from facing denial of service attacks.
We also illustrated the fact that we will provide data-sources to handle the database connexion. Here again, it is really important not to expose the database configuration information for security purposes and that is what justifies the need for environment and data-sources files.

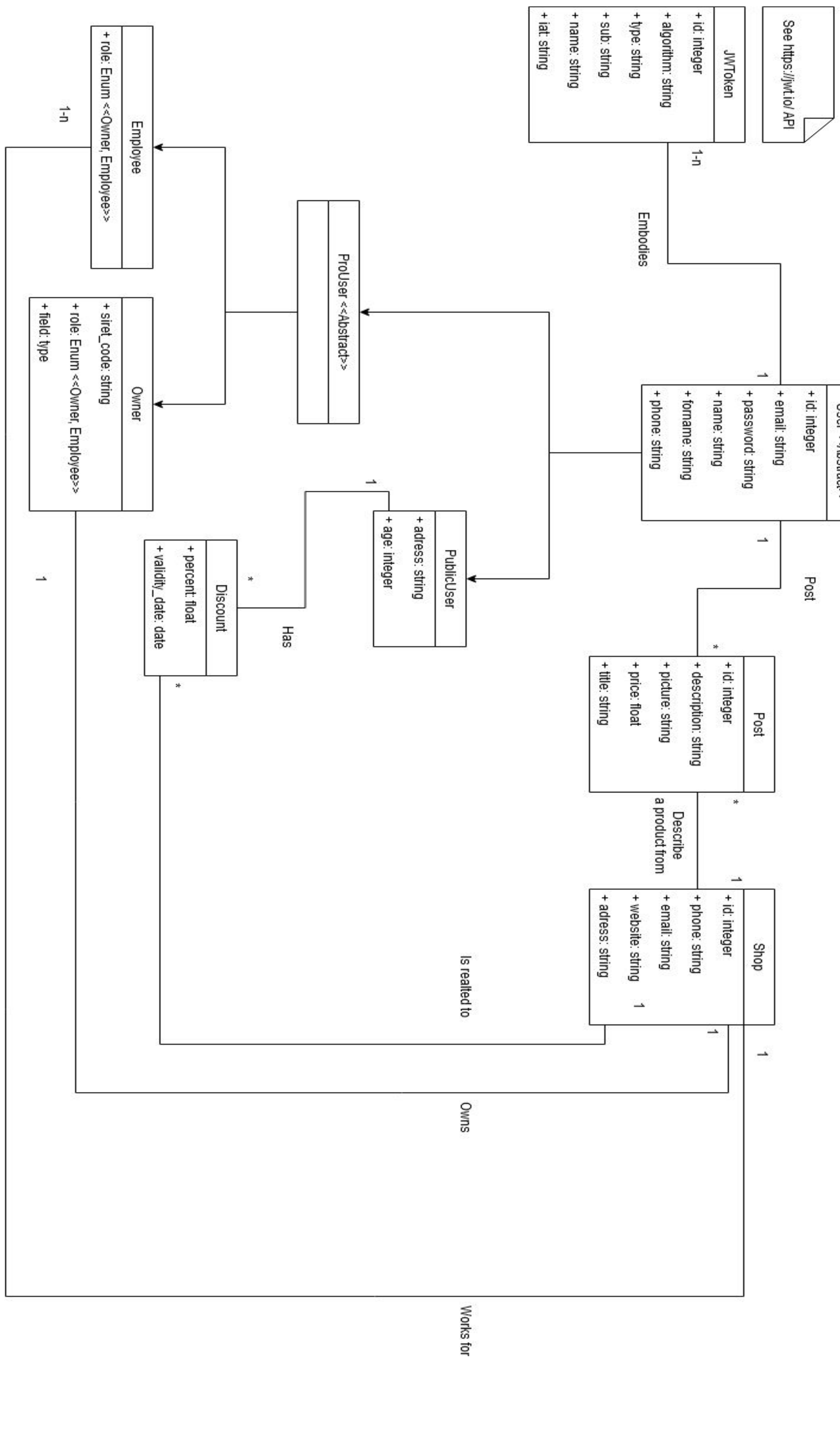
The storage part :

As we were more used to working with relational databases we naturally thought to include a MySql database within our project.

Dealing with the specific topic of image storage (pictures for the users posts, pictures for the user profile, pictures for the shops profile, etc.) different strategies were conceivable. The first strategy consists in storing the entire file within the database. Even if this approach is pretty simple, it generates an important amount of storage load which can dramatically impact the system's latency. In a real-world project, this would not be the best approach as it could provide a bad user experience with latency and delay. To answer this problem, we decided to use an AWS bucket to upload and retrieve pictures of the app easily. For each uploaded picture, the AWS service generates a signed urls which will allow us to make https request to retrieve the pictures. In this way, we will only have to store the picture's url link within our database in order to retrieve and display the pictures in the client applications.

After having specified our system's architecture, the next page will present you its UML diagram.

UML Diagram :



See https://jwt.io/ API
JWToken
+ id: integer
+ algorithm: string
+ type: string
+ sub: string
+ name: string
+ iat: string
1-n
Embodies
1
User <<Abstract>>
+ id: integer
+ email: string
+ password: string
+ name: string
+ forname: string
+ phone: string
ProUser <<Abstract>>
Employee
+ role: Enum <<Owner, Employee>>
Owner
+ siret_code: string
+ role: Enum <<Owner, Employee>>
+ field: type
PublicUser
+ adress: string
+ age: integer
1
Has
*
Discount
+ percent: float
+ validity_date: date
1
Post
*
Post
+ id: integer
+ description: string
+ picture: string
+ price: float
+ title: string
*
Describe
a product from
1
Shop
+ id: integer
+ phone: string
+ email: string
+ website: string
+ adress: string
*
1
Is realted to
1
Owns
1
1
1-n
Works for

Good security practices :

This part is a small digression about good security practices that we will try to implement within our code :

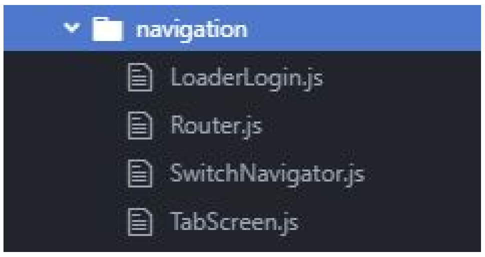
- Hash the passwords within the DB
- LoadBalancer to avoid denial of service attack
- Parameterized queries to avoid SQL injections
- Use tokenization (like json web tokens)
- Store the app global information within a .env file and add it to a .gitignore file to prevent sensitive information (like DB password, severs port, etc) from being committed to the GitHub repository

Few app coding explanations :

This part aims to explain a little bit more the operation of key features of the app. The next paragraphs will focus on explaining the setting up of the navigation and the local storage of the application.

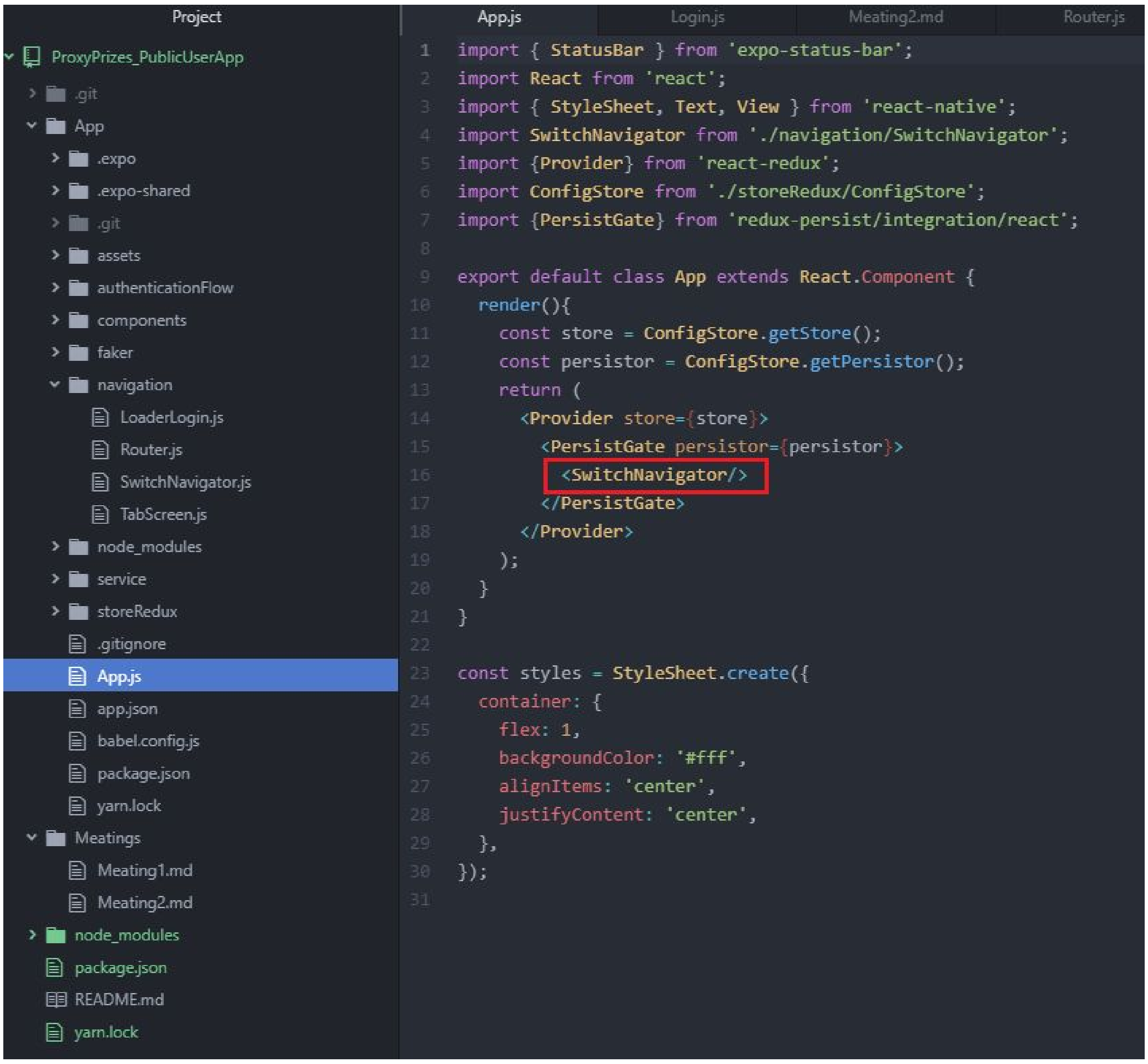
App Navigation :

The navigation of the app relies on 4 main files :



```
navigation
    LoaderLogin.js
    Router.js
    SwitchNavigator.js
    TabScreen.js
```

As you can see below, the app’s entry point leads to the SwitchNavigator File

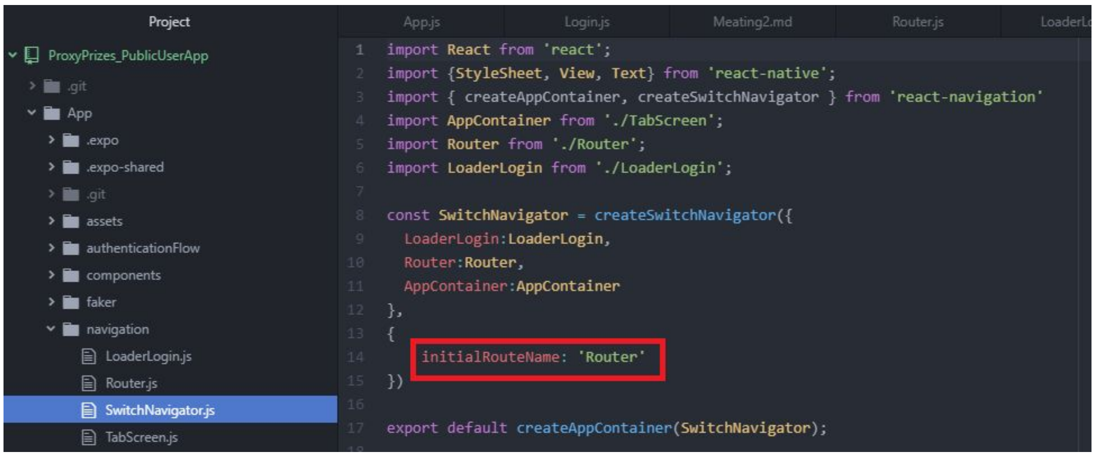


```
import { StatusBar } from 'expo-status-bar';
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';
import SwitchNavigator from './navigation/SwitchNavigator';
import {Provider} from 'react-redux';
import ConfigStore from './storeRedux/ConfigStore';
import {PersistGate} from 'redux-persist/integration/react';

export default class App extends React.Component {
  render(){
    const store = ConfigStore.getStore();
    const persistor = ConfigStore.getPersistor();
    return (
      <Provider store={store}>
        <PersistGate persistor={persistor}>
          <SwitchNavigator/>
        </PersistGate>
      </Provider>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

The SwitchNavigator aims to balance the navigation between the 3 other navigation files :

- The Router.js
- The LoaderLogin.js
- The Tabscreen.js
-

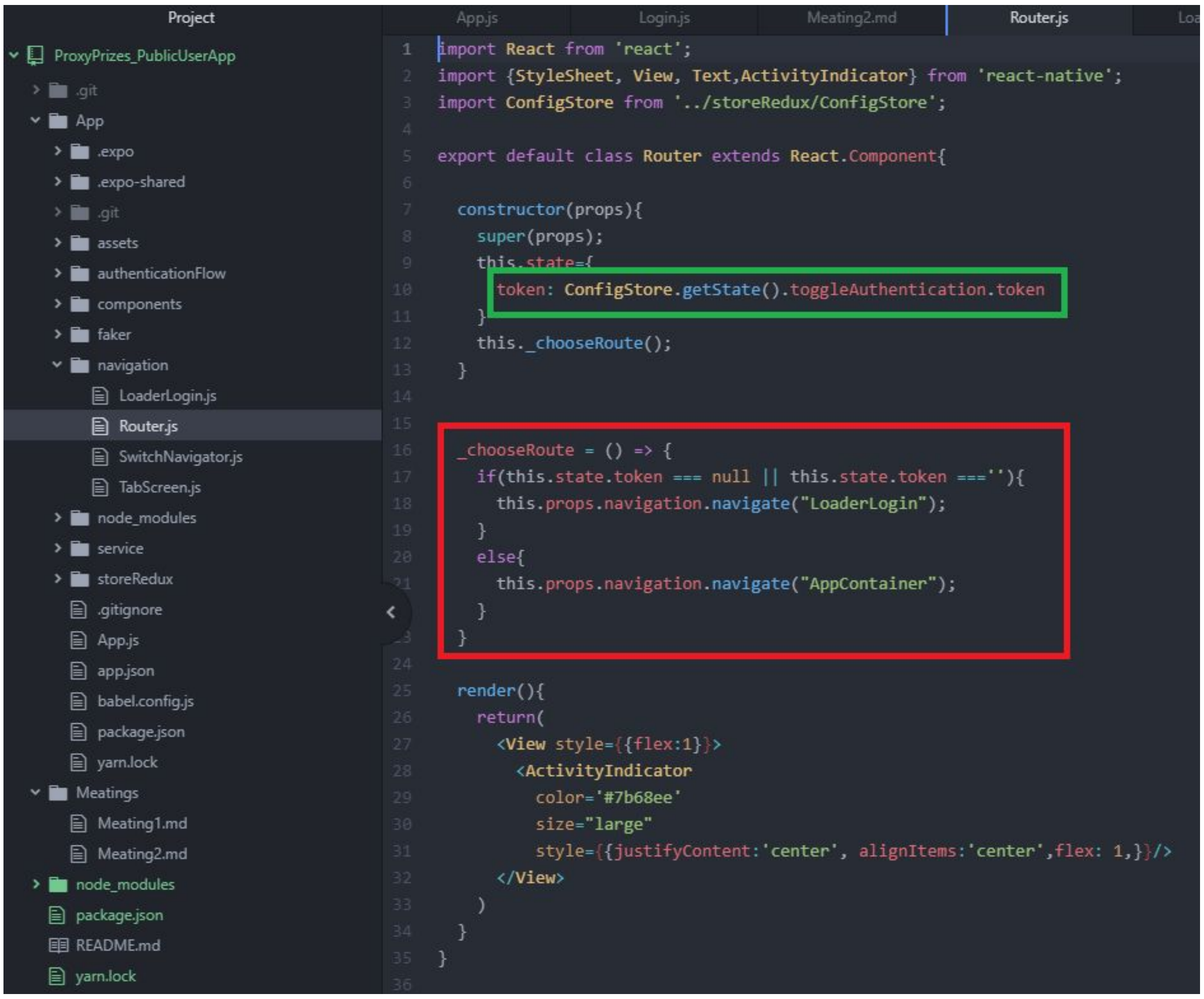
The initial route of the app leads to the Router component :



```
import React from 'react';
import {StyleSheet, View, Text} from 'react-native';
import { createAppContainer, createSwitchNavigator } from 'react-navigation'
import AppContainer from './TabScreen';
import Router from './Router';
import LoaderLogin from './LoaderLogin';

const SwitchNavigator = createSwitchNavigator({
  LoaderLogin:LoaderLogin,
  Router:Router,
  AppContainer:AppContainer
},
{
    initialRouteName: 'Router'
})

export default createAppContainer(SwitchNavigator);
```

As you can see in the import part of the code, the AppConrtainer variable corresponds to an instance of the Tabscreen component. Remember this component, I will speak about it later.

The Router aims to balance the navigation between the App container and the login part of the app :



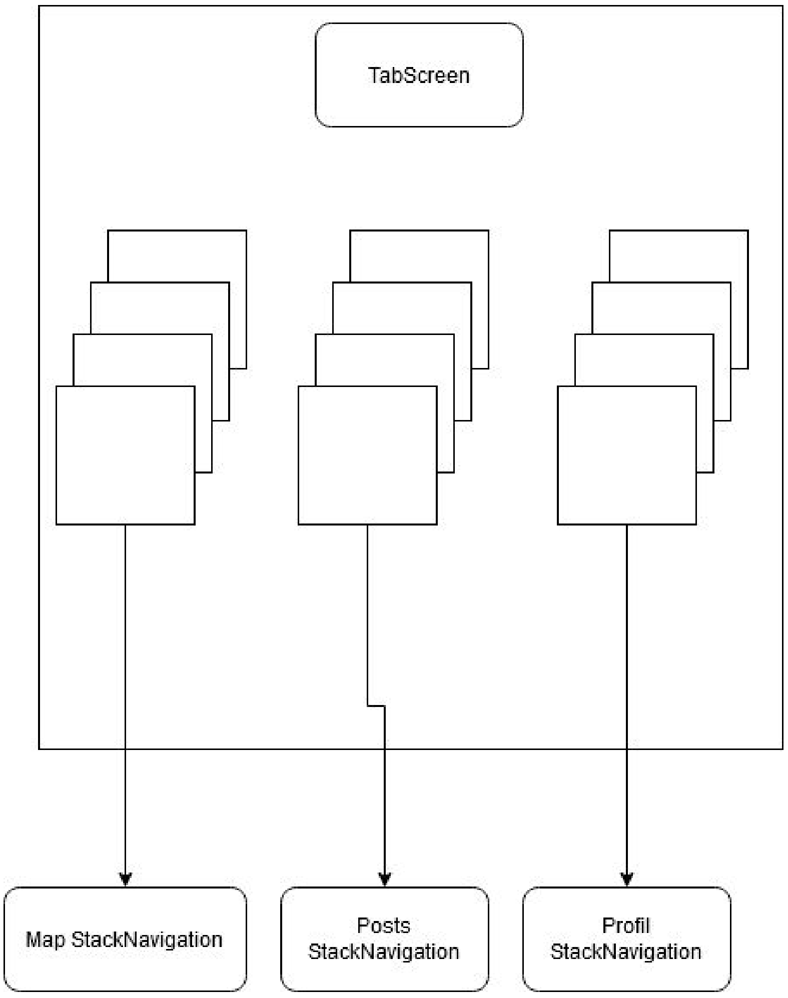
```
import React from 'react';
import {StyleSheet, View, Text,ActivityIndicator} from 'react-native';
import ConfigStore from '../storeRedux/ConfigStore';

export default class Router extends React.Component{

  constructor(props){
    super(props);
    this.state={
      token: ConfigStore.getState().toggleAuthentication.token
    }
    this._chooseRoute();
  }

  _chooseRoute = () => {
    if(this.state.token === null || this.state.token ===''){
      this.props.navigation.navigate("LoaderLogin");
    }
    else{
      this.props.navigation.navigate("AppContainer");
    }
  }

  render(){
    return(
      <View style={{flex:1}}>
        <ActivityIndicator
          color='#7b68ee'
          size="large"
          style={{justifyContent:'center', alignItems:'center',flex: 1,}}/>
      </View>
    )
  }
}
```

As you can see in the above picture, the Router component tries to retrieve a previously stored token from the redux local storage. Then, depending on the existence of this token, it will use the _chooseRoute function to navigate directly within the app (Appcoontainer) or to display the login part of the app (LoaderLogin).

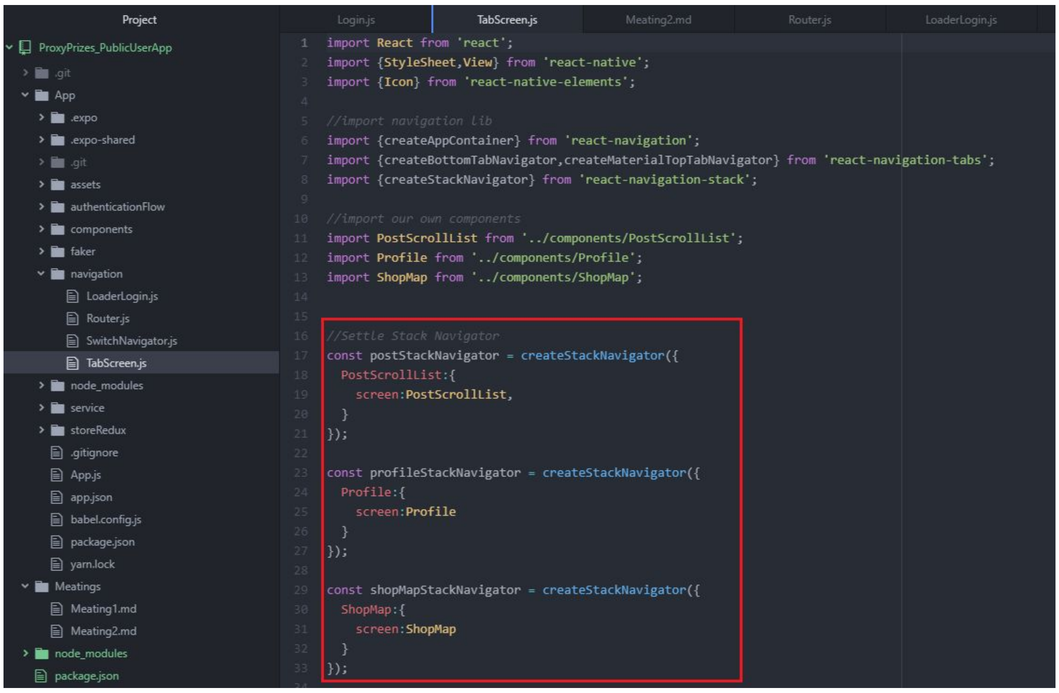
Let's firstly dive into the Appcontainer organization :

As previously explain, this Appcontainer is an instance of the TabScreen component.

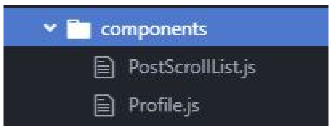
The TabScreen component enable to package our three main part of the app (Map / Posts list / Profile) into a single horizontal navigation.



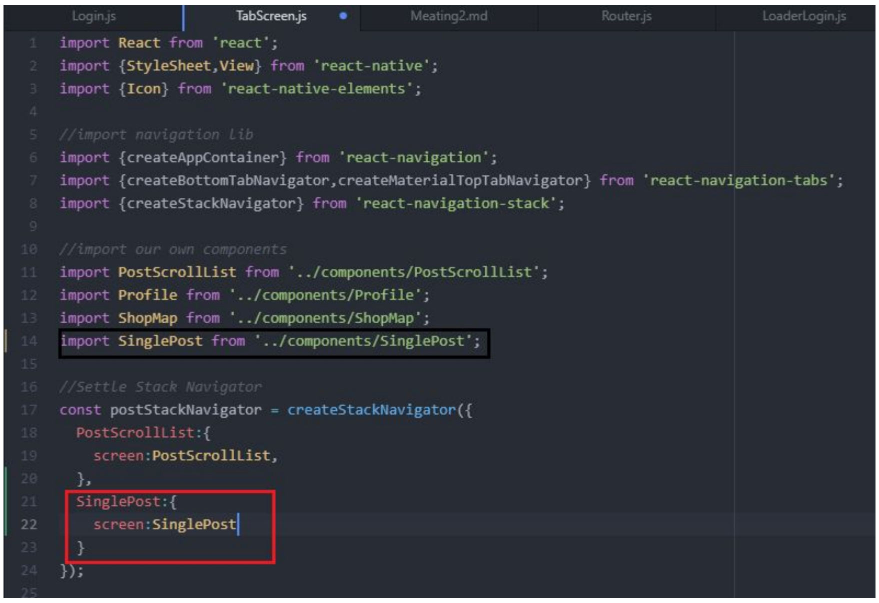
Here are the three navigations stacks :



```
import React from 'react';
import {StyleSheet,View} from 'react-native';
import {Icon} from 'react-native-elements';

//import navigation lib
import {createAppContainer} from 'react-navigation';
import {createBottomTabNavigator,createMaterialTopTabNavigator} from 'react-navigation-tabs';
import {createStackNavigator} from 'react-navigation-stack';

//import our own components
import PostScrollList from '../components/PostScrollList';
import Profile from '../components/Profile';
import ShopMap from '../components/ShopMap';


//Settle Stack Navigator
const postStackNavigator = createStackNavigator({
  PostScrollList:{
    screen:PostScrollList,
  }
});

const profileStackNavigator = createStackNavigator({
  Profile:{
    screen:Profile
  }
});

const shopMapStackNavigator = createStackNavigator({
  ShopMap:{
    screen:ShopMap
  }
});
```



Each time a new component is created, it has to be linked with its related stack navigator. In the following picture, a SinglePost component is added to the post stack navigator.

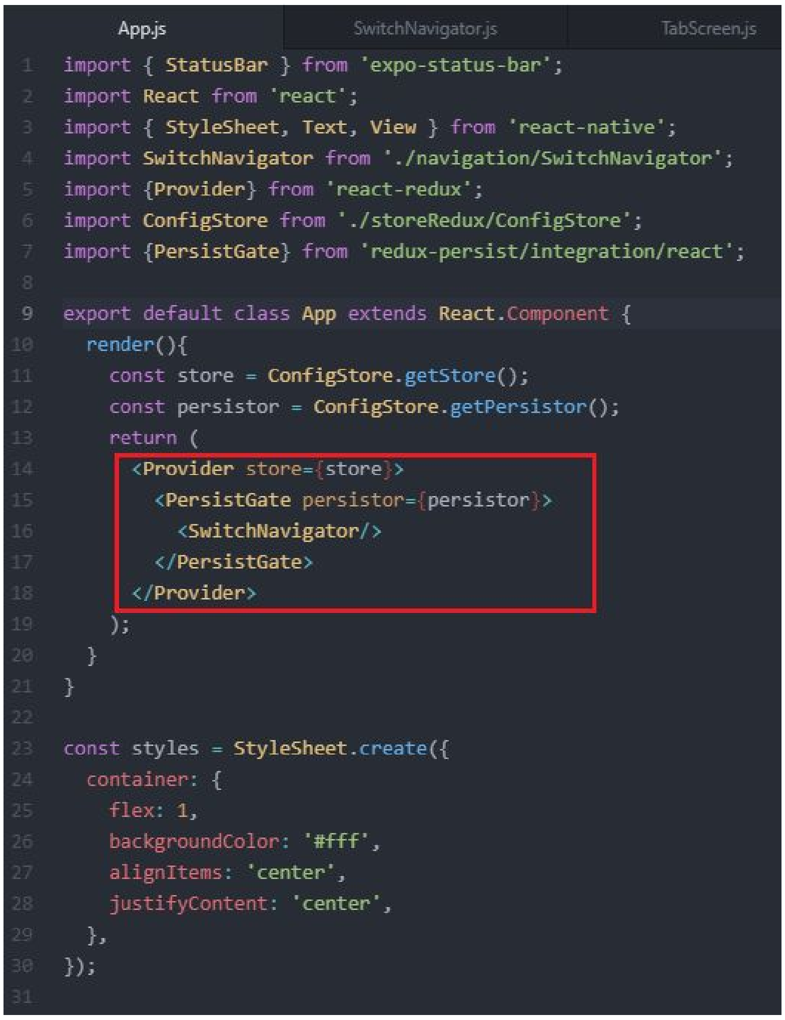


```
import React from 'react';
import {StyleSheet,View} from 'react-native';
import {Icon} from 'react-native-elements';

//import navigation lib
import {createAppContainer} from 'react-navigation';
import {createBottomTabNavigator,createMaterialTopTabNavigator} from 'react-navigation-tabs';
import {createStackNavigator} from 'react-navigation-stack';

//import our own components
import PostScrollList from '../components/PostScrollList';
import Profile from '../components/Profile';
import ShopMap from '../components/ShopMap';
import SinglePost from '../components/SinglePost';

//Settle Stack Navigator
const postStackNavigator = createStackNavigator({
  PostScrollList:{
    screen:PostScrollList,
  },
  SinglePost:{
    screen:SinglePost
  }
});
```

Local Storage :

This part aims to give much details on the Redux persistence part of the application. As previously explained, Redux could be seen as a session object that enables to store data which would later be accessible from every part of the app. Coupled with the redux-persist library, it ensures a local persistence even after the app has been shutdown.

To do so, the app root file should be wrapped into two different components. The Provider component allows Redux to encompass the app and to provide a meta access-data point from every single component of the app. The PersistGate component ensures the local persistence of the information.



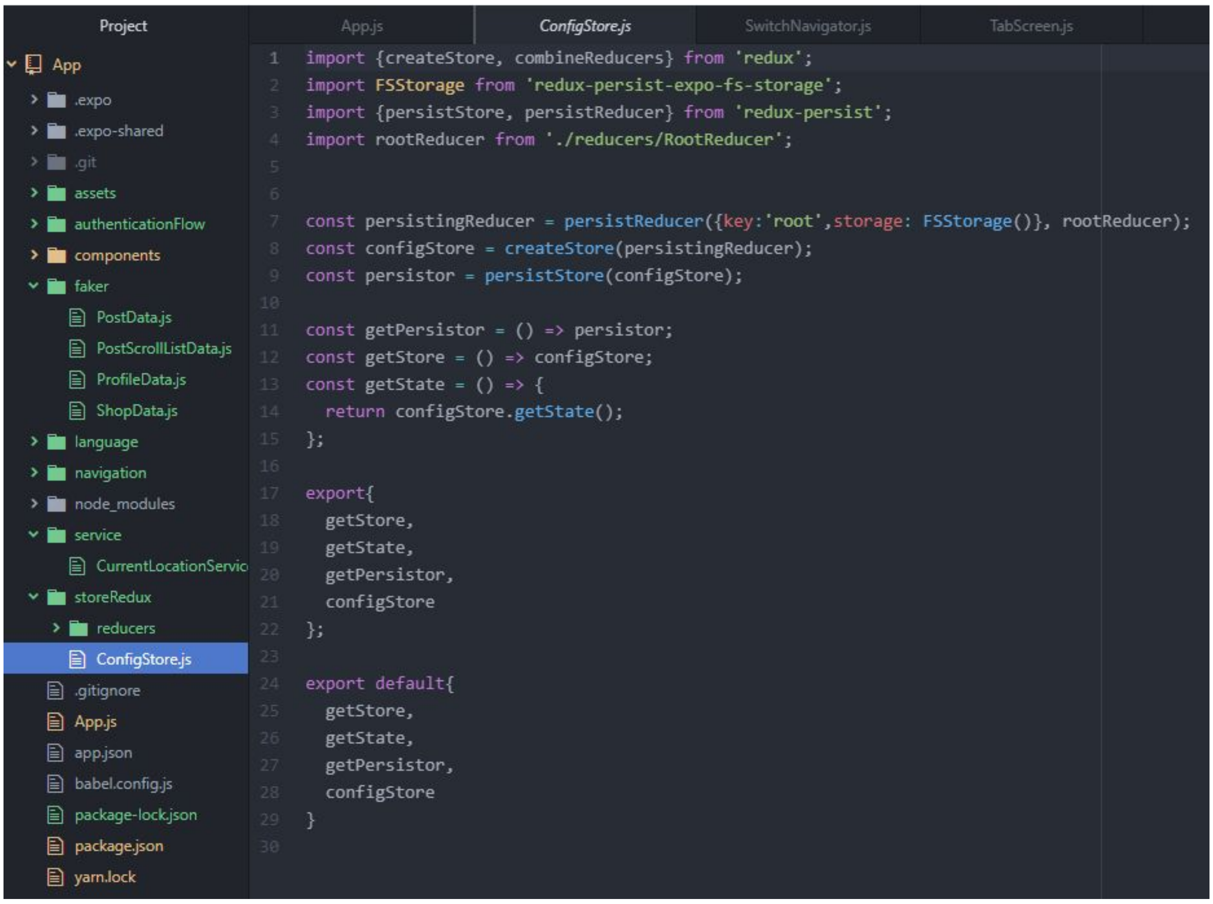
```
import { StatusBar } from 'expo-status-bar';
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';
import SwitchNavigator from './navigation/SwitchNavigator';
import {Provider} from 'react-redux';
import ConfigStore from './storeRedux/ConfigStore';
import {PersistGate} from 'redux-persist/integration/react';

export default class App extends React.Component {
  render(){
    const store = ConfigStore.getStore();
    const persistor = ConfigStore.getPersistor();
    return (
      <Provider store={store}>
        <PersistGate persistor={persistor}>
          <SwitchNavigator/>
        </PersistGate>
      </Provider>
    );
  }
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

As it is possible to see in the import part of the above code, another key element of the redux setting up is the ConfigStore component.

The following code belongs to the ConfigStore component. It works as a configuration file for the redux part.

In this code, we first instantiate a persistReducer class. Within the constructor of this class, we pass an object that contains the storage method (here using FSStorage for local storage). We also pass a “rootReducer” which clusters all the reducers of the app.

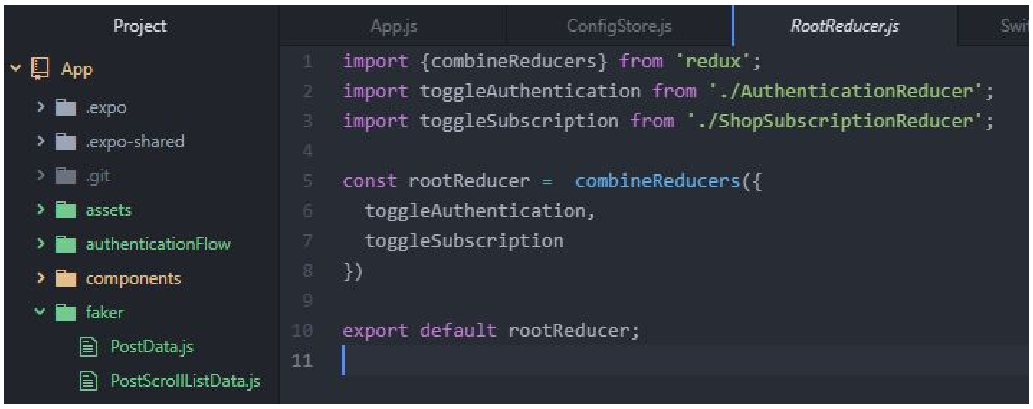
We then create an instance of a redux store with the createStore function, passing the persistReducer created as a parameter of the function.



```
import {createStore, combineReducers} from 'redux';
import FSStorage from 'redux-persist-expo-fs-storage';
import {persistStore, persistReducer} from 'redux-persist';
import rootReducer from './reducers/RootReducer';


const persistingReducer = persistReducer({key:'root',storage: FSStorage()}, rootReducer);
const configStore = createStore(persistingReducer);
const persistor = persistStore(configStore);

const getPersistor = () => persistor;
const getStore = () => configStore;
const getState = () => {
  return configStore.getState();
};

export{
  getStore,
  getState,
  getPersistor,
  configStore
};

export default{
  getStore,
  getState,
  getPersistor,
  configStore
}

```

Redux works upon reducers. A reducer is a function which aims to store and modify a global state. Whereas traditional components encompass their own state within their constructor, the state elements gathered within redux-reducers will be accessible from every single point of the app.
A single reducer function is responsible for the modification and the update of a unique state element.
All the reducers are clustered within a rootReducer which is passed to the persistReducer (as previously shown) so that the global state elements of the app can be persisted locally.

Here is the example of our Root Reducer component that for now clusters two reducers functions : The authentication reducer function and the subscription to a shop reducer function.

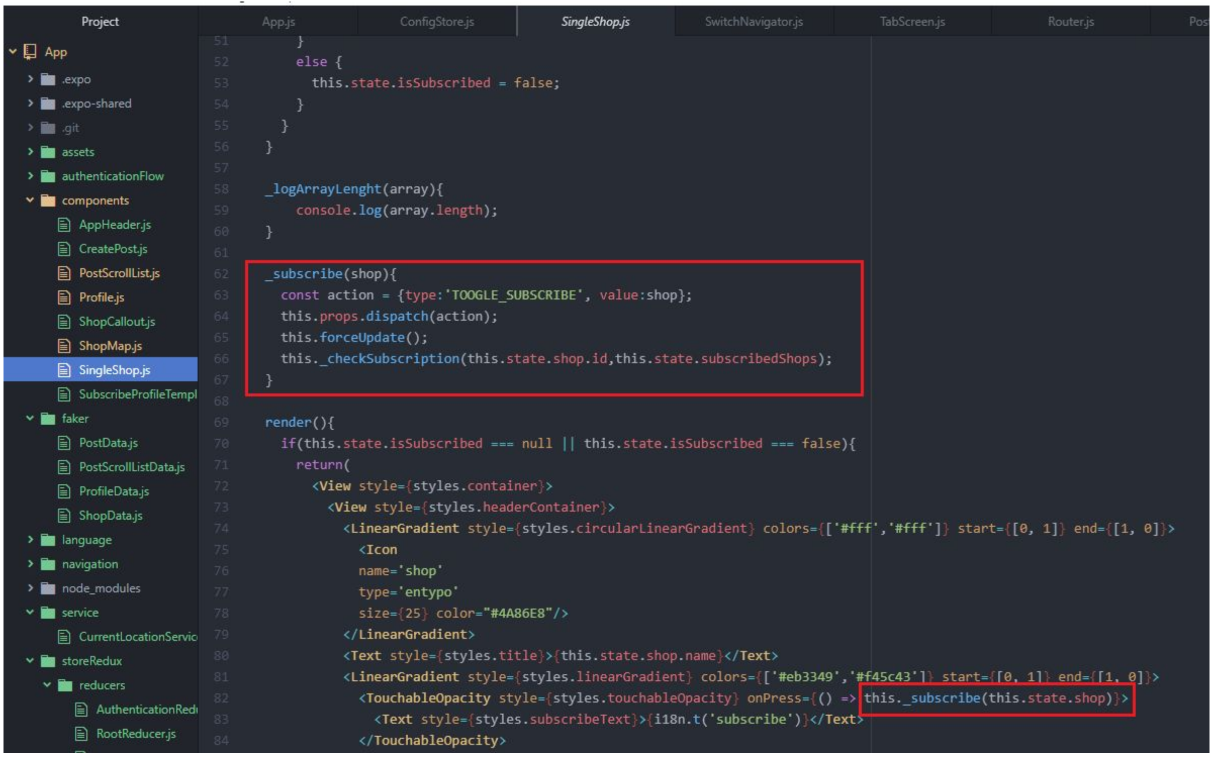


```
import {combineReducers} from 'redux';
import toggleAuthentication from './AuthenticationReducer';
import toggleSubscription from './ShopSubscriptionReducer';

const rootReducer =  combineReducers({
  toggleAuthentication,
  toggleSubscription
})

export default rootReducer;

```

Let’s take the example of the subscription to a shop reducer function.

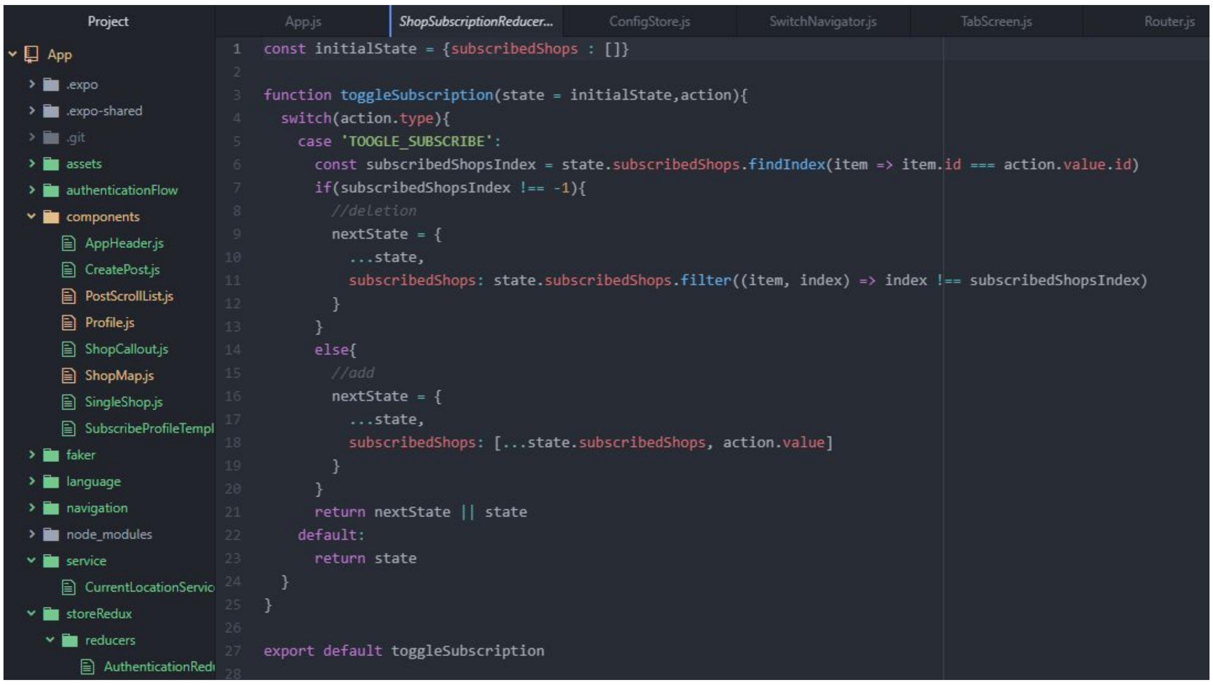
In the SingleShop component, the press on the subscribe button will trigger a _subscribe function. In this function, we create an action object. In the redux standars, an action should have two properties :

1. A type, whose aim is to triggered the related reducer
2. A value prop, which contains the data that need to be stored within the redux store.



```
      }
      else {
        this.state.isSubscribed = false;
      }
    }
  }

  _logArrayLenght(array){
      console.log(array.length);
  }

  _subscribe(shop){
    const action = {type:'TOOGLE_SUBSCRIBE', value:shop};
    this.props.dispatch(action);
    this.forceUpdate();
    this._checkSubscription(this.state.shop.id,this.state.subscribedShops);
  }

  render(){
    if(this.state.isSubscribed === null || this.state.isSubscribed === false){
      return(
        <View style={styles.container}>
          <View style={styles.headerContainer}>
            <LinearGradient style={styles.circularLinearGradient} colors={['#fff','#fff']} start={[0, 1]} end={[1, 0]}>
              <Icon
              name='shop'
              type='entypo'
              size={25} color="#4A86E8"/>
            </LinearGradient>
            <Text style={styles.title}>{this.state.shop.name}</Text>
            <LinearGradient style={styles.linearGradient} colors={['#eb3349','#f45c43']} start={[0, 1]} end={[1, 0]}>
              <TouchableOpacity style={styles.touchableOpacity} onPress={() => this._subscribe(this.state.shop)}>
                <Text style={styles.subscribeText}>{i18n.t('subscribe')}</Text>
              </TouchableOpacity>
```

The action is sent to the reducer thanks to the dispatch function.
Then the reducers function can operate logical operations on the action value.



```
const initialState = {subscribedShops : []}

function toggleSubscription(state = initialState,action){
  switch(action.type){
    case 'TOOGLE_SUBSCRIBE':
      const subscribedShopsIndex = state.subscribedShops.findIndex(item => item.id === action.value.id)
      if(subscribedShopsIndex !== -1){
        //deletion
        nextState = {
          ...state,
          subscribedShops: state.subscribedShops.filter((item, index) => index !== subscribedShopsIndex)
        }
      }
      else{
        //add
        nextState = {
          ...state,
          subscribedShops: [...state.subscribedShops, action.value]
        }
      }
      return nextState || state
    default:
      return state
  }
}

export default toggleSubscription
```

Economical part :

This part aims to summarize the main economical figures and documents we realised during this first sprint.

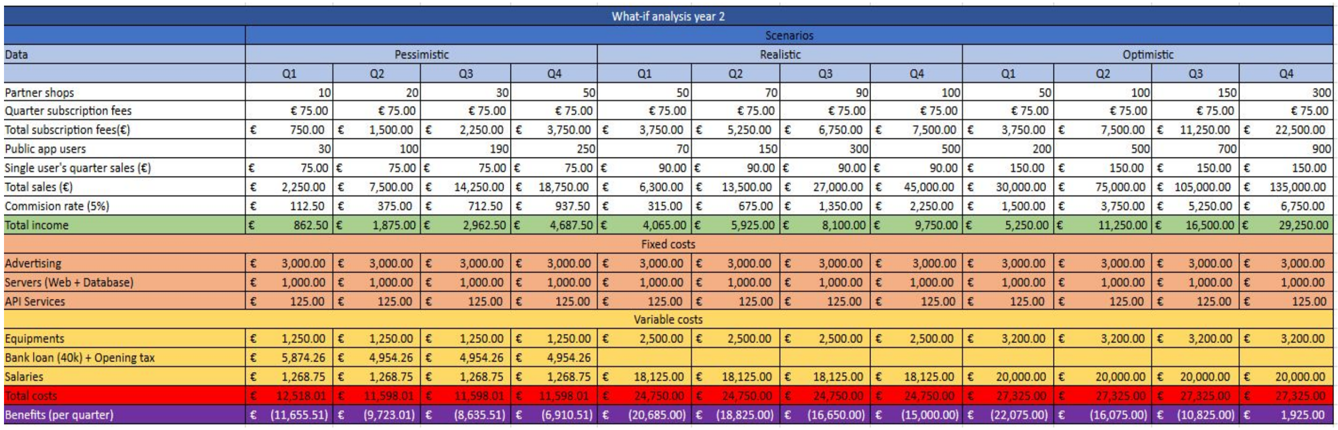
It presents :

1. The cashflow (4 years)
2. What if analysis from year two to year 4
3. Break even sales graph
4. NPV
5. ROI

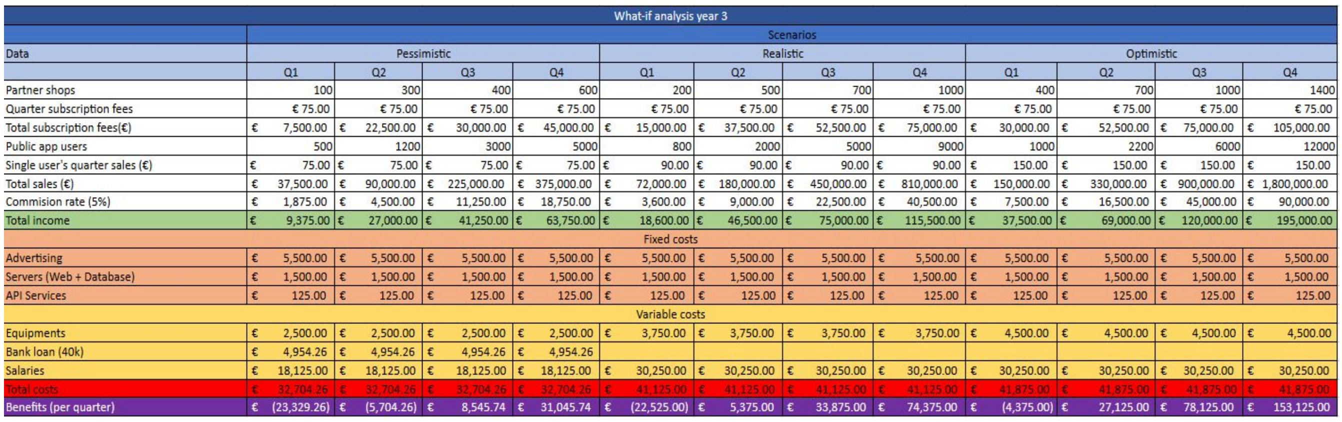
For further details, you could refer to the Financial factor document also provided in the deliver.

Project ProxiPrizes	Years															
	1				2				3				4			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
INNVIERTE program (Spain government help)	€ 40,000.00															
Num of Shops subscribed					50	70	90	100	200	500	700	1000	1500	2500	4000	4500
Shop Subscription (€75 per quarter)					€ 3,750.00	€ 5,250.00	€ 6,750.00	€ 7,500.00	€ 15,000.00	€ 37,500.00	€ 52,500.00	€ 75,000.00	€ 112,500.00	€ 187,500.00	€ 300,000.00	€ 337,500.00
Num of Users registered					70	150	300	500	800	2000	5000	9000	20000	28000	32000	40000
Single user's quarter sales (€90)					€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00	€ 90.00
Comission on Shop Sales (5%)					€ 315.00	€ 675.00	€ 1,350.00	€ 2,250.00	€ 3,600.00	€ 9,000.00	€ 22,500.00	€ 40,500.00	€ 90,000.00	€ 126,000.00	€ 144,000.00	€ 180,000.00
Total Income	€ 40,000.00	€ -	€ -	€ -	€ 4,065.00	€ 5,925.00	€ 8,100.00	€ 9,750.00	€ 18,600.00	€ 46,500.00	€ 75,000.00	€ 115,500.00	€ 202,500.00	€ 313,500.00	€ 444,000.00	€ 517,500.00
Open the company (S.L)	€ 3,600.00															
Advertising					€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 3,000.00	€ 5,000.00	€ 5,000.00	€ 6,000.00	€ 6,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00	€ 10,000.00
Scrum Master salary (€35,000/year)					€ 4,375.00	€ 4,375.00	€ 4,375.00	€ 4,375.00	€ 8,750.00	€ 8,750.00	€ 8,750.00	€ 8,750.00	€ 8,750.00	€ 8,750.00	€ 8,750.00	€ 8,750.00
Front-end developer salary (€26,000/year)					€ 3,250.00	€ 3,250.00	€ 3,250.00	€ 3,250.00	€ 6,500.00	€ 6,500.00	€ 6,500.00	€ 6,500.00	€ 6,500.00	€ 6,500.00	€ 6,500.00	€ 6,500.00
Back-end developer salary (€30,000/year)					€ 3,750.00	€ 3,750.00	€ 3,750.00	€ 3,750.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00
Full Stack developer salary (€30,000/year)					€ 3,750.00	€ 3,750.00	€ 3,750.00	€ 3,750.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00	€ 7,500.00
Sales Manager (€45,000/year)													€ 11,250.00	€ 11,250.00	€ 11,250.00	€ 11,250.00
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	€ 125.00	€ 125.00	€ 125.00	€ 125.00
Equipments	€ 5,000.00				€ 10,000.00				€ 15,000.00				€ 20,000.00			
Web Server	€ 2,000.00				€ 2,000.00				€ 3,000.00				€ 5,000.00			
Database Server	€ 2,000.00				€ 2,000.00				€ 3,000.00				€ 5,000.00			
Total Costs	€ 12,725.00	€ 125.00	€ 125.00	€ 125.00	€ 32,250.00	€ 18,250.00	€ 18,250.00	€ 18,250.00	€ 56,375.00	€ 35,375.00	€ 36,375.00	€ 36,375.00	€ 81,625.00	€ 51,625.00	€ 51,625.00	€ 51,625.00
Cashflow	€ 27,275.00	€ 27,150.00	€ 27,025.00	€ 26,900.00	€ (1,285.00)	€ (13,610.00)	€ (23,760.00)	€ (32,260.00)	€ (70,035.00)	€ (58,910.00)	€ (20,285.00)	€ 58,840.00	€ 179,715.00	€ 441,590.00	€ 833,965.00	€ 1,299,840.00

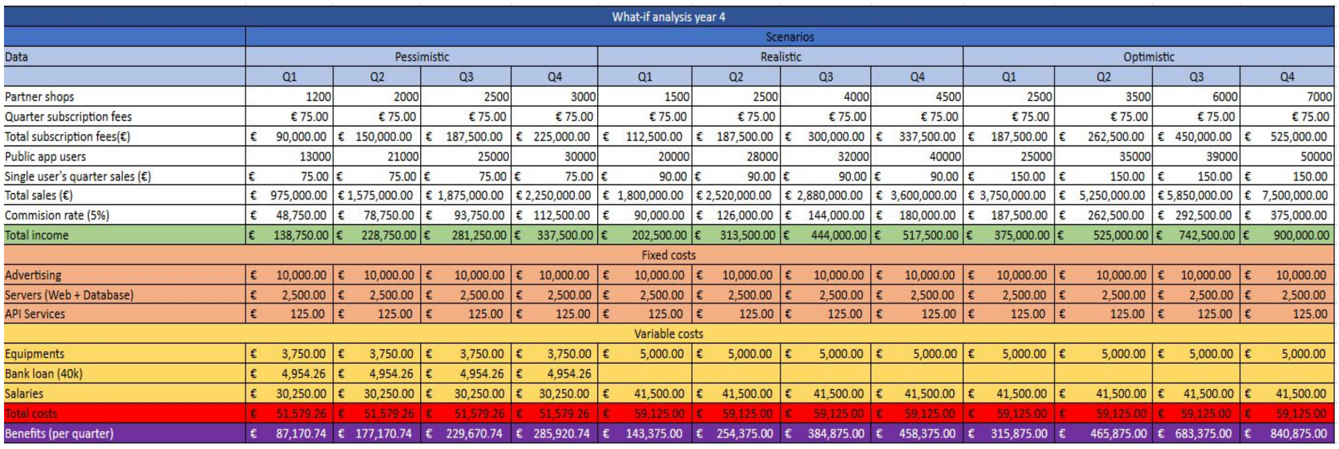
What if analysis :



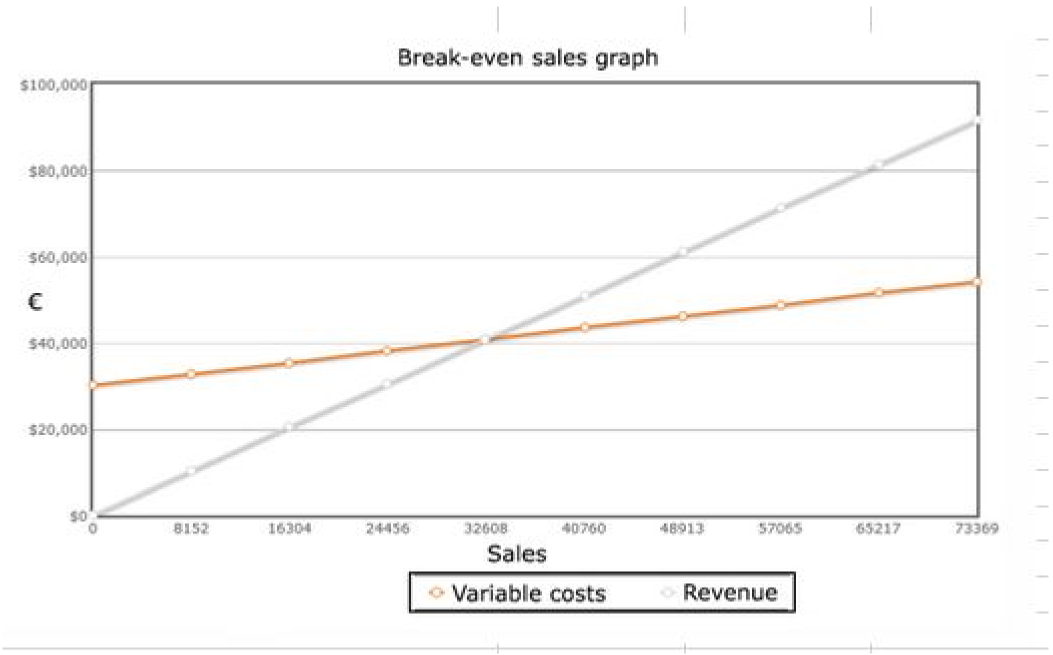
What-if analysis year 2												
	Scenarios											
Data	Pessimistic				Realistic				Optimistic			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Partner shops	10	20	30	50	50	70	90	100	50	100	150	300
Quarter subscription fees	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 75.00	€ 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



What-if analysis year 3												
	Scenarios											
Data	Pessimistic				Realistic				Optimistic			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Partner shops	100	300	400	600	200	500	700	1000	400	700	1000	1400
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(€)	€ 7,500.00	€ 22,500.00	€ 30,000.00	€ 45,000.00	€ 15,000.00	€ 37,500.00	€ 52,500.00	€ 75,000.00	€ 30,000.00	€ 52,500.00	€ 75,000.00	€ 105,000.00
Public app users	500	1200	3000	5000	800	2000	5000	9000	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 (€)	€ 37,500.00	€ 90,000.00	€ 225,000.00	€ 375,000.00	€ 72,000.00	€ 180,000.00	€ 450,000.00	€ 810,000.00	€ 150,000.00	€ 330,000.00	€ 900,000.00	€ 1,800,000.00
Commision rate (5%)	€ 1,875.00	€ 4,500.00	€ 11,250.00	€ 18,750.00	€ 3,600.00	€ 9,000.00	€ 22,500.00	€ 40,500.00	€ 7,500.00	€ 16,500.00	€ 45,000.00	€ 90,000.00
Total income	€ 9,375.00	€ 27,000.00	€ 41,250.00	€ 63,750.00	€ 18,600.00	€ 46,500.00	€ 75,000.00	€ 115,500.00	€ 37,500.00	€ 69,000.00	€ 120,000.00	€ 195,000.00
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	€ 18,125.00	€ 18,125.00	€ 18,125.00	€ 18,125.00	€ 30,250.00	€ 30,250.00	€ 30,250.00	€ 30,250.00	€ 30,250.00	€ 30,250.00	€ 30,250.00	€ 30,250.00
Total costs	€ 32,704.26	€ 32,704.26	€ 32,704.26	€ 32,704.26	€ 41,125.00	€ 41,125.00	€ 41,125.00	€ 41,125.00	€ 41,875.00	€ 41,875.00	€ 41,875.00	€ 41,875.00
Benefits (per quarter)	€ (23,329.26)	€ (5,704.26)	€ 8,545.74	€ 31,045.74	€ (22,525.00)	€ 5,375.00	€ 33,875.00	€ 74,375.00	€ (4,375.00)	€ 27,125.00	€ 78,125.00	€ 153,125.00



What-if analysis year 4												
	Scenarios											
Data	Pessimistic				Realistic				Optimistic			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
Partner shops	1200	2000	2500	3000	1500	2500	4000	4500	2500	3500	6000	7000
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(€)	€ 90,000.00	€ 150,000.00	€ 187,500.00	€ 225,000.00	€ 112,500.00	€ 187,500.00	€ 300,000.00	€ 337,500.00	€ 187,500.00	€ 262,500.00	€ 450,000.00	€ 525,000.00
Public app users	13000	21000	25000	30000	20000	28000	32000	40000	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 (€)	€ 975,000.00	€ 1,575,000.00	€ 1,875,000.00	€ 2,250,000.00	€ 1,800,000.00	€ 2,520,000.00	€ 2,880,000.00	€ 3,600,000.00	€ 3,750,000.00	€ 5,250,000.00	€ 5,850,000.00	€ 7,500,000.00
Commision rate (5%)	€ 48,750.00	€ 78,750.00	€ 93,750.00	€ 112,500.00	€ 90,000.00	€ 126,000.00	€ 144,000.00	€ 180,000.00	€ 187,500.00	€ 262,500.00	€ 292,500.00	€ 375,000.00
Total income	€ 138,750.00	€ 228,750.00	€ 281,250.00	€ 337,500.00	€ 202,500.00	€ 313,500.00	€ 444,000.00	€ 517,500.00	€ 375,000.00	€ 525,000.00	€ 742,500.00	€ 900,000.00
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	€ 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	€ 51,579.26	€ 51,579.26	€ 51,579.26	€ 51,579.26	€ 59,125.00	€ 59,125.00	€ 59,125.00	€ 59,125.00	€ 59,125.00	€ 59,125.00	€ 59,125.00	€ 59,125.00
Benefits (per quarter)	€ 87,170.74	€ 177,170.74	€ 229,670.74	€ 285,920.74	€ 143,375.00	€ 254,375.00	€ 384,875.00	€ 458,375.00	€ 315,875.00	€ 465,875.00	€ 683,375.00	€ 840,875.00



NPV	
Investment	€ (40,000.00)
Year 1 cashflow	€ 26,900.00
Year 2 cashflow	€ (32,260.00)
Year 3 cashflow	€ 58,840.00
Year 4 cashflow	€ 1,299,840.00
Interest %	1.05
Total	€ 1,393,320.00
NPV	€ 45,874.85

ROI	
Investment	€ (40,000.00)
Year 1 cashflow	€ 26,900.00
Year 2 cashflow	€ (32,260.00)
Year 3 cashflow	€ 58,840.00
Year 4 cashflow	€ 1,299,840.00
Revenues	€ 1,760,940.00
Expenses	€ 501,100.00
ROI	30.496

Internal Rate of Return	176.58%
Payback Period	2.611 years
	31.32 months