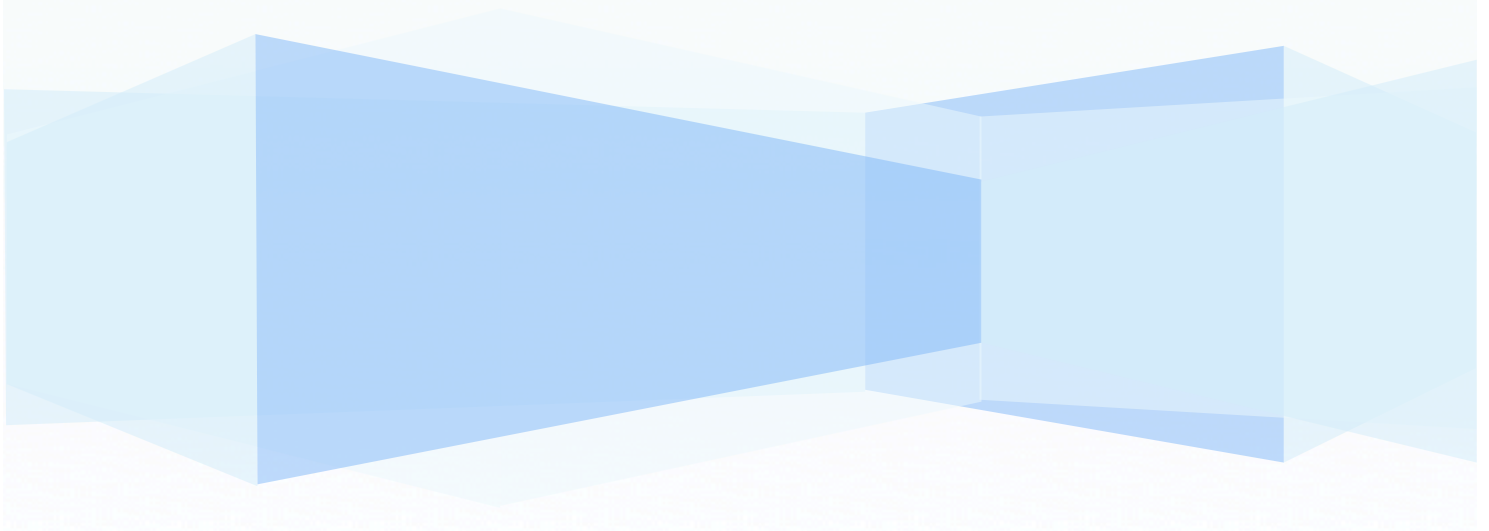


# DASHBOARD

Application de mesure de performance et  
d'aide à la prise de décision du restaurant  
« LA PECHE »

SHEY Louis CHIA

Master-1, Développement  
logiciel, mobile & IoT (IDEV)



## **REMERCIEMENTS**

Je remercie OUSMANE WADE, mon enseignant de gestion de projet qui m'a formé sur la gestion des projets, ce qui m'a permis d'obtenir par la suite une certification SCRUM et à bien mener ce projet jusqu'au bout. Je remercie également l'équipe pédagogique de Webitech pour la formation ainsi que ma famille pour m'avoir soutenu pendant cette période où tout n'a pas été facile. Enfin, j'exprime ma gratitude à tous ceux qui de près ou de loin m'ont soutenue dans ma formation.

# SOMMAIRE

REMERCIEMENTS	2
RESUME	5
INTRODUCTION	6
<b>PARTIE I: PRESENTATION DU PROJET</b>	<b>7</b>
<b>I – PRESENTATION DE “LA PECHE”</b>	<b>8</b>
<b>II – PRESENTATION DASHBOARD</b>	<b>9</b>
1. Dashboard	9
I. Visions et mission	9
<b>2. Ecosystème de dashboard</b>	<b>10</b>
I. Les partenaires de dashboard	10
II. Partenaires prospect de DASHBOARD	11
<b>PARTIE II : LA PROBLEMATIQUE DEGAGÉ</b>	<b>12</b>
<b>PARTIE III : SOLUTIONS PROPOSÉS</b>	<b>14</b>
<b>I - LA MÉTHODOLOGIE UTILISÉE</b>	<b>15</b>
1. Langages et outils utilises	15
2. Justification des outils utilisés	19
<b>II – IMPLEMENTATION DE “LA PECHE”</b>	<b>21</b>
1. Modelisation du systeme	21
2. Les acteurs du système	21
I. Le client	21
II. Administrateur	21
3. Les cas d’utilisation « Use Cases »	22
4. Diagrammes de classes	24

<b>5. Vue générale du système d'information</b>	<b>24</b>
<b>6. Implementations des Web services</b>	<b>27</b>
I. web service pour ressortir la catégorie de produit la plus commandée:	28
II. web service pour demontrer les proportions d'achat des produits par tranches d'ages:	29
<b>III – IMPLEMENTATION DE DASHBOARD</b>	<b>31</b>
<b>1. Creation du projet</b>	<b>31</b>
<b>2. Mise en place des composantes</b>	<b>32</b>
<b>IV – DEPLOIEMENT</b>	<b>35</b>
<b>1. Deploiement de Dashboard</b>	<b>35</b>
<b>2. Deploiement de “LA PECHE”</b>	<b>36</b>
<b>CONCLUSION</b>	<b>38</b>
<b>1. Leçons apprises</b>	<b>38</b>
<b>1. Les ameliorations du futures</b>	<b>38</b>
II. Testes Unitaires :	41
III. Amélioration de la performance	42
IV. Rendre le projets plus flexible	42
<b>WEBOGRAPHIE</b>	<b>43</b>
<b>GLOSSAIRE</b>	<b>44</b>

## **RESUME**

Les entreprises deviennent de plus en plus numériques et tout aussi compétitives. Afin de prospérer dans un tel scénario, les dirigeants utilisent des données informatiques, des logiciels, de l'intelligence artificielle pour aider à la prise de décision. Les entreprises veulent s'assurer que chaque somme d'argent investi n'est pas gaspillé et que les besoins des clients ciblés sont satisfaits. Ils veulent également anticiper les besoins futurs et être très informés des habitudes de consommation afin de mieux servir. Dashboard, se présente comme un outil qui manipule les données et le présente de manière visuellement attrayante, de sorte qu'une entreprise ne perdent pas de vue les indicateurs clés et prennent donc des meilleures décisions.

# INTRODUCTION

Le Rapport d'activité pour les étudiants de 4ème année de Webitech fait partie intégrante de la formation en ***Développement logiciel, mobile & IoT (IDEV)*** tout en constituant le virage essentiel de leur intégration dans le monde du travail.

C'est ainsi que j'ai travaillé sur mon projet personnel de mise en place d'une application pour résoudre un problème que j'ai remarqué dans un restaurant fictif, ce qui m'a permis de mettre en pratique les notions apprises pendant mon année académique, de faire valoir mes connaissances et monter en compétences de la bonne manière.

## **PARTIE I: PRESENTATION DU PROJET**

## I – PRESENTATION DE “LA PECHE”

Le restaurant “La Pêche” est un restaurant avec les caractéristiques suivantes :

- Il comporte 200 clients qui ont des comptes dans le site du restaurant.

```
3.0.1 :016 > Client.count #=> 200
(0.9ms) SELECT COUNT(*) FROM "clients"
```

- Il est présent dans quatre(04) villes pilote pour le moment : Paris, Marseille, Lyon, Bordeaux et Nice.

```
3.0.1 :020 > Client.select(:city).group(:city)
Client Load (50.9ms) SELECT "clients"."city" FROM "clients" GROUP BY "clients"."city"
```

id	city
	Bordeaux
	Lyon
	Marseille
	Nice
	Paris

- Il vent quatre (04) categories de produits : les Pizza, les boissons, les gateaux et les glaces.

```
3.0.1 :021 > Category.all
Category Load (14.9ms) SELECT "categories".* FROM "categories"
```

id	created_at	updated_at	cat_name
1	2021-04-14 16:22:53 UTC	2021-04-14 16:22:53 UTC	pizza
2	2021-04-14 16:22:53 UTC	2021-04-14 16:22:53 UTC	drinks
3	2021-04-14 16:22:53 UTC	2021-04-14 16:22:53 UTC	cake
4	2021-04-14 16:22:53 UTC	2021-04-14 16:22:53 UTC	glaces

- Il existe cinq (05) produits différentes dans chaque catégorie. Il y a donc 20 produits au total propose par “LA PECHE” telque indiqué ci-dessous :



Item Load (16.4ms) SELECT "items".\* FROM "items"

id	name	category_id	price
1	pizza-Sushi	1	4
2	pizza-California Maki	1	1
3	pizza-Tuna Sashimi	1	7
4	pizza-Ricotta Stuffed Ravioli	1	2
5	pizza-Pho	1	7
6	jus-Juniper Berries	2	4
7	jus-Cranberry	2	4
8	jus-Passionfruit	2	6
9	jus-Dried Apricots	2	1
10	jus-Lychees	2	1
11	cake-Avocado	3	7
12	cake-Dates	3	7
13	cake-Tomatoes	3	6
14	cake-Snowpeas	3	4
15	cake-Mulberries	3	2
16	glace-Mangosteens	4	8
17	glace-Cranberry	4	6
18	glace-Olives	4	8
19	glace-Prunes	4	2
20	glace-Butternut pumpkin	4	5

## II – PRESENTATION DASHBOARD

### 1. Dashboard

Dashboard est une application qui complemente l’application principale du restaurant “LA PECHE” en produisant des rapports ou statistique sur les commandes/achats, les clients, les produits et categorie de produits sur les cinq (05) villes pilotes ou se trouve le restaurant.

#### I. Visions et mission

La vision de DASHBOARD est d’être un outil qui aide des entreprises à avoir plus de visibilité sur leur performance.

La mission de DASHBOARD concernant le restaurant “LA PECHE” est de tracker sa performance, voir en temps reels ses données et

permet une meilleure appréciation sur le fonctionnement du restaurant et aide à la prise de décision stratégique telle que :

- Le client qui a fait le plus gros achat (on peut prévoir un cadeau pour celui-ci);
- Les produits les plus achetés (une classification des 05 meilleurs produits);
- Le nombre total de commandes;
- Le pourcentage de dépenses entre les hommes et les femmes;
- Le pourcentage de dépenses entre les différentes tranches d'âges (les adolescents, les jeunes et les adultes);
- Le pourcentage des commandes classées par villes;
- La liste totale des clients;
- Statistiques sur les habitudes de consommation par villes.

## **2. Ecosystème de dashboard**

### **1. Les partenaires de dashboard**

Le partenaire principal de DASHBOARD est le restaurant "LA PECHE" où l'expérimentation a été faite. Il faudra quand même relever que DASHBOARD compte s'étendre à toutes les entreprises partenaires de "LA PECHE". Il faut noter que si l'on peut traquer les entreprises partenaires de "LA PECHE" (notamment ceux qui approvisionnent en matières premières pour la restauration) avec leur accord s'il n'ont pas déjà un système mis en place, on pourra anticiper du côté de "LA PECHE" comment combler les lacunes d'un stock et anticiper sur sa commande chez une autre entreprise.

On pourrai également après observation prévoir s'il est nécessaire de commencer la production d'une matière première nécessaire pour le bon fonctionnement du restaurant.

## **II. Partenaires prospect de DASHBOARD**

DASHBOARD souhaite étendre ses services aux maximum de restaurants. Mais il souhaite proposer ses services après experimentation et observation avec son partenaire principale "LA PECHE". Après cette première étape de perfectionnement de l'outil taillé pour un restaurant et ensuite pour d'autres restaurants, le but est de créer un outil de monitoring pour aider n'importe quel entreprise pour améliorer sa gestion. Ainsi, au moment opportun, le pus de partenaires possibles sera la bienvenue.

## **PARTIE II : LA PROBLEMATIQUE DEGAGÉ**

Après analyse du restaurant “LA PECHE”, il se dégage que le problème principal est celui de pouvoir avoir une visibilité sur comment le restaurant fonctionne, un outil qui démontre la performance, un outil qui va aider le restaurant à anticiper sur le futur du restaurant et augmenter son chiffre d'affaire.

## **PARTIE III : SOLUTIONS PROPOSÉS**

# I - LA MÉTHODOLOGIE UTILISÉE

## 1. Langages et outils utilises

- **HTML** : HTML permet à l'utilisateur de créer et de structurer des sections, des paragraphes, des en-têtes, des liens et des citations pour les pages Web et les applications. HTML n'est pas un langage de programmation, ce qui signifie qu'il n'a pas la capacité de créer des fonctionnalités dynamiques;
- **CSS** : CSS est une technologie de base du World Wide Web, aux côtés de HTML et JavaScript. CSS est conçu pour permettre la séparation de la présentation et du contenu, y compris la mise en page, les couleurs et les polices;
- **Javascript** : JavaScript est un langage de script qui vous permet de créer du contenu mis à jour dynamiquement, de contrôler le multimédia, d'animer des images et plein d'autres choses;
- **ReactJs** : React est une bibliothèque JavaScript créée pour créer des interfaces utilisateur rapides et interactives pour les applications Web et mobiles. Il s'agit d'une bibliothèque frontale open source, basée sur des composants, responsable uniquement de la couche de vue de l'application. Dans l'architecture MVC (Model View Controller), la couche de vue est responsable de l'apparence et de la convivialité de l'application ;
- **Ruby** : Ruby est un langage de programmation interprété, de haut niveau et polyvalent. Il a été conçu et développé au milieu des années 1990 par Yukihiro "Matz" Matsumoto au Japon. Ruby est typé dynamiquement et utilise le ramasse-miettes. Il prend en charge

plusieurs paradigmes de programmation, y compris la programmation procédurale, orientée objet et fonctionnelle ;

- **Ruby on Rails** : Rails est une structure MVC (Model – View – Controller), fournissant des structures par défaut pour une base de données, un web service et des pages Web. Il encourage et facilite l'utilisation de standards Web tels que JSON ou XML pour le transfert de données et HTML, CSS et JavaScript pour l'interface utilisateur. En plus de MVC, Rails met l'accent sur l'utilisation d'autres modèles et paradigmes d'ingénierie logicielle bien connus, y compris la convention sur la configuration (CoC), "Don't Repeat Yourself" (DRY) et le "Active Record" ;
- **Faker** : Faker est un GEM Ruby permettant la production des données test. Elle a été utilisée pour la production des clients tests, les menu tests, les achats effectués dans notre restaurant fictive;
- **database\_cleaner** : C'est un GEM Ruby utilisé pour vider la base de données tests afin de le re-initialiser pour pouvoir recommencer les tests;
- **Irbtools** : est un GEM Ruby permettant une affichage des données sous forme de tableau (plus lisible) quand on faire des requête sur la console du serveur web;
- **SQLite3** : SQLite3 est une base de données gratuite et compacte que vous pouvez utiliser facilement pour créer et utiliser une base de données. Bien que SQLite3 ne soit pas une base de données complète, il prend en charge un ensemble étonnamment large de normes SQL et est idéal pour ceux qui commencent tout juste à apprendre SQL ainsi que pour les développeurs qui ont besoin d'un



moteur de base de données simple pour se connecter à leurs applications;

- **rack-cors** : est un GEM Ruby permettant de résoudre les problèmes CORS. CORS est un protocole qui permet aux scripts exécutés sur un navigateur client d'interagir avec des ressources d'une origine différente. Ceci est utile car, grâce à la politique de même origine suivie par *XMLHttpRequest* et *fetch*, JavaScript ne peut appeler que des URL qui vivent sur la même origine que l'emplacement où le script est exécuté. Par exemple, si une application JavaScript souhaite effectuer un appel AJAX vers une API s'exécutant sur un domaine différent, elle en serait empêchée grâce à la politique de même origine;
- **SASS** : Sass est un pré-processeur CSS. Sass est totalement compatible avec toutes les versions de CSS Sass réduit la répétition de CSS et fait donc gagner du temps Sass a été conçu par Hampton Catlin et développé par Natalie Weizenbaum en 2006;
- **VSCode** : Visual Studio Code est un éditeur de code source gratuit créé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, de la coloration syntaxique, de l'achèvement de code intelligent, des extraits de code, de la refactorisation du code et de Git intégré. Les utilisateurs peuvent modifier le thème, les raccourcis clavier, les préférences et installer des extensions qui ajoutent des fonctionnalités supplémentaires;
- **GIT/Github**: GitHub, Inc. est un fournisseur d'hébergement Internet pour le développement de logiciels et le contrôle de version à l'aide

- de Git. Il offre la fonctionnalité de contrôle de version distribué et de gestion du code source de Git, ainsi que ses propres fonctionnalités;
- **JSON** : JSON est un format de fichier standard ouvert et un format d'échange de données qui utilise un texte lisible par l'homme pour stocker et transmettre des objets de données constitués de paires attribut-valeur et de tableaux (ou d'autres valeurs) ;
  - **Heroku** : Heroku est une plate-forme cloud qui permet aux entreprises de créer, de fournir, de surveiller et de mettre à l'échelle des applications c'est l'un des moyens les plus rapide de passer de l'idée à l'URL, en contournant les maux de tête liés à l'infrastructure. Les applications sont transformatrices. Les applications sont la manière dont les clients interagissent désormais avec les entreprises. Les développeurs sont essentiels au succès des applications ;
  - **Netlify** : Netlify est une société de cloud computing basée à San Francisco qui propose des services d'hébergement et de backend sans serveur pour les applications Web et les sites Web statiques. Ses fonctionnalités incluent le déploiement continu de Git sur Netlify Edge, l'infrastructure réseau mondiale de distribution d'applications de l'entreprise, la gestion des formulaires sans serveur, la prise en charge des fonctions AWS Lambda et plus encore. L'application DASHBOARD est déployé sous Netlify ;
  - **Hiechart** : Une bibliothèque JavaScript robuste qui permet aux développeurs de créer facilement des graphiques interactifs pour allouer, coordonner et afficher des tâches, des événements et des ressources le long d'une chronologie.;

## 2. Justification des outils utilisés

Le choix des outils (langages, framework, techniques ) a été fait en tenant compte des avancés des technologies web actuel.

### – **Ruby/Ruby on Rails :**

Ce framework a été choisit parce qu’il permet de garantir :

- Un gain de productivité,
- Un travail en groupe entre développeurs plus facile,
- Une clarté dans l’organisation des sources du projet grâce à l’architecture « MVC »,
- Une maintenance évolutive et plus aisée ,
- Une réutilisation des composants sans les réinventer complètement,
- Une robustesse du code.

Mais qu’est-ce qu’un Framework ? Un Framework est un ensemble de composants qui servent à créer les fondations, l'architecture et les grandes lignes d'un système (logiciel, site web etc.). Il en existe des centaines pour différentes langages de programmation. Mais en revanche, cet outil demande aussi une courbe d’apprentissage plus élevée par rapport au langage natif dont il se sert pour faciliter les tâches du développeur. Ruby est utilisé pour contruire le backend pour l’application du restaurant pilote (“LA PECHE”). Elle a été utilisée pour mettre sur pied les web services consommés par DASHBOARD.

- **Javascript/ReactJs :** JavaScript est un langage de programmation utilisé pour rendre les pages Web interactives. C'est un langage qui

donne vie à une page: les éléments interactifs et les animations. React (une librairie javascript) facilite la création d'interface utilisateur interactives, création des vues simples pour chaque état de votre application, et React met à jour et restitue efficacement les composants lorsque les données changent. Elle est utilisée pour mettre sur pied DASHBOARD pour la consommation des web services mis sur pied sur le backend de "LA PECHE"

- **Hiechart** : est un tres bon outil avec des bon proposition de graphiques pour presenter des statistiques. Etant un package npm, cela m'a permit de bien l'integrer dans ReactJs.

## II – IMPLEMENTATION DE “LA PECHE”

### 1. Modelisation du systeme

Dans cette partie, nous évoquerons la modélisation du système «LA PECHE» qui va nous permettre de ressortir le système d’information associé. Pour se faire, le langage de modélisation UML sera employé, avec SQLite comme SGBD (gestion de la base de données).

### 2. Les acteurs du système

#### I. Le client

Sera considéré comme client, tout internaute qui a un compte et peut se connecter sur le site « LA PECHE » avec un login et un mot de passe, il pourra en outre :

- modifier les parametres de son compte;
- Se créer un compte « candidat » ou « recruteur »;
- Faire un achat dans le site a base des produits proposé.

#### II. Administrateur

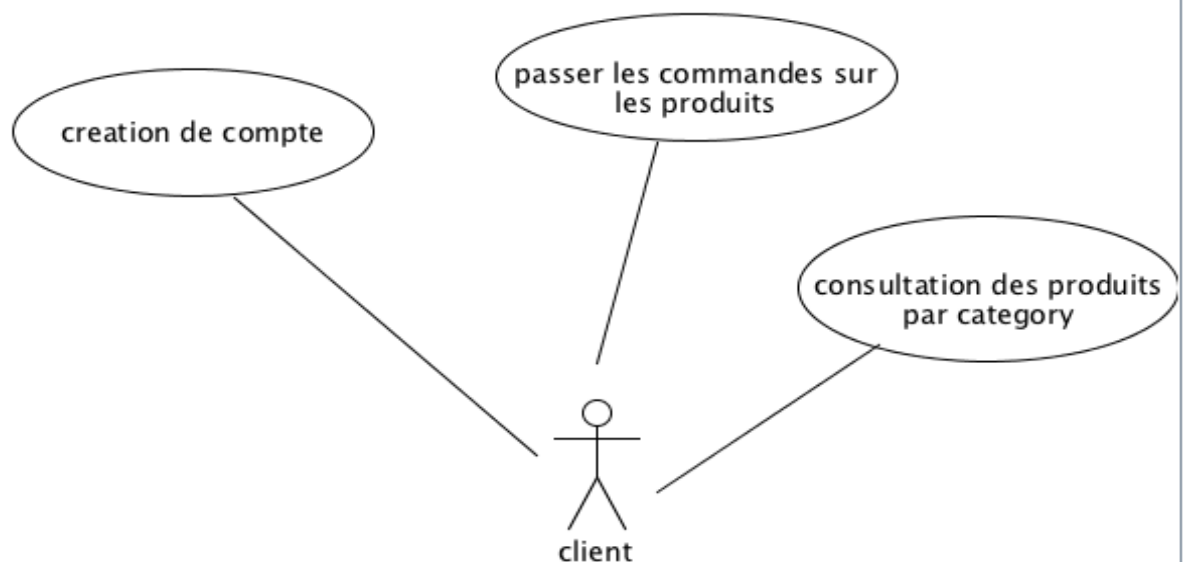
Il constitue ici la face cachée du système pour tout internaute. Son rôle est de gérer l’ensemble les différentes fonctionnalités du site, de surveiller leurs activités, de mettre à jour le site. Il pourra donc :

- ajouter/supprimer/modifie la catégorie des produits dans le restaurant ;
- ajouter/supprimer/modifier les produits proposés par le restaurant ;
- Gérer les comptes des clients ;

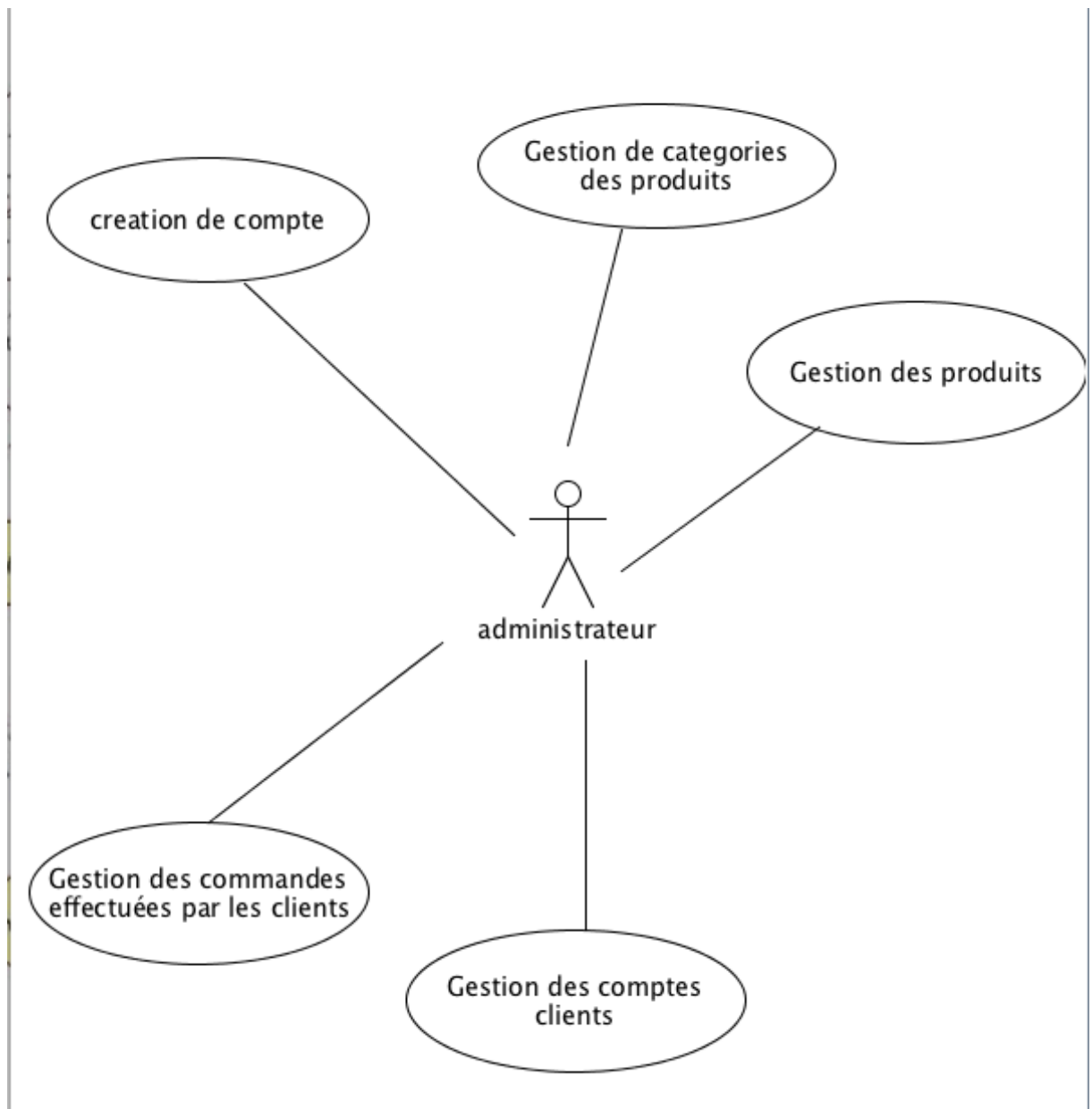
- consulter et voir l'ensemble des achats effectués dans le restaurant ;
- Consulter les différents états d'activités sur le site.

### 3. Les cas d'utilisation « Use Cases »

Il s'agit ici tout simplement de schématiser les différentes actions que pourront effectuer les acteurs du système mentionnés précédemment.



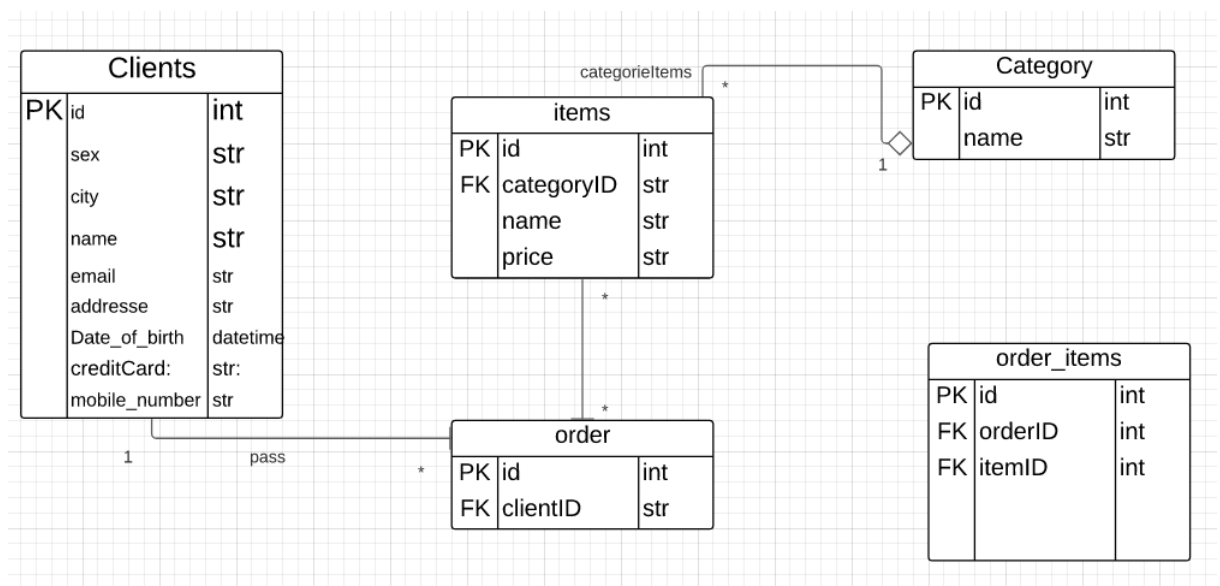
**Figure 1 : Diagramme cas d'utilisation : le client**



**Figure 1 : Diagramme cas d'utilisation : l'administrateur**

## 4. Diagrammes de classes

En génie logiciel, un diagramme de classes dans le langage de modélisation unifié (UML) est un type de diagramme de structure statique qui décrit la structure d'un système en montrant les classes du système, leurs attributs, opérations (ou méthodes) et les relations entre les objets.



*int* – integer/entier

*str* – string/chaîne de caractere

*datetime* – format de date

*PK* – Primary Key/Clé primaire

*FK* – Foreign key/ Clé étrangère

## 5. Vue générale du système d'information

Le système d'information de « LA PECHE » a été implémenté avec SQLite comme SGBD. Il s'agit ici, à ce niveau de réalisation du projet, de passer à la structuration des informations concernant l'application web dans une base de données. Grâce aux règles régissant le passage d'un modèle conceptuel de données matérialisé ici par nos diagrammes de classes, à un modèle logique de données donc à la



base de données, nous avons généré le système d'information.

Sous Ruby on rails, nous avons le fichier «*db/schema.rb*» qui est reproduit à chaque fois que vous exécutez une migration. Il reproduit l'intégralité du schéma de votre base de données. Il est également utilisé par certaines commandes de migration dans Rails pour éviter de refaire toutes les migrations une à une (dans le cas d'une réinitialisation de migration par exemple).

Les migrations vous permettent de modifier le schéma de votre base de données dans votre application Rails. Vous utilisez donc du code Ruby au lieu de SQL. L'utilisation de migrations Rails au lieu de SQL présente plusieurs avantages. Les applications Rails qui peuvent rester dans le modèle Active Record sont indépendantes de la base de données. Si vous devez écrire du SQL pour modifier votre schéma, vous perdez cette indépendance. Les migrations évitent cela puisque vous apportez les modifications dans Ruby indépendant de la plateforme. Les migrations conservent les modifications du schéma de votre base de données avec le code de votre application. Ils sont écrits en Ruby et versionnés avec le reste de votre application.

C'est aussi le seul endroit où vous pouvez voir toutes vos tables à la fois pour la référence des colonnes.

```

ActiveRecord::Schema.define(version: 2021_04_14_161229) do

  create_table "categories", force: :cascade do |t|
    t.datetime "created_at", precision: 6, null: false
    t.datetime "updated_at", precision: 6, null: false
    t.string "cat_name"
  end

  create_table "clients", force: :cascade do |t|
    t.string "sex"
    t.string "city"
    t.string "name"
    t.string "email"
    t.string "address"
    t.datetime "dateofbirth"
    t.string "creditCard"
    t.datetime "created_at", precision: 6, null: false
    t.datetime "updated_at", precision: 6, null: false
    t.string "mobile"
  end

  create_table "items", force: :cascade do |t|
    t.string "name"
    t.integer "category_id", null: false
    t.integer "price"
    t.datetime "created_at", precision: 6, null: false
    t.datetime "updated_at", precision: 6, null: false
    t.index ["category_id"], name: "index_items_on_category_id"
  end
end

```

**Figure 1: Schema de la BD, fichier "db/schema.rb"**

```

create_table "order_items", force: :cascade do |t|
  t.integer "order_id", null: false
  t.datetime "created_at", precision: 6, null: false
  t.datetime "updated_at", precision: 6, null: false
  t.integer "client_id", null: false
  t.integer "item_id", null: false
  t.string "sex"
  t.integer "price"
  t.string "city"
  t.integer "age"
  t.string "age_group"
  t.index ["client_id"], name: "index_order_items_on_client_id"
  t.index ["item_id"], name: "index_order_items_on_item_id"
  t.index ["order_id"], name: "index_order_items_on_order_id"
end

create_table "orders", force: :cascade do |t|
  t.datetime "created_at", precision: 6, null: false
  t.datetime "updated_at", precision: 6, null: false
  t.integer "client_id", null: false
  t.index ["client_id"], name: "index_orders_on_client_id"
end

add_foreign_key "items", "categories"
add_foreign_key "order_items", "clients"
add_foreign_key "order_items", "items"
add_foreign_key "order_items", "orders"
add_foreign_key "orders", "clients"
end

```

**Figure 2: Schema de la BD, fichier "db/schema.rb" (suite)**

afin de créer les données de base de données appropriées à partir de ce fichier `schema.rb` dans Ruby on rails, la commande suivante doit être exécutée sur le terminal: `rails db:migrate`

## 6. Implementations des Web services

Après avoir construit les bases de données du restaurant «LA PECHE», il a fallu implémenter des web services (API) au niveau des contrôleurs et exposer ces données afin qu'ils puissent être consommés par une application front-end. Ces web services

consistes principalement à interroger la base de données afin d'obtenir diverses statistiques concernant les informations que le restaurateur souhaite tracker et les rendre disponibles (avec Ruby on rails) sous la forme de JSON, qui est le format de données le plus utilisé par les API.

Quelques exemples de web services

### I. web service pour ressortir la catégorie de produit la plus commandée:

```
11
12 # most ordered Category
13 def most_ordered_category
14   itemsHash = { results: []}
15   orders = OrderItem.most_ordered_5_items
16   category = Category.find(Item.where('id' => orders.first.item_id)[0].category_id)
17   if category
18     itemsHash[:results] << {category: category.cat_name}
19     render json: itemsHash
20   else
21     render json: { message: 'Category not found' }
22   end
23 end
24
```

accessible via l'URL

`http://127.0.0.1:3000/categories/most\_ordered\_category`.

Elle produit le resultat JSON ci-dessous:

```
// http://127.0.0.1:3000/categories/most_ordered_category

{
  "results": [
    {
      "category": "drinks"
    }
  ]
}
```

**Figure 3 : Résultat JSON => http://localhost:3000/categories/most\_ordered\_category**

## II. web service pour demontrer les proportions d'achat des produits par tranches d'ages:

```
# most spending age group (sum oney spent by each group)
# 0-26 [Adolescence]
# 27-35 [Youths]
# 35- [adults]
def spending_amounts_by_age_group
  itemsHash = { results: []}
  orders = OrderItem.spending_by_age_group
  total_orders = OrderItem.count
  orders.keys.each do |age_group|
    percentage = orders[age_group]/@total_spent * 100
    itemsHash[:results] << {age_group: age_group, amount: orders[age_group], total_orders: total_order
    |percentage: helper.number_to_percentage(percentage, precision: 2)}
  end
  render json: itemsHash
end
```

- 0 – 26 ans : represente les adolescents
- 27 – 35 ans represente les jeunes
- 35 ans et plus, represente les adultes

accessible via l'url

`http://127.0.0.1:3000/categories/most\_ordered\_category`, elle produit le resultat JSON ci-dessous:

```
// http://127.0.0.1:3000/orderitems/orders_by_sex

{
  "results": [
    {
      "sex": "Female",
      "numberOrders": 1781,
      "percentage": "50.89%",
      "totalOrders": 3500
    },
    {
      "sex": "Male",
      "numberOrders": 1719,
      "percentage": "49.11%",
      "totalOrders": 3500
    }
  ]
}
```

Figure 4: Résultat JSON => http://localhost:3000/orderitems/order\_by\_sex

Pour voir la liste complet des web services disponibles, il suffit de regarder l'ensemble des URL mise a` disposition depuis le fichier *'app/config/routes.rb'* don't le contenu est ci-dessous:

```
1  Rails.application.routes.draw do
2    get 'categories/index'
3    # namespace :api do
4    #   resources :items, only: [:index]
5    get '/clients/total_clients', to: 'clients#total_clients'
6    get '/orders/total_orders', to: 'orders#total_orders'
7
8    get '/items', to: 'items#index'
9    get '/items/most_ordered_items', to: 'items#most_ordered_items'
10
11   get '/categories', to: 'categories#index'
12   get '/categories/most_ordered_category', to: 'categories#most_ordered_category'
13
14   get '/orderitems', to: 'order_items#index'
15   get '/orderitems/orders_by_sex', to: 'order_items#orders_by_sex'
16   get '/orderitems/orders_by_city', to: 'order_items#orders_by_city'
17   get '/orderitems/highest_spender', to: 'order_items#highest_spender'
18   get '/orderitems/spending_amounts_by_age_group', to: 'order_items#spending_amounts_by_age_group'
19   get '/orderitems/volume_orders_by_age_group', to: 'order_items#volume_orders_by_age_group'
20
21   # end
22
23 end
```

Il est aussi possible de verifier l'ensemble des chemins d'accès via le terminal avec la commande : *'rails routes'*

### III – IMPLEMENTATION DE DASHBOARD

Dashboard est une application Front-end qui va permettre la consommation des web services développés dans “LA PECHE”. La technologie utilisée ici est REACTJs.

#### 1. Creation du projet

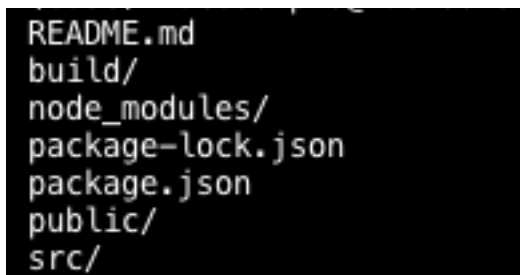
Pour créer un nouveau projet React, il faut utiliser les commandes ci-dessous dans un terminal pour créer le projet, se déplacer dans le répertoire du projet et ensuite lancer le projet:

```
npx create-react-app dashboard
```

```
cd dashboard
```

```
npm start
```

Les sous-dossiers et fichiers suivants sont présents dans le projet de dashboard :



```
README.md
build/
node_modules/
package-lock.json
package.json
public/
src/
```

**src/** : est le répertoire dans lequel les composants se trouvent. Les composants React sont des bouts de code indépendants et réutilisables. Elles ont le même objectif que les fonctions JavaScript, mais fonctionnent de manière isolée et renvoient du HTML via une fonction `render()`. Les composants sont de deux types, les composants de classe et les composants de fonction. La majorité des codes sources du projet se trouve dans ce répertoire;

**Package.json:** ce fichier est l'un des principaux composants de l'environnement d'exécution NodeJs. Il est contenu dans tous les packages npm. Il s'agit d'un fichier JSON qui contient les informations de base sur le projet. Ce fichier est utilisé pour contenir les différentes métadonnées et informations sur le projet;

**Readme:** c'est un fichier texte qui est souvent inclus avec un logiciel et contient des informations générales ou des instructions sur le logiciel;

**node\_modules/:** ce dossier contient les bibliothèques téléchargées depuis npm. Il ne doit pas être téléchargé sur github, car tous ceux qui clonent le depot/projet peuvent le télécharger eux-mêmes;

**build/:** La commande `npm run build` crée un répertoire *build/* avec une version de production de l'application. Dans le répertoire *build/*, se trouveront les fichiers JavaScript et CSS statiques. Chaque nom de fichier à l'intérieur de *build/* contiendra un hachage unique du contenu du fichier. Ce hachage dans le nom du fichier permet des techniques de mise en cache à long terme.

## 2. Mise en place des composantes

plusieurs composantes ont été mise en place pour consommer ces différentes web services. Tous les composantes de DASHBOARD se trouvent dans le repertoire *src/components/*. La consommation effective a été réalisé avec la fonction `fetch()` de javascript. Un exemple de composante qui consomme le web service qui detail les produits les plus acheter est présenté ci-dessous:



```

1  import React, { Component } from 'react'
2  import Highcharts from 'highcharts';
3  import HighchartsReact from 'highcharts-react-official';
4  import baseUrl from '../configBaseUrl'
5
6
7  export class MostOrderedItems extends Component {
8      constructor(props) {
9          super(props)
10
11          this.state = {
12              itemsArr: 0
13          }
14      }
15
16
17      componentDidMount() {
18          const path = '/items/most_ordered_items'
19          const fetchURL = baseUrl + path
20          fetch(fetchURL)
21              .then(Response => Response.json())
22              .then(apiData => {
23                  this.setState({
24                      itemsArr: apiData.results
25                  })
26              })
27              .catch(e => {
28                  console.log(e);
29                  return e;
30              });
31      }
32
33      getItemArrAndPercentageArr(originalArr) {
34          const isApiDataReady = !!originalArr[0]
35          if (isApiDataReady) {
36              const itemNames = []
37              const itemPercentages = []
38              for (let item of originalArr) {
39                  itemNames.push(item.itemName)
40                  itemPercentages.push(item.numberOfOrders)
41              }
42              return { itemNames, itemPercentages }
43          } else {
44              return 0
45          }
46      }
47

```

Figure 5: une partie du code du composant pour les produits les plus commandés

Après consommation de ces web services, les données qui s’y trouvent sont formatées pour s’adapter aux format dont le module

npm `HighCharts` peut lire et générer des graphiques. Ils sont ensuite affichés dans la fonction `render()` de chaque composant

```

92     return (
93       <
94         <HighchartsReact highcharts={Highcharts} options={options} />
95       </>
96     )

```

Figure 6: affichage des graphique grace au module HighCharts

### 3. Rendu finale de DASHBOARD

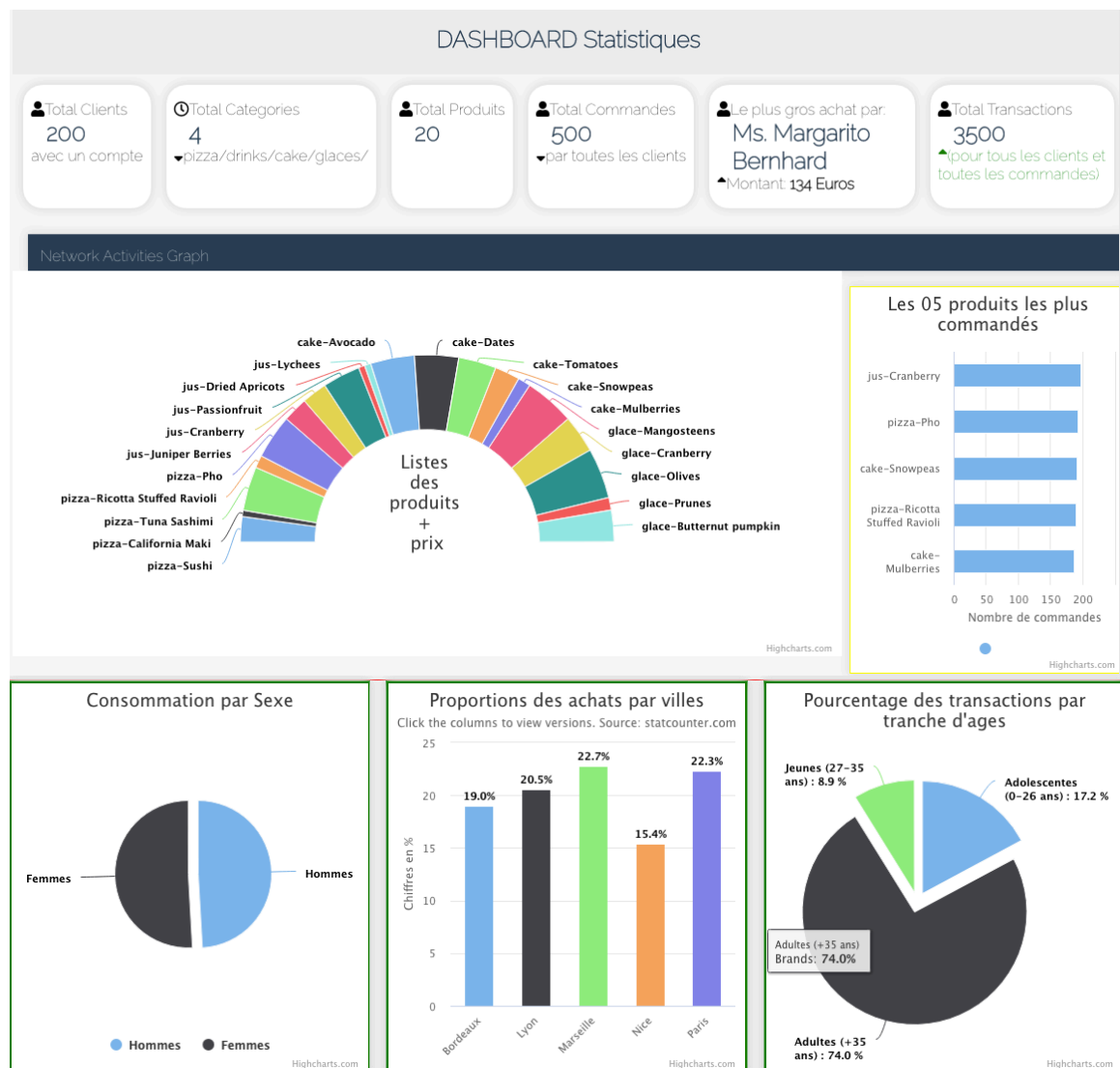


Figure 7: Rendu finale DASHBOARD

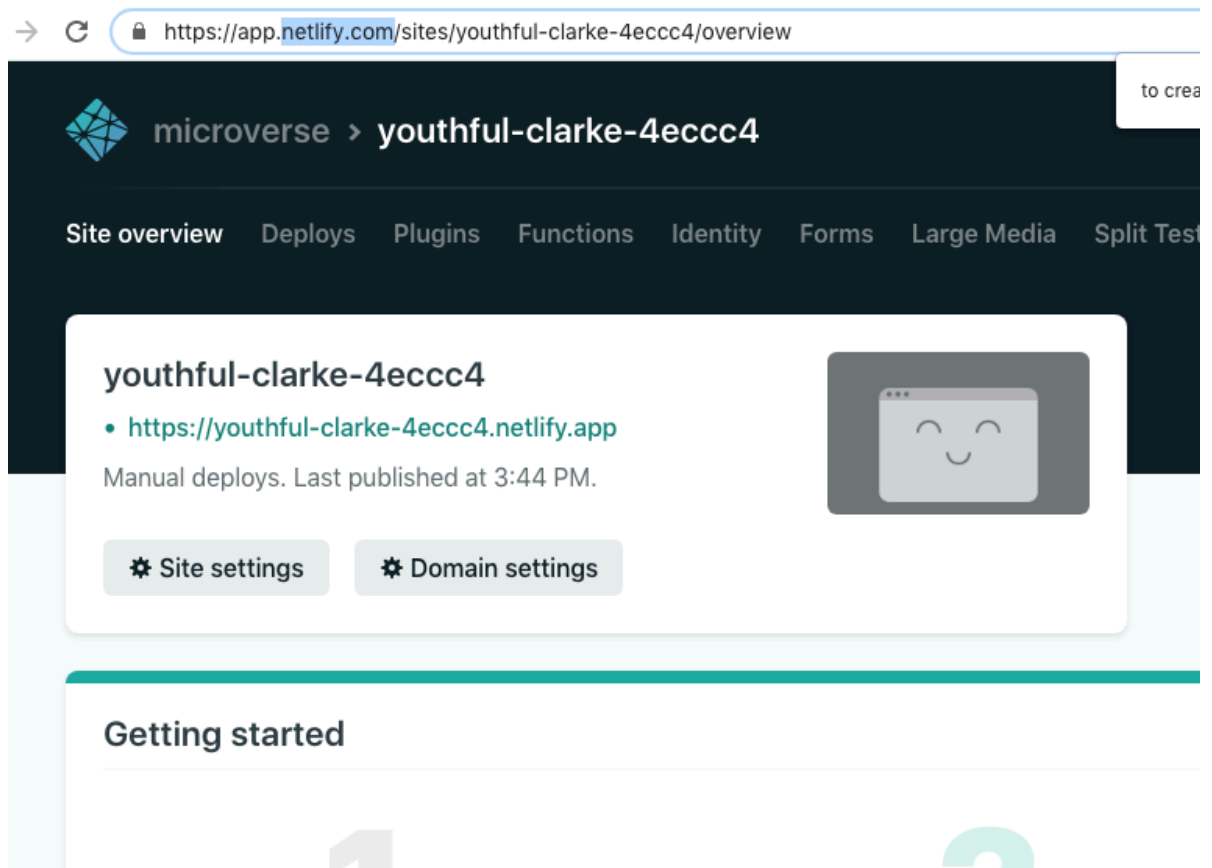
## IV – DEPLOIEMENT

Le déploiement permet de mettre en

ligne une application pour pouvoir le tester par le grand public ou faire des tests par l'équipe de développement. Il faut noter que avant le déploiement, les deux (02) projets sont stockés sous github. Cette étape est nécessaire parce que certain plateforme de déploiement recupère les codes sources a partir de Github.

### 1. Deploiement de Dashboard

Dashboard a été déployé via la plateforme <https://netlify.com>. Lorsqu'il est temps de passer votre application en production, avoir plusieurs fichiers JavaScript ou CSS n'est pas idéal. Lorsqu'un utilisateur visite votre site, chacun de vos fichiers nécessitera une requête HTTP supplémentaire, ce qui ralentira le chargement de votre site. Donc, pour remédier à cela, vous pouvez créer un «build» de notre application, qui fusionne tous vos fichiers CSS en un seul fichier, et fait de même avec votre JavaScript. De cette façon, vous réduisez le nombre et la taille des fichiers que l'utilisateur obtient. Pour créer ce «build», on utilise la commande '`npm run build`'. Ceci va créer un repertoire « *build/* ». Il faut créer un compte sur Netlify et ainsi copier ce dossier dans le dashboard de son compte personnelle. Netlify va ensuite généré une URL à partir de la quel le site déployé peut être consulté.



Le dashboard est donc accessible via l'URL <https://youthful-clarke-4eccc4.netlify.app>

## 2. Deploiement de “LA PECHE”

Heroku fournit également des buildpacks personnalisés, dans lesquels les développeurs peuvent déployer des applications dans plusieurs langage de programmation. Pour cette raison, Heroku est une plateforme polyglotte.

Les étapes/commandes suivantes sont nécessaires pour déployer une application Ruby on Rails sous Heroku :

- `bundle install --without production` : pour installer toutes les bundles configurés dans le fichier `Gemfile` du projet;

- `'heroku login'` pour se connecter sous heroku afin de pouvoir saisir le reste des commandes dans son espace personnel ;
- `'heroku keys:add'` pour configurer la clé SSH nécessaire pour pouvoir saisir la suites des commandes Heroku sans entrer son mot de passe tous le temps. Elle fonctionne exactement comme Github;
- `'heroku create'` : ceci permet la creation d'une URL sous Heroku de l'application qu'on veut deployer;
- `'git push heroku master'`: ceci permet le deploiement effective des codes sources dans le repertoire du projet dans Heroku. En effet, cette commande va transferer les codes sources de la branche 'master' du projet dans Github pour la branche 'master' dans Heroku
- `'heroku open'` c'est la commande qui va lancer le projet deployé sous ton navigateur en ouvrant l'URL généré par Heroku;

# CONCLUSION

## 1. Leçons apprises

J'ai énormément appris de ce projet. J'ai avancé dans ma capacité à analyser un problème et apporter des solutions informatiques et piloter le projet de bout en bout. Je peux relever les points positifs suivants :

- analyse des besoins des clients;
- faire le choix des technologies à utiliser;
- modélisation du système d'information et conception de la base de données
- une bonne capacité technique à manipuler plusieurs technologies front-end et back-end telles que ceux utilisés dans "LA PÊCHE" et "DASHBOARD"
- l'ouverture d'esprit, la patience et le courage pour faire face aux bugs dans le processus de développement, faire des recherches dans les forums sur internet ou demander de l'aide des camarades de classes ou professeurs WEBITECH afin d'apporter une solution aux difficultés.

## 1. Les améliorations futures

Malgré ces leçons apprises, ce projet a encore beaucoup de points pouvant être améliorés tels que :

### I. La sécurité

La sécurité doit être améliorée dans ces projets. Nous pouvons relever le fait que avec la mise en place des web services, il faudra

dans le future travailler sur un système d'authentification OAUTH2 entre le serveur back-end et le front-end dans un scénario ou une organization ne veut pas que ses webservices soient accessibles par le grand publique.

**OAuth** est un protocole d'autorisation ouvert, qui permet d'accéder aux ressources du propriétaire de la ressource en activant les applications clientes sur les services HTTP tels que Facebook, GitHub, etc. Il permet le partage des ressources stockées sur un site vers un autre site sans utiliser leurs identifiants. Il utilise à la place des jetons ('token') de nom d'utilisateur et de mot de passe.

OAuth 2.0 est développé par le groupe de travail IETF OAuth, publié en octobre 2012.

### **Pourquoi utiliser OAuth 2.0?**

- Vous pouvez utiliser OAuth 2.0 pour lire les données d'un utilisateur à partir d'une autre application.
- Il fournit le flux de travail d'autorisation pour le Web, les applications de bureau et les appareils mobiles.
- Il s'agit d'une application Web côté serveur qui utilise le code d'autorisation et n'interagit pas avec les informations d'identification de l'utilisateur.

## **Caractéristiques d'OAuth 2.0**

- OAuth 2.0 est un protocole simple qui permet d'accéder aux ressources de l'utilisateur sans partager les mots de passe.
- Il fournit des flux d'agent utilisateur pour exécuter l'application client à l'aide d'un langage de script, tel que JavaScript. En règle générale, un navigateur est un agent utilisateur.
- Il accède aux données à l'aide de jetons au lieu d'utiliser leurs informations d'identification et stocke les données dans le système de fichiers en ligne de l'utilisateur tel que Google Docs ou le compte Dropbox.

## **Avantages d'OAuth 2.0**

- OAuth 2.0 est un protocole très flexible qui repose sur **SSL** ("Secure Sockets Layer" qui garantit que les données entre le serveur Web et les navigateurs restent privées) pour enregistrer le jeton d'accès utilisateur.
- OAuth 2.0 s'appuie sur SSL qui est utilisé pour garantir les protocoles de l'industrie de la cryptographie et est utilisé pour protéger les données.
- Il permet un accès limité aux données de l'utilisateur et permet d'accéder à l'expiration des jetons d'autorisation.
- Il a la capacité de partager des données pour les utilisateurs sans avoir à divulguer des informations personnelles.
- Il est plus facile à mettre en œuvre et fournit une authentification renforcée.



## II. Testes Unitaires :

Des testes devraient être écrites pour toutes les fonctionnalités implémentés dans le backend et une grande partie du Front-end.

Le test de logiciel est le processus d'évaluation et de vérification qu'une application fait ou ce qu'il est censé faire. Les avantages des tests incluent la prévention des bugs, la réduction des coûts de développement et l'amélioration des performances.

Il existe de nombreux types de tests logiciels, chacun avec des objectifs et des stratégies spécifiques:

- **Test d'acceptation:** vérifier si l'ensemble du système fonctionne comme prévu.
- **Test d'intégration:** s'assurer que les composants ou fonctions du logiciel fonctionnent ensemble.
- **Test unitaire:** valider que chaque unité logicielle fonctionne comme prévu. Une unité est le plus petit composant testable d'une application.
- **Test fonctionnel:** vérification des fonctions en émulant des scénarios d'entreprise, en fonction des exigences fonctionnelles. Les tests en boîte noire sont un moyen courant de vérifier les fonctions.
- **Test de performance:** test des performances du logiciel sous différentes charges de travail. Les tests de charge, par exemple, sont utilisés pour évaluer les performances dans des conditions de charge réelles.
- **Test de régression:** vérifier si les nouvelles fonctionnalités interrompent ou dégradent les fonctionnalités. Les tests

d'intégrité peuvent être utilisés pour vérifier les menus, les fonctions et les commandes au niveau de la surface, lorsqu'il n'y a pas de temps pour un test de régression complet.

- **Test de résistance:** test de la charge que le système peut supporter avant de tomber en panne. Considéré comme un type de test non fonctionnel.
- **Test d'utilisabilité:** valider dans quelle mesure un client peut utiliser un système ou une application Web pour effectuer une tâche.

### III. Amélioration de la performance

l'application "LA PECHE" a été testée avec une base de 200 clients, 04 catégories de produit, 20 produits différentes, environ 300 achats de produits dans 05 villes. Il faudra tester la performance de l'application dans un scénario avec beaucoup plus de clients et évaluer comment elle va fonctionner avec une grande quantité de données et si nécessaire optimiser les algorithmes pour plus de performance.

### IV. Rendre le projet plus flexible

Il serait intéressant de rendre Dashboard plus flexible de telle sorte qu'elle puisse être utilisée dans plusieurs projets. On pourra prévoir un fichier de configuration dans lequel les projets dont dashboard doit tracker sont listés dessus ainsi que les web services qui s'y trouvent et il va générer automatiquement les graphes.

## WEBOGRAPHIE

- [https://www.w3schools.com/sass/sass\\_intro.asp](https://www.w3schools.com/sass/sass_intro.asp)
- <https://blog.webdevsimplified.com/2021-05/cors/>
- <https://en.wikipedia.org/wiki/GitHub>
- <https://www.highcharts.com/>
- <https://www.thoughtco.com/what-is-javascript-2037921>
- <https://www.hostinger.com/tutorials/what-is-html>
- <https://reactjs.org>
- <https://www.netlify.com>
- <https://stackify.com/rails-migration-a-complete-guide/>
- <https://create-react-app.dev/docs/getting-started/>
- <https://oauth.net/2/>
- <https://www.ibm.com/topics/software-testing>

# **GLOSSAIRE**

API - Application Programming Interface

CSS - Cascading Style Sheets

JSON - JavaScript Object Notation

ORM - Object-Relational Mapper

PDF - Portable Document Format

POO - Programmation Orientée Objet

SGBD - Système de Gestion de Bases de Données

SQL - Structured Query Language

UML – Unified Modeling Language

URL - Uniform Resource Locator

WWW - World Wide Web

HTML - HyperText Markup Language XML eXtensible Markup Language

MVC - Model – View – Controller

CoC – Convention Over Configuration

PK – Primary Key

FK – Foreign Key

CORS - Cross-Origin Resource Sharing

SSL - Secure Sockets Layer