Rapport de PC3R

Système de Gestion de Location

Author :
Qiwei XIAN
Ruiwen WANG

Professeur :
Prof.Romain Demangeon

10 juin 2020

Table des matières

1 Introduction **3**

2 API Web choisie **3**

3 Mettre à jour les données **3**
3.1 Mettre à jour Weatherstack . . . 3
3.2 Mettre à jour Google Map . . . 4

4 Fonctionnalités **7**

5 Cas d'utilisations **8**
5.1 Module Compte . . . 8
5.1.1 Inscription . . . 8
5.1.2 Authentification . . . 8
5.1.3 Consulter l'espace du client . . . 8
5.1.4 Créer l'espace du client . . . 9
5.1.5 Modifier l'espace du client . . . 9
5.2 Module Ressource . . . 9
5.2.1 Consulter les ressources . . . 9
5.2.2 Ajouter la ressource . . . 9
5.2.3 Modifier la ressource . . . 10
5.2.4 Supprimer la ressource . . . 10
5.3 Module Demande . . . 10
5.3.1 Réserver des logements . . . 10
5.3.2 Consulter des demandes envoyées . . . 11
5.3.3 Consulter des demandes reçues . . . 11
5.3.4 Supprime des demandes envoyées . . . 11
5.3.5 Supprime des demandes reçues . . . 11
5.3.6 Accepter la demande . . . 12

6 Stockage des données **12**

7 Structure du serveur **12**
7.1 Package Servlet . . . 13
7.2 Package Bean et DAO . . . 13
7.3 Util . . . 14

8 Structure du client **14**
8.1 Le page jsp . . . 14
8.2 JavaScript . . . 14

9 Requêtes et réponse **14**

10 Conclusion **14**

11 Annexe **14**

12 Référence **14**

1 Introduction

Système de gestion de location est une plateforme communautaire de location et de réservation de logements personnels. Les fonctionnalités du système est ressemble à Airbnb. Il permet aux utilisateurs de louer leur propriétés immobilières et de réserver un logement de l'autre utilisateur.
Chaque l'utilisateur doit inscrire sur le système afin d'obtenir un compte, il bénéficie de la service du système en utilisant ce compte. Il peut créer une espace de client et enregistrer les informations publiques, chaque utilisateur peut consulter les informations des autres utilisateurs.
Il peut tourver des logements qui satisfait à ses beosins, par exemple sous la condition de période de location, la ville, le nombre des locataires, ainsi que le droit de fumer. L'utilisateur peut aussi gérer leur propriétés par ce systèmes, il d'abord ajoute ses logements à louer dans le compte et met les contraintes pour les locataires, par exemple le nombre des locataires ou l'interdiction de fumer, etc. En plus il peut les modifier ou supprimer comme il veut. Lors que l'utilisateur reçoit les demandes envoyées par les autres utilisateurs. Il a droit de la refuser ou accepter, mais si la demande a risque de causer un conflit, le système va le faire remarquer au propriétaire.

2 API Web choisie

Weatherstack Nous utilisons cette API pour aider les utilisateurs à savoir la météo la ville qu'il souhaite réserver.
L'API Weatherstack[1] est développée par une société britannique qui excelle en SaaS avec des sociétés comme Ipstack, Currencylayer, Invoicely et Eversign. Destiné principalement aux sites Web et aux applications mobiles qui cherchent à inclure un widget météo en direct à un coût minime, offre la météo en temps réel, la météo historique, la météo internationale, etc.
Intégration et format de l'application : l'API REST renvoie des réponses au format JSON et prend en charge les rappels JSONP. HTTPS est activé pour les abonnements payants.
Nous enverrons régulièrement les informations contenant la ville qu'utilisateur veut réserver au serveur wheather via l'Api. Puis,l'API REST renvoie des réponses au format JSON à notre serveur Notre serveur reçoit les informations au format JSON et affiche les informations météo sur la page.

Google Map Pour aider les utilisateurs à savoir où se trouvent ces propriétés immobilières,nous utilisons Google Maps pour localiser ces propriétés immobilières. Google Maps[2] est un service de cartographie en ligne. C'est un service disponible sur PC, sur tablette et sur smartphone qui permet, à partir de l'échelle mondiale, de zoomer jusqu'à l'échelle d'une habitation.
En utilisant l'Api Google Maps, on peut intégrer Google Maps sur notre site. Il nous propose une carte sur une interface graphique. Nous enverrons l'adresse de la propriété immobilière au serveur de Google via l'Api google maps. La interface graphique du Google maps sur le site sera localisées en fonction de cette adresse et affichera cette adresse.

3 Mettre à jour les données

3.1 Mettre à jour Weatherstack

Observons d'abord à quoi ressemblent les données json renvoyées de Weatherstack, comme montré dans Annexe d'image "cf.Données Json".

```
//--call getWheather after charge web page-------------------
html += '<script type="text/javascript">\
$(document).ready(autoRefreshWheather);\
</script>';
//---------------------
```

FIGURE 2 – c
f.Données Json

```
function autoRefreshWheather(){
    getWheather();
    setInterval(getWheather,300000);
}
```

FIGURE 3 – c
f.reFresh

Lorsque l'utilisateur entre dans une page de détail de la propriété immobilière :notre fonction js ajoutera en texture html une fonction **autoRefreshWheather** pour appeler régulièrement la fonction **getWheather**

La fonction **autoRefreshWheather** appelle la fonction **getWheather** toutes les 5 minutes.

La fonction **getWheather** envoie les informations correspondant à la ville du client au serveur et attend que le serveur renvoie les données json, puis affiche les données json sur html en appelant htmlWheather.
Après réception de la demande, le serveur établit une connexion avec le serveur Weatherstack. Le serveur Weatherstack jugera s'il faut accepter la demande en fonction de la clé d'accès et retournera les données json en fonction de la valeur de la requête après avoir accepté la demande. Le serveur renvoie ensuite les données json au client via la méthode resp.getWriter().Write.

Enfin, nous afficherons les résultats des données json sur html.

3.2 Mettre à jour Google Map

Chaque fois ,l'utilisateur entre dans une page de détail de la propriété immobilière, nous appellerons Google Maps pour localiser la propriété immobilière.Le lien ci-dessous **googleapis** appellera en façon callback la méthode de js :**initMap**. Il est à noter que le mot-clé **asyns** peut réaliser un chargement asynchrone, ce qui signifie que ce lien sera appelé après le chargement du HTML
Les fonctions initMap et geocodeAddress permettent à Google Maps de s'initialiser en fonction du

```
function getWheather() {
    var city = $("#city").attr("data-city");
    $.ajax({
        type: "POST",
        data: { dataCity: city },
        url: "Service?method=getWheather",
        success: function (result, status) {
            var str = result;
            var resp = JSON.parse(str);
            var html = htmlWheather(resp);
            $('#DivWheather').html(html);
        }, error: function (res) {
            var str = res;
            alert("error:" + str);
            alert("error=" + res.responseTest)
        }
    });
}
```

FIGURE 4 – c
f.getWheather

```
public static void sendData(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    String res = "";

    System.out.println("server recevoir Data ok----------------------------------------------------------------------");

    String GET_WHEATHER_URL = "http://api.weatherstack.com/current?access_key=" + Parameter.access_key + "&query=";
    String city = req.getParameter("dataCity");
    System.out.println(city);

    String query = URLEncoder.encode(city, StandardCharsets.UTF_8);
```

FIGURE 5 – c
f.setupConnection

```
function htmlWheather(WheatherJson) {
    var html =
        '<div class="container bootstrap snippet">\
            <div class="row">\
                <div class="col-sm-10">\
                </div>\
            </div>\
        </div>\
        <div class="col-sm-5">\
            <ul class="list-group">\
                <li class="list-group-item text-muted">Real time Wheather</li>\
                <li class="list-group-item text-right"><span class="pull-left"><strong>City</strong></span>\
                    '+ WheatherJson.request.query + '\
                </li>\
                <li class="list-group-item text-right"><span class="pull-left"><strong>Time</strong></span>\
                    '+ WheatherJson.location.localtime + '\
                </li>\
                <li class="list-group-item text-right"><span class="pull-left"><strong>Temperature</strong></span>\
                    '+ WheatherJson.current.temperature + '\
                </li>\
                <img src="'+WheatherJson.current.weather_icons+'"   />\
            </ul>\
        </div>';
    return html;
}
```

FIGURE 6 – c
f.htmlWheather

```
html += '<script async defer\
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDpVUVmmjb1fQP25RB5TCYR20_3WkKlCok&callback=initMap">\
</script>';//callback google map
```

FIGURE 7 – c
f.htmlWheather

```
function initMap() {
    var map = new google.maps.Map(document.getElementById('map'), {
        zoom: 15,
        center: { lat: -34.397, lng: 150.644 }
    });
    var geocoder = new google.maps.Geocoder();
    geocodeAddress(geocoder, map);
}

function geocodeAddress(geocoder, resultsMap) {
    //    var idd = $(this).attr("data-id");

    //var address = document.getElementById('adresse').value;
    var address = $("#address").attr("data-address");
     //alert(address);
    geocoder.geocode({'address': address}, function(results, status) {
      if (status === 'OK') {
        resultsMap.setCenter(results[0].geometry.location);
        var marker = new google.maps.Marker({
          map: resultsMap,
          position: results[0].geometry.location
            });
        } else {
            alert('Geocode was not successful for the following reason: ' + status);
        }
    });
}
```

FIGURE 8 – c
f.initMaps

lieu donné.

4 Fonctionnalités

Le système est séparer en trois modules, **Account**, **Demand**, **Resource**.
Le module **Account** sert à gérer les données de l'authentification et l'espace de l'utilisateur. Le module **Resource** permet de gérer les données des logements. Et le module **Demand** sert à gérer les données des demands.
Cela permet de faciliter le développement parallèle et la maintenance de l'application. On a réaliser les fonctionnalités de l'ajoute, de la suppression et de la consultation, ainsi que la modification de ces trois modules.

5 Cas d'utilisations

5.1 Module Compte

5.1.1 Inscription

Acteur : L'utilisateur
Contexte : L'utilisateur crée un compte.
Scénario principal :

1. L'utilisateur entre dans le page d'inscription.
2. L'utilisateur saisit l'identifiant et le mot de passe.
3. L'utilisateur clique Signup pour soumettre les information au serveur.
4. Le système répond le message de success et afficher sur l'écran.

Cas particulier :
4a. L'identifiant est déjà utilisé dans le système, le système affiche le message d'erreur.

5.1.2 Authentification

Acteur : L'utilisateur
Contexte : L'utilisateur s'authentifie dans le système par son compte.
Scénario principal :

1. L'utilisateur entre dans le page de l'authentification.
2. L'utilisateur saisit l'identifiant et le mot de passe.
3. L'utilisateur clique le button **Login**.
4. Le système vérifie l'identifiant et le mot de passe.
5. Le système rend le page web de **mainPage.jsp**, l'identifiant est stocké dans la session.

Cas particulier :
4a. L'identifiant n'est pas reconnu, le système affiche le message d'erreur.
4b. Le mot de passe n'est pas correspondant à l'identifiant, le système affiche le message d'erreur.

5.1.3 Consulter l'espace du client

Acteur : L'utilisateur
Contexte : L'utilisateur connecté veut consulter son espace du client.
Scénario principal :

1. L'utilisateur clique le button Profile dans mainPage.
2. Le système génère dynamiquement l'espace personnelle de l'utilisateur.

cas particulier :
2a. L'utilisateur n'a pas encore son espace du client, le système génère le page de creation d'espace, permet à l'utilisateur de créer son espace.

5.1.4 Créer l'espace du client

Acteur : L'utilisateur
Contexte : L'utilisateur consulte sont espace première fois Scénario principal :

1. L'utilisateur clique le button Profile première fois dans le mainPage.
2. Le système repond un message et génère dynamiquement le formulaire pour créer l'espace du client.
3. L'utilisateur saisit les informations publiques.
4. L'utilisateur clique le button **Créer**.
5. Le système crée l'espace du client et le stocke dans la base de données, ainsi que génère le page web de l'espace du client.

5.1.5 Modifier l'espace du client

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans son espace du client.
Scénario principal :

1. L'utilisateur clique le button **here**.
2. Le système génère un formulaire permet au client de modifier son espace.
3. Le système remplit le formulaire selon les informations actuelles de l'utilisateur.
4. L'utilisateur modifie ce qu'il veut changer.
5. L'utilisateur clique le button **Modify**.
6. Le système met à jour les informations de l'utilisateur.

5.2 Module Ressource

5.2.1 Consulter les ressources

Acteur : L'utilisateur
Contexte : L'utilisateur s'authentifié se trouve à mainPage, il veut savoir le détail du logement.
Scénario principal :

1. L'utilisateur clique le button **Your houses**.
2. Le serveur cherche dans la base de donnée et génère dynamiquement le page sert à afficher les informations du logement, du propriétaire.

5.2.2 Ajouter la ressource

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans le page **Your House**.
Scénario principal :

1. L'utilisateur clique le button **Add House**.
2. La page est générée dynamiquement par js permet à l'utilisateur de saisir les informations du logement.
3. L'utilisateur clique le button **Créer**.

4. Le système vérifie que le status de l'authentification de l'utilisateur est connecté.
5. Le système stocke les informations du logement et affiche un message de la réussite sur l'écran.

5.2.3 Modifier la ressource

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans la page du logement, il veut modifier les informations du logement.
Scénario principal :

1. L'utilisateur clique le button **here**.
2. Le système génère le formulaire de modification du logement.
3. Le système remplit automatiquement le formulaire par les informations actuelles du logement.
4. L'utilisateur modifie où il veut changer.
5. L'utilisateur clique le button **Modify**.
6. Le système met à jours les informations du logement et Le système affiche un message de la réussite sur l'écran.

5.2.4 Supprimer la ressource

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans mainPage, il veut supprimer quelque logement.
Scénario principal :

1. L'utilisateur clique le button **Your houses**.
2. Le système cherche toutes les propriétés de l'utilisateur et les affiche sur une page générée automatiquement.
3. L'utilisateur clique le button **delete** qui est à la même ligne avec le logement qu'il veut supprimer.
4. Le système supprime le logement dans la base de donnée et affiche le message de la réussite sur l'écran.

Cas particulier : 4a. Le logement est attaché par les demandes acceptées, le système ne peut pas le supprimer et affiche le message d'erreur sur l'écran.

5.3 Module Demande

5.3.1 Réserver des logements

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans mainPage, il veut réserver un ou plusieurs logements.
Scénario principal :

1. L'utilisateur saisit la ville où il veut faire un séjour, le type du logement, la période du séjour, le droit de fumer et le nombre des voyageurs.
2. L'utilisateur clique le button **Go**.

3. Le système cherche tous les logements qui peut satisfaire les besoins.
4. Le système filtre les logements dont le propriétaire est l'utilisateur lui-même.
5. Le système affiche les restes logements sur l'écran.
6. L'utilisateur clique le button **reserve** qui est sur la même ligne avec le logement que l'utilisateur veut réserver.

5.3.2 Consulter des demandes envoyées

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans mainPage, il veut consulter les demandes envoyées.
Scénario principal :

1. L'utilisateur clique le button **Sended demands**.
2. Le système cherche les demandes envoyées par l'utilisateur dans la base de donnée et affiche sur une page générée automatiquement.

5.3.3 Consulter des demandes reçues

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans mainPage, il veut consulter les demandes reçues.
Scénario principal :

1. L'utilisateur clique le button **Recieved demands**.
2. Le système cherche les demandes dont le logement est à l'utilisateur, et les affiche sur une page générée automatiquement.

5.3.4 Supprime des demandes envoyées

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans la page **Sended demands**, il veut supprimer quelque demande.
Scénario principal :

1. L'utilisateur clique le button **delete** qui est sur la même ligne avec la demande envoyée qu'il veut supprimer.
2. Le système supprime la demande dans la base de donnée et affiche le message de la réussite sur l'écran.

5.3.5 Supprime des demandes reçues

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans la page **Recieved demands**, il veut supprimer quelque demande.
Scénario principal :

1. L'utilisateur clique le button **delete** qui est sur la même ligne avec la demande reçue qu'il veut supprimer.
2. Le système supprime la demande dans la base de donnée et affiche le message de la réussite sur l'écran.

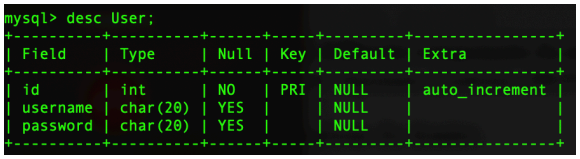


FIGURE 9 – c
f.tableUser

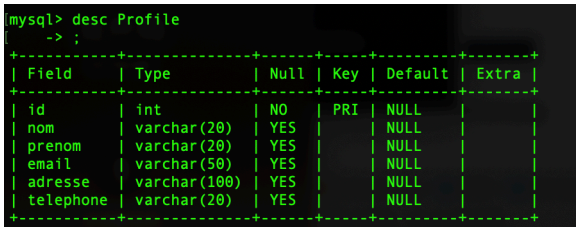


FIGURE 10 – c
f.tableProfile

5.3.6 Accepter la demande

Acteur : L'utilisateur
Contexte : L'utilisateur connecté se trouve dans la page **Recieved demands**, il veut accepter quelque demande.
Scénario principal :

1. L'utilisateur clique le button **accept** qui est sur la même ligne avec la demande reçue qu'il veut accepter.
2. Le système change l'état de demande de **Pending** à **Accepted**.
3. Le système actualise les demandes et affiche un message de la réussite sur l'écran.

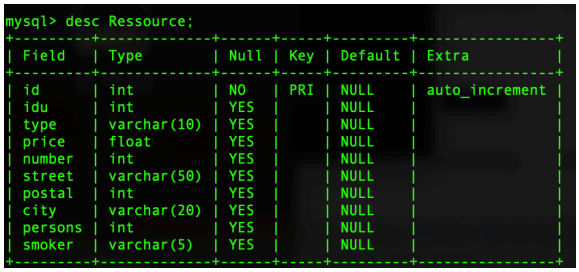
Cas particulier : 2a. La demande cause un conflit avec une autre demande, le système refuse de modifier la base de donnée et affiche un message d'erreur sur l'écran.

6 Stockage des données

Les données sont stockées dans la base de données SQL, avec 4 quatre tables, la structure de ces 4 tables est comme indiqué ci-dessous.

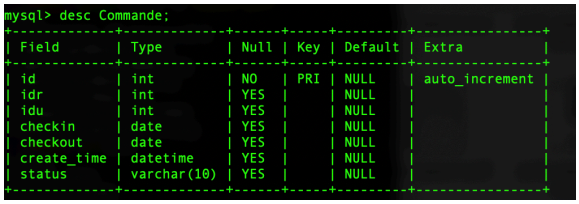
7 Structure du serveur

La serveur se compose de quatres packages, **Bean**, **DAO**, **servlet** et **util**.



```
mysql> desc Ressource;
+---------+-------------+------+-----+---------+----------------+
| Field   | Type        | Null | Key | Default | Extra          |
+---------+-------------+------+-----+---------+----------------+
| id      | int         | NO   | PRI | NULL    | auto_increment |
| idu     | int         | YES  |     | NULL    |                |
| type    | varchar(10) | YES  |     | NULL    |                |
| price   | float       | YES  |     | NULL    |                |
| number  | int         | YES  |     | NULL    |                |
| street  | varchar(50) | YES  |     | NULL    |                |
| postal  | int         | YES  |     | NULL    |                |
| city    | varchar(20) | YES  |     | NULL    |                |
| persons | int         | YES  |     | NULL    |                |
| smoker  | varchar(5)  | YES  |     | NULL    |                |
+---------+-------------+------+-----+---------+----------------+
```

FIGURE 11 – c
f.tableRessource



```
mysql> desc Commande;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| id          | int         | NO   | PRI | NULL    | auto_increment |
| idr         | int         | YES  |     | NULL    |                |
| idu         | int         | YES  |     | NULL    |                |
| checkin     | date        | YES  |     | NULL    |                |
| checkout    | date        | YES  |     | NULL    |                |
| create_time | datetime    | YES  |     | NULL    |                |
| status      | varchar(10) | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
```

FIGURE 12 – c
f.tableCommande

7.1 Package Servlet

Ce package possède les classes qui contient les **bisiness logic functions**. Lors que l'utilisateur produit un événement comme appuie sur le bouton. Le client demande asynchronement au serveur par un URL. La classe **Service** sert à vérifier la demande et appeler la fonction correspondante.
Les fonctions pour gérer les informations du compte et s'authentification, ainsi que l'inscription sont réalisées dans **Client**
Ressource ont les méthodes servent à gérer les informations des logements.
Gopage permet de redirect à une autre page web.
Demande sert à gérer les information des demandes.
Wheather peut appeler API externe afin de rendre les informations du météo.

7.2 Package Bean et DAO

Dans la partie serveur, on utilise le pattern **DAO** afin d'encapsuler la manipulation de la base de donnée.
Le pattern **DAO** est l'abréviation de **Data Access Object**, pour chaque l'objet, on doit réaliser deux classes, **objetBean** et **objetDAO**.

- Pour l'objetBean, il n'y a que les méthodes **get** et **set** et toutes les attributes sont privées, chaque attribute corresponde à chaque colonne de la table de base de donnée.
- Pour l'objetDAO, il encapsule les manipulations, par exemple le stockage, la modification, la consultation et la suppression des données dans la base de données.

Par conséquence on n'a pas besoin de répéter les instructions SQL, il ne suffit d'appeler les interfaces proposées par l'objetDAO. Ce parttern facilite de manipuler la base de donnnée et évite beaucoup d'erreur.

7.3 Util

Ce package a seulement deux classes, **Parameter** stocke des variables globales, par exemple l'identifiant et le mot de passe de base de données. **ToJson** est un objet pour créer une chaine de caractères en format de JSON.

8 Structure du client

8.1 Le page jsp

Cette application possède deux pages jsp, une est la page d'accueil. Après l'authentification de l'utilisation, l'utilisateur est redirect à la mainPage.jsp. Tous les contenus sont affichés sur cette page.

8.2 JavaScript

Nous avons séparer 7 fichiers **js** dans le projet.

1. **apiExterne.js** sert à l'appeler les APIs externes, GoogleMap et wheatherStack.
2. **client.js** gère les événements du module **Account**. Par exemple, envoyer la requête au serveur pour créer et modifier l'espace de l'utilisateur.
3. **demande.js** contient les fonctions concernant du module **Demande**. Par exemple, réserver un logement, accepter et supprimer la demande.
4. **Ressource.js** sert à envoyer la requête concernant du module **Resource**, comme la consulation, la modification et la creation d'un logement.
5. **htmlGenerator.js** permet de générer dynamiquement les contenus par un Json.
6. **page.js** contrôle les effets de visualisation de page.
7. **mainPage.js** attache tous les fonctions aux boutons.

9 Requêtes et réponse

On utilise mainPage.js pour attacher les fonctions sur les boutons. Lors que les boutons sont appuyés, la fonction correspondante envoie asynchronement une requête aux client, la requête est un URL "Service ?methode=xxxx&parametre". Lors que le serveur reçoit ce URL, la classe **Service** peut savoir quelle fonction le client veut appeler. En plus, après l'exécution de la fonction correspondante, le serveur rend un json au client. Dès que le client reçoit le json, il va générer dynamiquement le contenu par la fonction de **htmlGenerator.js**.

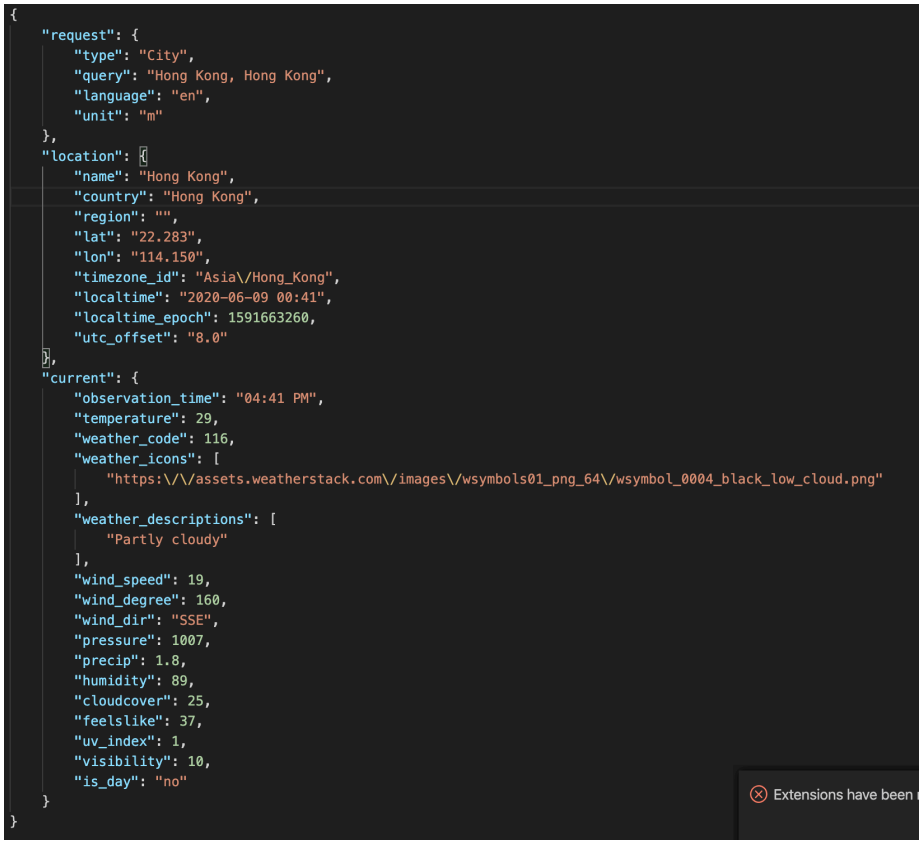


FIGURE 13 – c
f.Données Json

10 Annexe

11 Référence

[1]Weatherstack,[Online].Available :https ://weatherstack.com/

[2]Google Maps,[Onlone].Available :https ://cloud.google.com/maps-platform/