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

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".

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**

´

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

cf.Données Json

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

´

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

cf.reFresh

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.

```
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)
        }
    });
}
```

cf.getWheather

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.

```
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);
```

cf.setupConnection

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

```
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;
}
```

cf.htmlWheather

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

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

cf.htmlWheather

Les fonctions initMap et geocodeAddress permettent à Google Maps de s'initialiser en fonction du lieu donné.

```
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);
        }
    });
}
```

cf.initMaps

4 Fonctionnalités

Le système est séparer en trois modules, **compte**, **demande**, **propriété**. Cela permet de faciliter le développement 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 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.

5.2 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 pour soumettre les informations au serveur.
4. Le serveur vérifie l'identifiant et le mot de passe.
5. Le serveur 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.

5.3 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 serveur génère dynamiquement l'espace personnelle de l'utilisateur.

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

5.4 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 serveur 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 afin de soumettre les information.
5. Le serveur crée l'espace du client et le stocke dans la base de données, ainsi que génère le page web de l'espace du client.

5.5 Modifier l'espace du client

Acteur : L'utilisateur Contexte : L'utilisateur connecté se trouve dans le mainPage. Scénario principal :

1. L'utilisateur clique le button Profile
2. Le serveur

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.

´

```
mysql> desc User;
+----------+----------+------+-----+---------+----------------+
| Field    | Type     | Null | Key | Default | Extra          |
+----------+----------+------+-----+---------+----------------+
| id       | int      | NO   | PRI | NULL    | auto_increment |
| username | char(20) | YES  |     | NULL    |                |
| password | char(20) | YES  |     | NULL    |                |
+----------+----------+------+-----+---------+----------------+
```

cf.tableUser

´

```
mysql> desc Profile
    -> ;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| id        | int          | NO   | PRI | NULL    |       |
| nom       | varchar(20)  | YES  |     | NULL    |       |
| prenom    | varchar(20)  | YES  |     | NULL    |       |
| email     | varchar(50)  | YES  |     | NULL    |       |
| adresse   | varchar(100) | YES  |     | NULL    |       |
| telephone | varchar(20)  | YES  |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
```

cf.tableProfile

´

```
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    |                |
+---------+-------------+------+-----+---------+----------------+
```

cf.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    |                |
+-------------+-------------+------+-----+---------+----------------+
```

cf.tableCommande

7 Structure du serveur

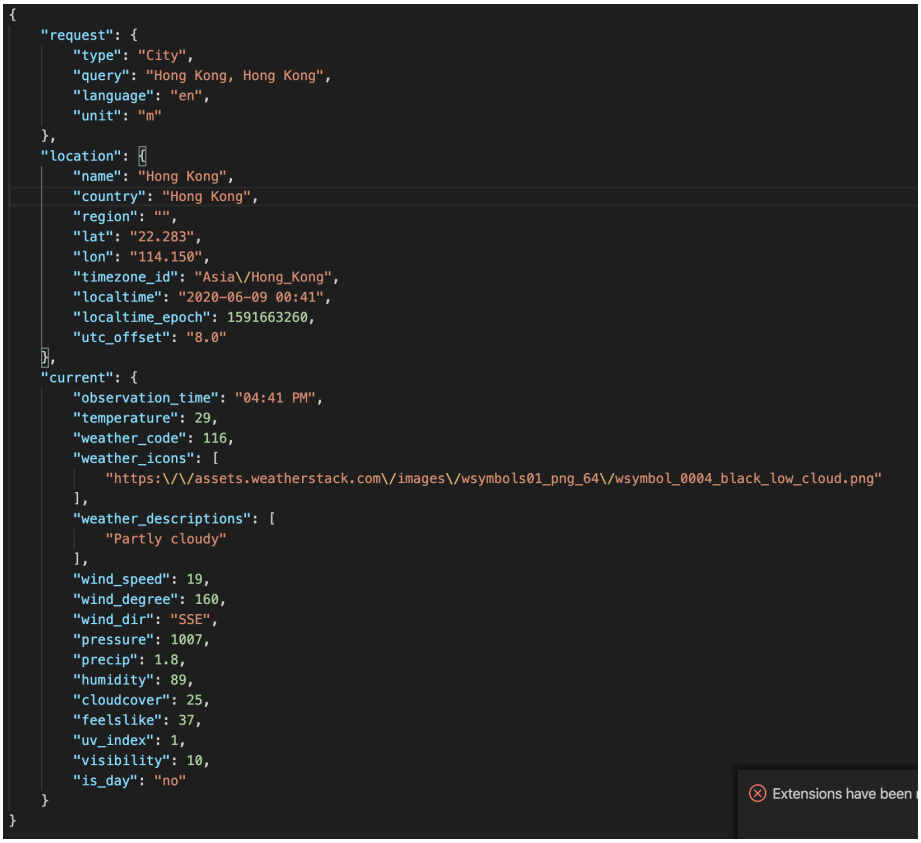
8 Structure du client

9 Requêtes et réponse

10 Conclusion

11 Annexe

´



cf.Données Json

12 Référence

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

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