

PROJET GESTION DE PARKING  
(VUEJS/CAPACITOR)

*Réalisé par :*  
*HAMON Cyril*  
*BOIRO Mamadou Yero*

Année universitaire  
2020-2021

## Table des matières

Introduction .....	1
Présentation de l'application .....	1
Choix des implementations effectués .....	5
Conclusion .....	10
Annexe .....	11

# Introduction

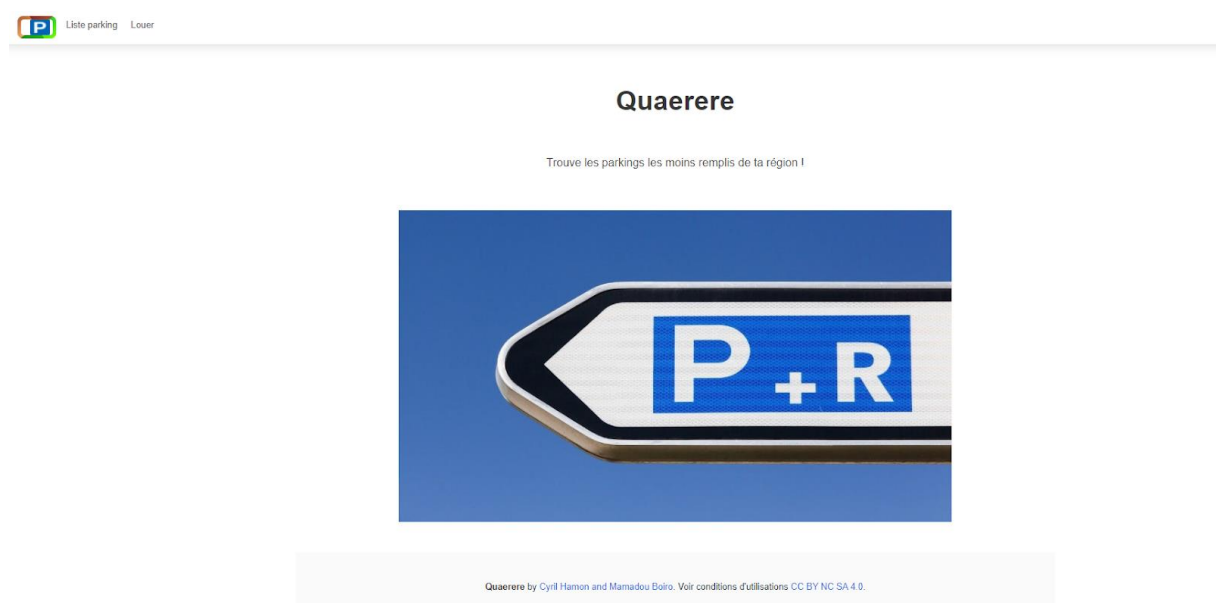
Dans le cadre du module de développement d'applications mobiles, nous avons réalisé une application web de gestion de parking sur la métropole nantaise. Le jeu de données que nous utilisons est fourni par Nantes Métropoles et est accessible à tout public. Nous avons également mis en place une application mobile afin de mettre à disposition notre application à un plus large public et permettre son utilisation à chaque moment.

## Présentation de l'application

Notre application de gestion de parking présente deux fonctionnalités principales :

- La recherche d'une place de parking proche de notre position
- La mise en location d'une place de parking

La page d'accueil de l'application se présente comme ci-dessous :

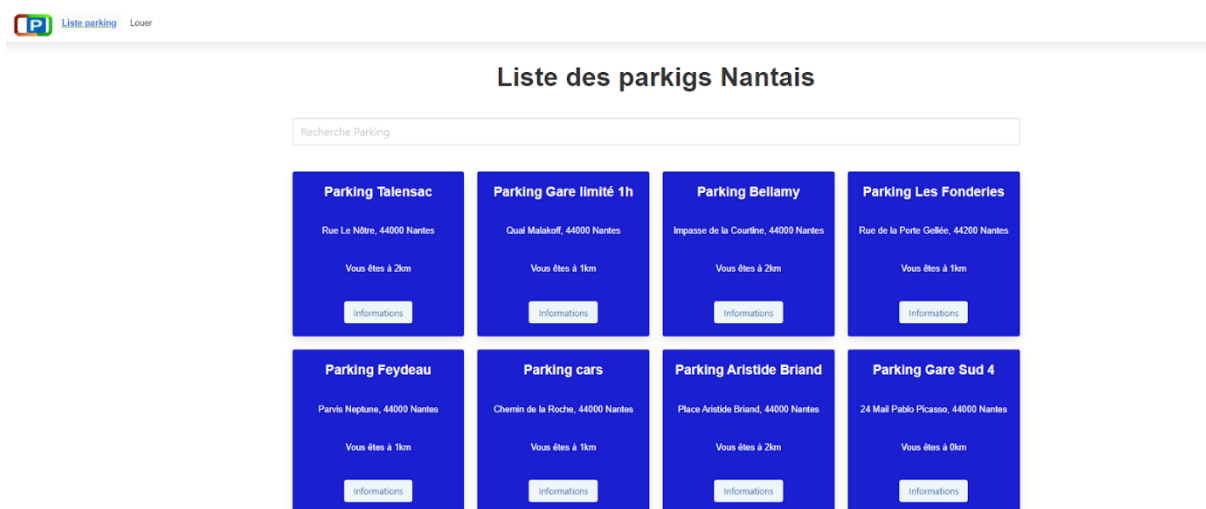


C'est la page vitrine du site qui présente notre logo mais aussi les différentes fonctionnalités que nous mettons à disposition des utilisateurs et leurs conditions d'utilisation.

Le bouton « Liste parking » permet d'accéder à la page Parking-List.



Le bouton « Louer » permet d'accéder au formulaire de mise en location d'un parking privé.

La page Parking-List se présente comme suit :



Cette page affiche les parkings de la ville de Nantes, il est également possible de rechercher un parking à partir de son nom. Le bouton « informations » permet d'afficher les détails d'un parking via la page « parking-detail ». La recherche d'un parking sous smartphone dans la barre de recherche doit se faire avec une validation clavier.

La page « parking-detail » présente toutes les informations utiles concernant un parking : les horaires d'ouvertures, le type de véhicule accepté, son accessibilité en transport en commun, les moyens de paiements. Il est également possible d'accéder directement au site web du parking ou d'ouvrir sa position sur Google Maps via les boutons « consulter site web » et « ouvrir dans Google Maps ».

## Parking Talensac

Rue Le Nôtre, 44000 Nantes

Parking de Centre-ville. Parking Accessible 6h30 - 23h00 du lundi au samedi, le dimanche de 7h30 à 20h, 7j/7 pour les abonnés.

**Informations complémentaires**

Voiture(340) Moto(6) Voiture électrique(6) Vélo(6)

**Accès transports en commun**

Station Tramway Lignes 3 "50 Otages" Station Bicloo n°1 "Préfecture" (équipée CB) Station Marguerite "50 Otages" (Véhicule en libre service).

**Moyens de paiement**

CB en borne de sortie, Espèces, Total GR

[Consulter site web](#)

[Ouvrir dans Google Maps](#)

Notre application offre également la possibilité de mettre en location sa place de parking privée via le formulaire présent sur la page « Louer ». Il faut saisir les informations de la location tel que le prix, le type (garage, place extérieure, stationnement privé, ...) mais également une photo de la place à louer. Aucun enregistrement de location n'est fait ici car ce n'était pas l'objectif de l'application. Ce principe nous permettait d'ajouter des plugins capacitor dans notre application mobile.

### Mon petit parking

Nom de la location

Pouvoir de l'ami

User

User

Email

Exemple@mail.com

Type

Garage

Prix

€ 0,0

Message

Commentaire

☐ I agree to [Je suis humain](#)

108/108

Choisir une photo

<https://bulma.io/images/plac...>

Enregistrer

Ensuite, par le biais de Capacitor, nous avons généré une application mobile. Nous avons utilisé 4 plugins Capacitor. Le premier correspondant à la géolocalisation, nous reprenons le même principe que vue en cours, qui nous permet de nous situer par rapport à la localisation d'un parking. Les trois suivants sont utilisés sur la page Loué. Tout d'abord, le plugin caméra permettant de prendre une photo pour l'ajouter

à notre parking. Ensuite, le plugin Toast, qui permet d'afficher un message sous Android lorsque le formulaire n'est pas correctement rempli et pour finir le plugin Local notification, qui envoie une notification à l'utilisateur sur son smartphone lorsqu'il enregistre sa localisation.



## Choix d'implémentation effectués :

Nous avons décidé de partir du projet « CompostMap » vu en cours, car cela nous permettait de partir d'une structure déjà établie et de réutiliser certains éléments en les adaptant à notre projet. Nous avons suivi l'ensemble des points listé dans le sujet de TP, mise en avant des différentes mécaniques de Vue présentées en cours, un client-side routing, une application responsive avec un rendu graphique soigné et utilisant bulma et une API Rest et 4 plugins Capacitors.

- **La page parking-list :**

Nous avons décidé d'afficher la liste des parkings sous forme de composants, comme vu en cours. Chaque composant représente un parking unique comme suit :

```

<div class="columns is-multiline">
  <Parking
    v-for="parking in parkings"
    :key="parking.fields.idobj"
    class="column is-half-tablet is-one-third-desktop is-one-quarter-widescreen"
    :parking="parking"
    :userLocation="currentUserLocation"
  />
</div>

```

La liste des parkings est récupérée dans **mounted()** via appel de l'api Rest fournie par Nantes Métropole. La définition de cette méthode dans **mounted()** permet de récupérer la liste des parkings avant la création du composant « parking ».

```

mounted() {
  axios
    .get(
      "https://data.nantesmetropole.fr/api/records/1.0/search/?dataset=244400404_parkings-publics-nantes&q=&fac"
    )
    .then((response) => {
      this.parkings = response.data.records;
      this.parkingSearcher = response.data.records;
    })
    .catch((error) => {
      console.log(error);
    });

  Geolocation.getCurrentPosition().then((location) => {
    this.currentUserLocation = location.coords;
  });
},

```

Nous définissons ensuite une fonction **searchParking()** dans **methods()** qui permet, en fonction des valeurs saisies dans le champs « Recherche Parking », de retrouver le parking dont le nom correspond à la recherche. Cette recherche se base sur la liste préalablement récupérée dans **mounted()**, cela nous permet de faire un seul appel à l'API Rest à l'initialisation de la page. Il n'est pas nécessaire dans notre cas de refaire un appel à l'api pour la recherche car le nombre de parking récupéré n'est pas conséquent (moins de 20).

```

methods:{
  searchParking:function(){
    if(this.q != ""){
      this.parkings = this.parkingSearcher.filter(parking=>{
        return parking.fields.nom_complet.toLowerCase().includes(this.q.toLowerCase());
      });
    }
    else{
      axios
        .get(
          "https://data.nantesmetropole.fr/api/records/1.0/search/?dataset=244400404_parkings-publics-nantes&q=&facet=libcategorie&facet=libtype&"
        )
        .then((response) => {
          this.parkings = response.data.records;
        })
        .catch((error) => {
          console.log(error);
        });
    }
  },
},

```

- **Le component Parking :**

Les informations spécifiques à chaque parking sont gérées par le component Parking (Parking.vue).

Nous utilisons la fonction **calcDistance()** définie dans le component Parking afin de calculer la distance entre l'utilisateur de l'application et le parking. La position actuelle de l'utilisateur est récupérée via le plugin « Geolocation » fournie par Capacitor. L'utilisation de cette méthode sur l'application android nécessite d'accorder les permissions préalables dans AndroidManifest.xml. La position du parking est récupérée durant l'appel à l'api Rest.

```
methods: {  
  // Les méthodes suivantes proviennent de ce qui a été fait en cours.  
  // Elles permettent de calculer la distance entre notre position et la localisation d'un parking.  
  calcDistance(lat1, lon1, lat2, lon2) {  
    var R = 6371; // km  
    var dLat = this.toRad(lat2 - lat1);  
    var dLon = this.toRad(lon2 - lon1);  
    lat1 = this.toRad(lat1);  
    lat2 = this.toRad(lat2);  
  
    var a =  
      Math.sin(dLat / 2) * Math.sin(dLat / 2) +  
      Math.sin(dLon / 2) *  
      Math.sin(dLon / 2) *  
      Math.cos(lat1) *  
      Math.cos(lat2);  
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));  
    var d = R * c;  
    return Math.round(d);  
  },  
  // Converts numeric degrees to radians  
  toRad(Value) {  
    return (Value * Math.PI) / 180;  
  },  
}
```

Pour afficher les détails concernant un parking, nous avons défini un routing lors du click sur le bouton information comme suit :

```
<router-link :to="{ name: 'ParkingDetails', params: { id: parking.fields.idobj } }">  
  <br>  
  <br>  
  <button class="button is-info is-light">  
    Informations  
  </button>
```

- **La page Parking-Detail**

On est ensuite redirigé vers la page ParkingDetails, lors du click sur le bouton « Informations » qui affiche le contenu de l'objet parking passé en paramètre. Nous utilisons des V-IF pour afficher le type de véhicules acceptés dans chaque parking comme suit :

```
<div class="content">  
  <p class="title is-6">Informations complémentaires</p>  
  <!-- Les véhicules possible au stationnement et leur capacité -->  
  <span class="tag is-primary is-light" v-if="parking.fields.capacite_voiture > 0">Voiture({{parking.fields.capacite_voiture}})</span>  
  <span class="tag is-danger is-light" v-if="parking.fields.capacite_moto > 0">Moto({{parking.fields.capacite_moto}})</span>  
  <span class="tag is-link is-light" v-if="parking.fields.capacite_vehicule_electrique > 0">Voiture électrique({{parking.fields.capacite_ve}}</span>  
  <span class="tag is-success is-light" v-if="parking.fields.capacite_velo > 0">Vélo({{parking.fields.capacite_moto}})</span>  
  <div v-if="parking.fields.acces_transports_communs != ''">
```



- **La page Louer :**

Une des fonctionnalités importante de notre application est la possibilité de faire louer une place de parking en remplissant un formulaire, mais aussi en ajoutant une photo de la place de parking à faire louer.

Pour Mettre en place le formulaire, nous avons récupéré du code de Bulma et l'adapté en fonction de notre besoin.

Nous avons utilisé le plugin **Capacitor Camera()** qui permet de récupérer une photo, comme suit :

```
// Méthode permettant de récupérer une photo.
async TakePhoto(){
  const image = await Camera.getPhoto({
    quality: 90,
    allowEditing: true,
    resultType: CameraResultType.Uri
  });
  var imageUrl = image.webPath;
  this.image = imageUrl;
},
```

Ensuite, dans le Template, nous avons utilisé le binding d'événement pour lier le texte « choisir photo » à notre fonction TakePhoto().

```
<input class="file-input" v-on:click="TakePhoto()" name="resume">
```

Nous avons également défini une méthode « **NotifValidation()** » permettant de créer une notification lorsque l'utilisateur remplit son formulaire et appuie sur le bouton « Enregistrer ». Cette méthode utilise les plugins Capacitor « LocalNotifications » et « Toast » qui permettent d'envoyer une notification à l'utilisateur qui lui indique si oui ou non la proposition de location de parking est enregistrée.

```

async NotifValidation(){
  if(this.nom != "" && this.user != "" && this.mail != "" && this.prix != "" && this.val == true){
    await LocalNotifications.schedule({
      // correspond au corps de la notification
      notifications:[
        {
          title : 'Enregistrement',
          body : 'Vous avez enregistré : ' + this.nom,
          id: 1,
          iconColor: '#B8B8B8'
        }
      ]
    })
    this.nom = ""
    this.user = ""
    this.mail = ""
    this.prix = ""
    this.com = ""
    this.val = false
    this.image = "https://bulma.io/images/placeholders/128x128.png"
    // Nous aurions preferer afficher une alerte sur web et le toast sur smartphone
    // mais l'alerte s'affiche aussi sur smartphone.
    Toast.show({
      text: 'Parking enregistré' });
    alert("Parking enregistré")
  }else{
    Toast.show({
      text: 'Il manque des valeurs' });
    alert("Il manque des valeurs")
  }
},

```

- **Gestion du style :**

Nous utilisons les fonctionnalités fournies par le Framework Bulma afin de gérer le style de notre application, comme vu en cours. Cela nous a permis de rendre l'application responsive sur tablette, desktop, portrait et paysage.

```
<div class="column is-four-fifths-mobile is-two-thirds-tablet is-half-desktop is-one-third-widescreen is-full-fullhd">
```

- **Routing dans l'application :**

Le routing de notre application est gérée dans router/index.js. Nous proposons 4 routes différentes dont certaines qui requièrent des paramètres, tels que le routing vers la page détail.

```

{
  path : '/parking-detail/:id',
  name : 'ParkingDetails',
  props: true,
  component : ParkingDetails
},

```

Cet ID est utilisé lors de la création du component Parking présenté précédemment.

## Conclusion :

La réalisation de ce projet a été très intéressante pour nous, car il nous a permis de mettre en pratique tout ce que nous avons appris en cours.

Une des difficultés que nous avons rencontrées durant ce projet a été l'affichage de l'alerte, lors du remplissage du formulaire, seulement sur la page Web, et le Toast, seulement sur l'application mobile. Mais dans notre cas, le Toast et l'alerte s'affichent sur l'application mobile.

# Annexes :

Quelques images de notre application Android générée via Capacitor :

