

# Let's Go App

<b>Introduction</b>	<b>2</b>
<b>Native SWIFT - Anthony Loroscio</b>	<b>3</b>
Comment l'exemple a été réalisé	3
Accès au GPS	3
Accès et représentation de la boussole	4
Avec quel outil de développement, de tests ?	5
Comment déploie-t-on et exécute-t-on l'exemple ?	6
Comment teste-t-on les capacités d'adaptations ?	7
<b>CrossPlatform avec Ionic 3 - Thomas Monzein</b>	<b>8</b>
Environnement de développement	8
Node.Js	8
Ionic et Cordova	8
Android Studio (Recommandé)	8
Xcode pour IOS (Seulement pour Mac OS)	9
Créer un premier projet	9
Développer avec Ionic	9
Ajouter des pages	10
Naviguer entre les pages	10
Ajouter des modules complémentaires	10
Exécuter l'application	11
Capacités d'adaptations	12
L'application et sa capacités d'adaptation	12
Autopsie de la boussole	13
<b>Web component avec Vue.js - Alicia Marin</b>	<b>15</b>
Outil de développement	15
Installer le projet	15
Générer le projet	15
Exécuter le projet	16
Déployer le projet	16
Description d'un composant	16
Réalisation	17
L'Accueil	17
La Navigation	18
Les Horaires	19
Adaptation	20

Composants réutilisables	20
L'onglet - Tabs	20
Le tableau de données - Datatable	21
Message de détails - Modal	24
Champ de text - Input	27
<b>Responsive Web Design avec Bootstrap - Thibaut Terris</b>	<b>28</b>
Installation	28
Réalisation	29
La structure	29
Les fondations	30
Page de recherche	30
Adaptation au contexte d'usage	31
Adaptation à l'environnement	32
Page de la carte	33
Adaptation au contexte d'usage	35
Page des horaires	37
Adaptation au contexte d'usage	38
Avantages	40
Limites	40
Bibliographie	40
<b>Conclusion</b>	<b>41</b>
Swift	41
Ionic	41

# Introduction

Let's Go App est une application de réseau de bus.

Nous avons organisé notre développement en deux parties que nous comparerons : la partie web et la partie mobile.

La partie mobile se base sur une application qui accède à différents capteurs (GPS et boussole). Une version est développée en natif iOS avec SWIFT et une version hybride développée grâce à Ionic. Le scénario de l'application prévoit qu'un utilisateur veut se déplacer grâce aux bus. Son application trouve l'arrêt le plus proche et remplit le champ de départ. Le champ d'arrivée peut être rempli grâce à la position d'un ami. Une boussole intégrée à l'application permet à l'utilisateur de s'orienter et de se déplacer jusqu'à l'arrêt de bus le plus proche. Lorsqu'un trajet est validé, l'utilisateur peut utiliser la boussole pour se déplacer d'arrêt en arrêt. Il peut aussi modifier son itinéraire pour prendre en compte un détour.

Une version est développée en responsive web design à l'aide de Bootstrap et une autre en web component avec Vue.js. Le scénario de l'application prévoit des usagers différents avec différentes intentions : elle différencie utilisateur régulier et ponctuel. L'utilisateur aura la possibilité de voir de manière globale les arrêts possibles, et de consulter les horaires des lignes.

# Native SWIFT - Anthony Lorosco

## Comment l'exemple a été réalisé

Ma partie a été développée en natif IOS, plus précisément en SWIFT. Ce langage a été récemment lancé par Apple. Celui-ci hérite de problèmes inhérents à sa "jeunesse"<sup>1</sup>. En effet, il a été présenté en 2014 et rendu open source l'année suivante. Nous sommes actuellement à la version 4.

Swift étant open source on peut développer aussi bien sur mac que sur linux, qui sont les plateformes sur lesquelles le compilateur est disponible. Cependant il faut nécessairement disposer d'un mac pour déployer sur un produit tournant sous iOS ainsi qu'un compte développeur<sup>2</sup>.

Pour réaliser l'application je n'ai pas utilisé de bibliothèques externes ni de framework. L'application se base uniquement sur le langage swift ainsi que les bibliothèques natives à celui-ci, notamment UIKit.

Le but de ce développement natif a été de le comparer à une solution hybride (ionic) développée par un camarade. L'accès au capteurs (GPS et boussole) représente donc le principal travail de l'application.

## Accès au GPS

Pour obtenir l'autorisation d'accéder aux capteurs GPS du téléphone, il faut ajouter ces deux lignes dans le fichier Info.plist.

```
<key>NSLocationWhenInUseUsageDescription</key>  
<string>$(PRODUCT_NAME) uses location</string>  
<key>NSLocationAlwaysAndWhenInUseUsageDescription</key>  
<string>$(PRODUCT_NAME) uses location</string>
```

La première ligne permet d'accéder aux capteurs GPS quand l'application est ouverte, la troisième ligne permet d'accéder aux capteurs GPS quand l'application tourne en fond et n'est plus au premier plan.

---

<sup>1</sup> Nombreux tutoriels et snippet d'exemple ne sont pas disponibles dans la dernière version.

<sup>2</sup> Il est possible d'utiliser un compte gratuit pour déployer uniquement sur un nombre réduit de devices.

Pour obtenir la localisation de l'utilisateur, il faut déclarer les lignes suivantes dans la classe qui s'occupe de trouver l'arrêt de bus le plus proche (BusStopController dans notre cas) :

```
// Used to start getting the users location
let locationManager = CLLocationManager()

// If location services is enabled get the users location
if CLLocationManager.locationServicesEnabled() {
    locationManager.delegate = self
    locationManager.desiredAccuracy = kCLLocationAccuracyBest //locaiton accuary

    locationManager.startUpdatingLocation()
}
}
```

Ensuite on obtient la localisation en accédant aux attributs de locationManager.

Pour calculer la distance entre ma position et une position GPS précise, il faut utiliser la fonction .distance fournie par la bibliothèque CoreLocation native à SWIFT.

```
currentLocation.distance(from: templeier)
```

Ceci renvoie la distance en mètres entre la position currentLocation et la position templeier.

## Accès et représentation de la boussole

Pour faire fonctionner la boussole nous n'avons pas besoin de spécifier de nouveaux droits dans le fichier Info.plist car nous accédons aux mêmes capteurs.

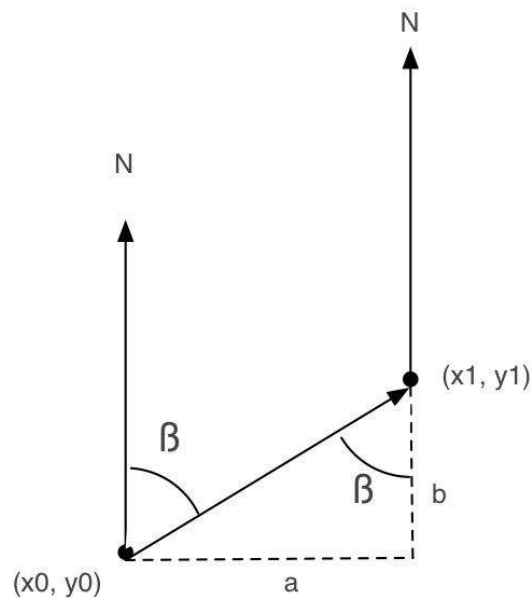
Nous avons besoin d'observer en continu la direction vers laquelle pointe le téléphone :

```
let locationManager: CLLocationManager = {
    $0.requestWhenInUseAuthorization()
    $0.startUpdatingLocation()
    $0.startUpdatingHeading()
    return $0
}(CLLocationManager())
```

Pour que la boussole pointe vers un point précis (par exemple des coordonnées GPS), nous avons besoin d'obtenir la localisation de notre téléphone :

```
func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation])
{
    guard let currentLocation = locations.last else { return }
    lastLocation = currentLocation // store this location somewhere
}
```

Maintenant que nous avons notre localisation, la localisation de la destination et l'orientation du téléphone, nous devons calculer le palier (qui correspond à l'angle qui pointe vers notre destination et donc la direction de la destination). L'image ci-dessous explique cela,  $\beta$  est notre palier.



$$\beta = \text{arc tan} (a , b)$$

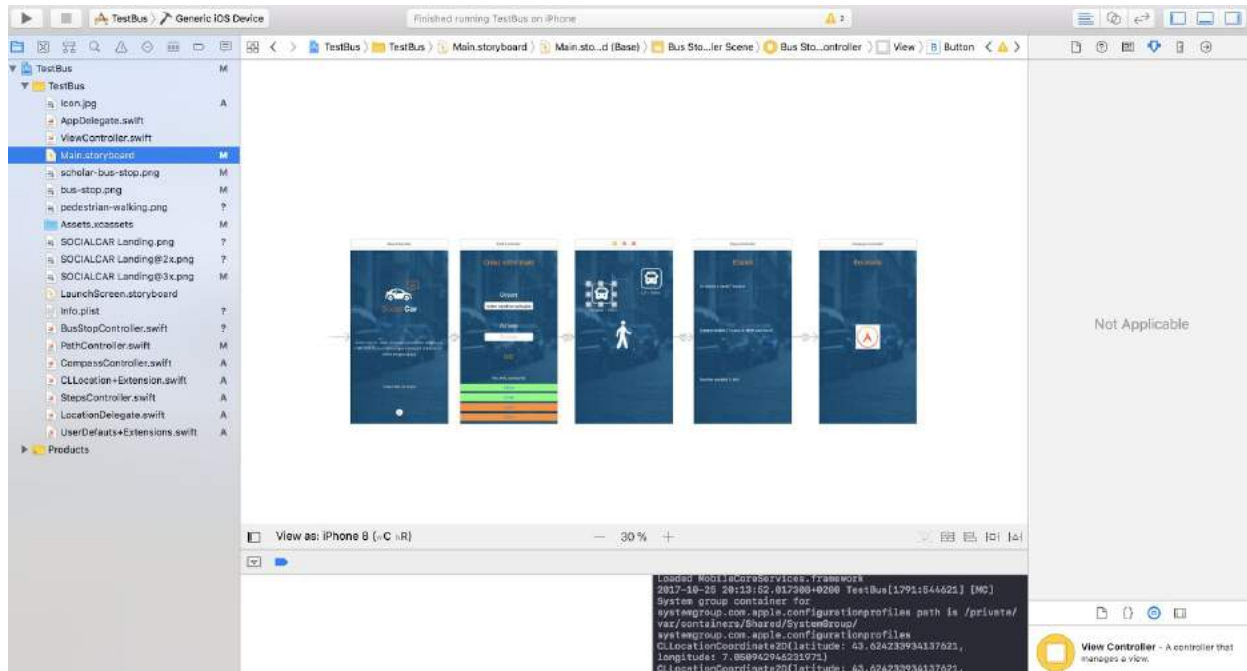
Nous devons donc juste calculer  $\beta$  avec la formule ci dessus. Une fois que celui-ci est calculé, nous orientons l'image dans la bonne direction.

```
UIView.animate(withDuration: 0.5) {
    self.imageView.transform = CGAffineTransform(rotationAngle: latestBearing - latestHeading)
}
```

## Avec quel outil de développement, de tests ?

Les outils de développement que j'ai utilisés sont ceux proposés par Apple sur la plateforme macOS. Cela signifie que j'ai utilisé Xcode comme IDE et mon iPhone comme plateforme pour essayer l'application. La pierre angulaire de ce projet a été l'utilisation du storyboard de Xcode

qui permet de créer des interfaces facilement et rapidement, puis de lier les éléments de l'interface au code pour ajouter le "cerveau" à l'application.

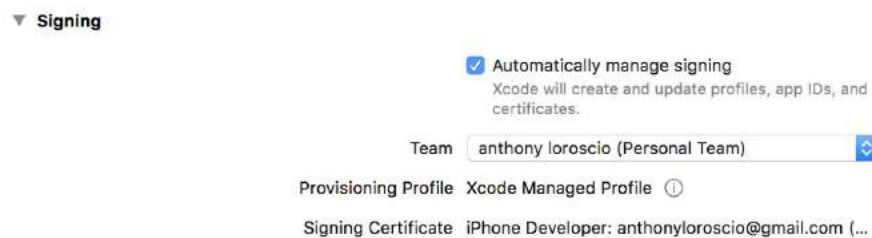


Story board de l'application en cours de développement

On peut voir que chaque écran de l'application est créé ici, avec les boutons et leur couleur ainsi que leur disposition et leur action (représenté par les flèches).

## Comment déploie-t-on et exécute-t-on l'exemple ?

Comme mentionné plus haut, le code est théoriquement compilable sous linux et macOS. Cependant il faut nécessairement un ordinateur tournant sous macOS et un iPhone ainsi qu'un compte développeur. Une fois le compte développeur créé, il faut s'authentifier sous Xcode avec son compte afin de pouvoir signer les applications.



Gestion des signatures

Une fois l'application signée, il faut autoriser l'iPhone à exécuter une application issue d'un développeur non certifié. Pour cela il suffit de suivre les indications que Xcode affichera lors du premier déploiement. Une fois ceci fait, il suffit de lancer l'application en sélectionnant votre device sur le menu déroulant en haut à gauche à côté des boutons play et stop, puis de cliquer sur le bouton play pour déployer.

## Comment teste-t-on les capacités d'adaptations ?

L'adaptation de notre application mobile se situe au niveau utilisateur et dispositif. J'utilise l'application pour me rendre chez un ami qui dispose aussi de celle-ci. Je n'ai pas à saisir l'endroit où je suis car l'application récupère directement mes coordonnées GPS et me propose les arrêts de bus les plus proches. Cela représente une adaptation au dispositif car nous utilisons une donnée issue d'un capteur propre à la plateforme.

Maintenant que mon point de départ est rentré, je peux saisir ma destination en rentrant le lieu que je souhaite ou en cliquant sur le nom d'un de mes amis qui dispose aussi de l'application. Par exemple, je décide d'aller là où adrien se trouve? La destination sera alors automatiquement remplie avec les données GPS correspondantes. Je peux alors choisir entre plusieurs arrêts de bus si il y en a plusieurs proches de moi, leur distance par rapport à moi est affichée. Puis je lance le trajet.

Si lors du trajet je dois effectuer un détour, par exemple passer chez louis, je n'ai qu'à cliquer sur l'icône en bas de l'écran de mon trajet qui me demandera par où je veux passer. L'application va alors utiliser ma localisation actuelle pour calculer le nouvel itinéraire et modifier le trajet en conséquence. J'exploite alors les capacités de la plateforme pour satisfaire un changement de besoin de l'utilisateur ce qui est une adaptation à l'utilisateur.

Concrètement, les écrans qui font appel aux capteurs sont : l'écran qui permet de choisir l'arrêt de bus le plus proche (la distance est calculée en temps réel par rapport à la position actuelle de l'utilisateur) et l'écran de la boussole (qui tire accès lui aussi au capteurs). Les autres écran sont "mockés" afin de représenter le comportement de l'application.



# CrossPlatform avec Ionic 3 - Thomas Monzein

Le but de cette section est de réaliser une application mobile pour un réseau de bus. Cette application doit être développée pour tous les systèmes mobiles soient Android, IOS et Windows Phone. Pour simplifier et réduire les coûts de développement, nous utiliserons une technologie dite CrossPlatform. L'avantage est d'avoir un seul projet pour développer une application sur tous les appareils. Il existe plusieurs technologies CrossPlatform, ici nous traiterons Ionic 3 avec Angular 4. Les langages utilisés sont HTML, CSS et Javascript/Typescript avec Angular 4.

## Environnement de développement

Ce tutorial est réalisé pour une machine sous Windows 8.1, la procédure ne prend pas en compte les spécificités des autres systèmes d'exploitation. Pour les environnements Unix, les commandes **npm** doivent être précédées de **sudo**.

La procédure d'installation est décrite ici :

<http://ionicframework.com/docs/v1/guide/installation.html>

## Node.Js

Téléchargez la dernière version recommandée de Node.js (6.11.5) : <https://nodejs.org/en/>

Exécutez l'installer et suivez les étapes d'installations.

Une fois l'installation terminée, Node.js et npm sont installés. Vous pouvez vérifier que npm est correctement installé en tapant la commande **\$ npm -v** dans une fenêtre de commande.

## Ionic et Cordova

Ouvrez une fenêtre de commande Windows en mode administrateur.

Tapez la commande **\$ npm install -g cordova**

Puis la commande **\$ npm install -g ionic**

Cordova et Ionic sont désormais installés. Vous pouvez vérifier qu'ils sont correctement installés en tapant dans la fenêtre de commande **\$ cordova -v** pour cordova et **\$ ionic -v** pour ionic. Cordova peut alors vous demander de rapporter votre usage, tapez **n**.

## Android Studio (Recommandé)

Il est recommandé d'installer Android Studio si vous voulez lancer vos applications Ionic sur un téléphone Android. Ionic utilise les SDK d'Android Studio pour construire les apk.

La page suivante de Cordova décrit la procédure à suivre :

<http://cordova.apache.org/docs/en/latest/guide/platforms/android/index.html>

Il faut avant tout avoir Java Development Kit (JDK) en version 8 ou supérieur.

Pour télécharger Android Studio : <https://developer.android.com/studio/index.html>

Exécuter l'installer et suivez les étapes d'installations. Une fois installé, ouvrez Android Studio, créer un nouveau projet, compiler et exécuter le sur le simulateur et/ou un smartphone Android connecté. Ionic pourra construire les apk et lancez vos applications.

Soyez assuré que la variable d'environnement JAVA\_HOME pointe bien sur le chemin de votre JDK. Pour cela, ouvrez l'explorateur de fichier, cliquez sur *Ordinateur* puis en haut sur *Propriétés système* et sur *Modifier les paramètres*. Dans la fenêtre qui s'ouvre, rendez-vous sur l'onglet *Paramètres systèmes avancés* puis cliquez sur le bouton *Variables d'environnement*. Dans les *variables système*, cherchez JAVA\_HOME. Si elle n'y est pas, cliquez sur *nouvelle* dans les *variables systèmes*. Ajoutez JAVA\_HOME avec pour valeur le chemin vers le SDK. Vous devrez faire la même manipulation pour ANDROID\_HOME mais avec le chemin vers android\_sdk d'Android Studio. Ensuite, modifier la variable d'environnement PATH, et rajouter à la fin ;%JAVA\_HOME%;%ANDROID\_HOME%;

## Xcode pour IOS (Seulement pour Mac OS)

Pour pouvoir déployer sur les plateformes IOS, il faut Xcode et un Mac.

La page suivante de Cordova décrit la procédure à suivre pour la mise en place de l'environnement de développement IOS :

<http://cordova.apache.org/docs/en/latest/guide/platforms/ios/index.html>

## Créer un premier projet

Avec la fenêtre de commande Windows déplacer vous dans un répertoire de travail. Taper la commande **\$ ionic start todo blank --type ionic1** pour créer votre premier projet Ionic. Ionic vous proposeras de lier le projet au dashboard, tapez **n**. Entrez dans le dossier créer **\$ cd todo**. Tapez **\$ ionic serve**, votre navigateur devrait se lancer avec votre projet ionic.

Vous pouvez ensuite éteindre l'émulation avec CTRL + C dans la ligne de commande suivi de **o**.

## Développer avec Ionic

Le projet créé par Ionic à l'architecture d'un site web. Vous pouvez utiliser n'importe quelle IDE. La compilation et l'exécution sont gérés par Ionic via la fenêtre de commande Windows.

Pour développer avec Visual Studio 2015, ouvrez un site web existant et sélectionner le répertoire racine du projet Ionic. Vous pouvez modifier le contenu des fichiers générés avec Visual Studio.

## Ajouter des pages

Cependant, pour ajouter des pages, utilisez les lignes de commande Ionic comme **\$ ionic generate page nom\_de\_la\_page**. Une fois la page générée, rendez-vous dans `src/app/app.module.ts` et ajoutez pour la nouvelle page créée :

- Un import :  
`import { Classe_de_la_page } from './pages/nom_de_la_page/nom_de_la_page';`
- Dans de `@NgModules`, ajoutez le nom de la classe de votre page aux deux tableaux `declarations` et `entryComponents`.

## Naviguer entre les pages

Une fois vos pages développés, vous voulez vous rendre d'une page à l'autre. Ionic utilise un contrôleur de navigation pour passer entre les pages. L'exemple ci-dessous montre comment passer d'une page à une autre.

Fichier HTML

...

```
<button (click)=avance()>avance</button>
```

```
<button (click)=recule()>recule</button>
```

...

Fichier Typescript

```
Import {PageA} from './a/a';
```

...

```
avance(){ this.navCtrl.push(PageA); } // passer à une nouvelle page
```

```
recule() { this.navCtrl.pop(); } // Passer la page précédente et retirer la page actuelle
```

## Ajouter des modules complémentaires

Pour prendre en charge les capteurs, il faut installer des modules complémentaires. Pour cela, rendez-vous sur la documentation du module à installer et suivez la procédure décrite.

Pour notre application de réseau de bus, nous utiliserons le capteur GPS et la boussole.

Pour les installer tapez les commandes suivantes :

```
$ ionic cordova plugin add cordova-plugin-geolocation --variable
```

```
GEOLOCATION_USAGE_DESCRIPTION="To locate you"
```

```
$ npm install --save @ionic-native/geolocation
```

```
$ ionic cordova plugin add cordova-plugin-device-orientation
```

**\$ npm install --save @ionic-native/device-orientation**

Il faut ensuite les ajouter à chaque page qui les utilise.

```
import { Geolocation } from '@ionic-native/geolocation';
import { DeviceOrientation, DeviceOrientationCompassHeading } from
 '@ionic-native/device-orientation';

...

@Component({
  ...
  providers : [Geolocation, DeviceOrientation]
})

constructor(private geolocation: Geolocation, private deviceOrientation: DeviceOrientation) {}

...
```

Pour la documentation relative à l'utilisation des deux modules :

<https://ionicframework.com/docs/native/geolocation/>

<https://ionicframework.com/docs/native/device-orientation/>

## Exécuter l'application

Pour exécuter l'application, ouvrez une fenêtre de commande Windows et rendez-vous à la racine du projet à exécuter. Tapez **\$ ionic serve** pour lancer l'application sur le navigateur. Le navigateur ne peut pas reproduire certains capteurs propres aux appareils mobiles comme la boussole. Vous pouvez redimensionner la fenêtre du navigateur pour simuler la taille des smartphone ou utiliser le lien <http://localhost:8100/ionic-lab> pour essayer les rendus sur les différents systèmes d'exploitation et une taille mobile.

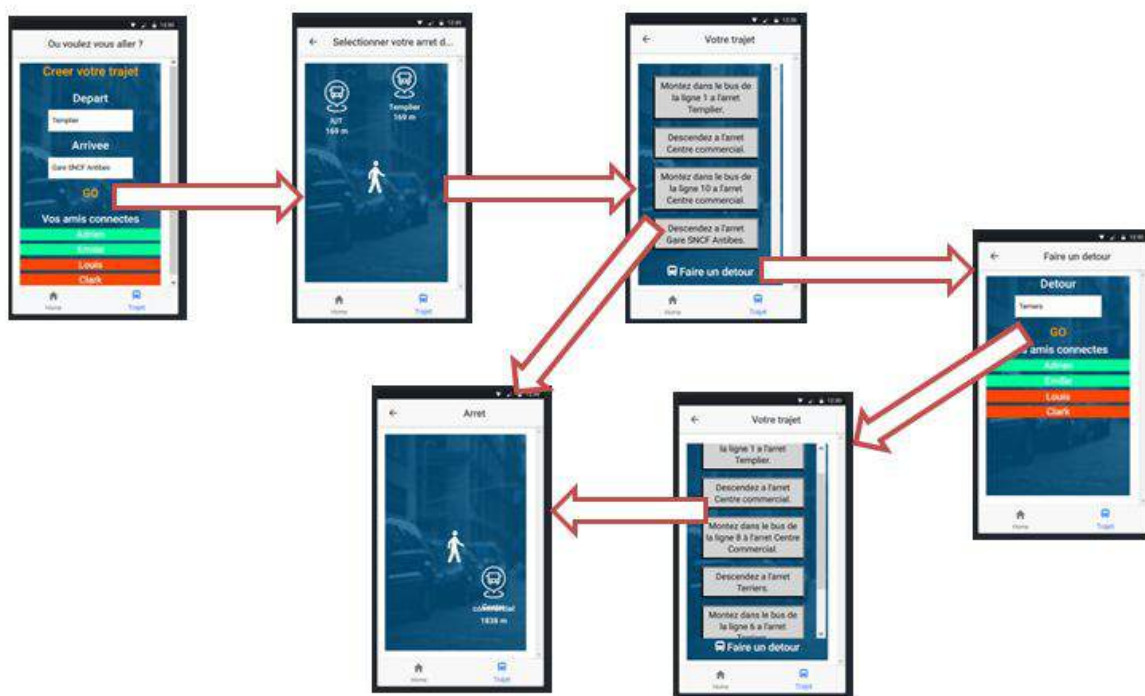
Vous pouvez utiliser un téléphone Android connecté ou l'émulateur Android pour exécuter l'application. Pour cela, tapez la commande **\$ ionic cordova run android**, cette commande construit l'apk puis l'exécute sur un appareil Android connecté, ou si ce n'est pas possible, utilise l'émulateur d'Android Studio. Vous pouvez lancer l'émulateur depuis Android Studio, il sera utilisé lorsque vous exécuterez la commande précédente. Pour le lancer sur un appareil physique, il faut qu'il soit connecté et en mode debug pour les développeurs. Certains Android ont besoin d'étape de configuration ou de logiciel supplémentaire comme certains smartphones de Huawei qui nécessitent l'utilisation de HiSuite.

Si vous voulez uniquement construire l'apk, tapez la commande `$ionic cordova build android`. Vous pourrez l'installer sur les appareils Android en utilisant l'apk généré qui se trouve dans le dossier `platforms/android/build/outputs/apk/`.

## Capacités d'adaptations

Notre application consiste à guider l'utilisateur de son point de départ à son point de départ en utilisant les bus. Pour cela, l'utilisateur peut s'aider d'une boussole qui le guide vers les arrêts proches pour monter dans le bus de son trajet.

### L'application et sa capacités d'adaptation



#### Navigation au sein de l'application

Le premier écran attend de recevoir un lieu de départ et d'arrivée. Le lieu de départ est rempli par la géolocalisation avec l'arrêt le plus proche de l'utilisateur. L'arrivée peut être remplie à partir de l'arrêt le plus proche d'un ami. Il s'agit d'une adaptation aux dispositifs puisque l'on utilise une donnée issue d'un capteur de l'appareil.

La boussole indique les arrêts les plus proches de l'utilisateur ainsi que la distance qui les sépare. Les arrêts se déplacent autour de l'utilisateur, selon l'orientation du téléphone, pour que l'utilisateur puisse se servir de son téléphone pour rejoindre l'arrêt voulu. L'application s'adapte au dispositif qui utilise les données issues des capteurs du téléphone. L'utilisateur peut alors choisir et se rendre à l'arrêt le plus proche de lui, dans la direction de sa destination.

S'il sélectionne un arrêt, les étapes de son trajet à partir de l'arrêt choisis sont affichées. L'utilisateur peut alors cliquer sur une étape pour faire apparaître la boussole avec uniquement l'arrêt de l'étape. Il peut aussi choisir de faire un détour et d'avoir un changement de son trajet. Les nouvelles étapes du trajet bénéficient de la boussole si on clique dessus. Ainsi, l'application s'adapte au changement du besoin de l'utilisateur puisqu'il veut changer son parcours en cours de route.

## Autopsie de la boussole

La boussole utilise le capteur de position GPS et la boussole du téléphone. Il faut abonner la page aux deux capteurs pour qu'à chaque nouvelle valeur renvoyer par le capteur l'affichage soit rafraîchi. L'orientation n'est pas disponible sur navigateur et donc peut causer des erreurs. Le taux de rafraîchissement du GPS est de trois secondes. Ce n'est pas le cas du capteur d'orientation qui par défaut envoie une nouvelle orientation toutes les 100 millisecondes. Nous avons ajouté une option pour recevoir une valeur d'orientation par seconde.

```
constructor(public navCtrl: NavController, private geolocation: Geolocation,
  private deviceOrientation: DeviceOrientation) {
}

ionViewDidEnter() {
  let watch = this.geolocation.watchPosition();
  watch.subscribe((data) => {
    this.refresh();
  });

  var subscription = this.deviceOrientation.watchHeading(ArretPage.options).subscribe(
    (data: DeviceOrientationCompassHeading) => {
      this.refresh();
    },
    (error: any) => {
      console.log(error);
    }
  );
}
```

L'accès aux valeurs des capteurs se fait comme dans le code suivant. Pour le GPS on récupère la longitude et la latitude. Pour la boussole, on récupère la valeur de l'orientation en degré. On effectue une conversion en radian pour pouvoir l'utiliser avec les méthodes de trigonométrie de Math qui utilise des radians.

```
refresh() {
  this.geolocation.getCurrentPosition().then((resp) => {
    this.longitude = resp.coords.longitude;
    this.latitude = resp.coords.latitude;
  });
  this.deviceOrientation.getCurrentHeading().then(
    (data: DeviceOrientationCompassHeading) => {
      this.orientation = Math.PI * data.trueHeading / 180;
    },
    (error: any) => {
      this.orientation = 0;
    }
  );
  this.focus_arret(3);
}
```

```
distance(long: number, lat: number): number {
  let R: number = 6370000 //Rayon de la terre en mètre
  let lat_a = (Math.PI * this.latitude) / 180;
  let lon_a = (Math.PI * this.longitude) / 180;
  let lat_b = (Math.PI * lat) / 180;
  let lon_b = (Math.PI * long) / 180;

  let result: number = R * (Math.PI / 2 - Math.acos(Math.sin(lat_b) * Math.sin(lat_a) +
    Math.cos(lon_b - lon_a) * Math.cos(lat_b) * Math.cos(lat_a)));
  return result;
}

azimut(long: number, lat: number): number {
  let x: number = Math.cos(lat) * Math.sin(this.latitude) - Math.sin(lat) * Math.cos(this.latitude)
    * Math.cos(this.longitude - long);
  let y: number = Math.sin(this.longitude - long) * Math.cos(this.latitude);

  let atan2: number = Math.atan2(x, y);
  return atan2;
}
```

La distance calculée entre l'utilisateur et chaque arrêt permet de connaître les arrêts à afficher. L'azimut permet de savoir où se trouve un arrêt par rapport à l'utilisateur. Il faut cependant

ajouter l'orientation de l'utilisateur modulo deux pi pour tenir compte de l'orientation de l'appareil.

```
var distance = proche.distance(this.longitude, this.latitude);

this.arretLongitude = proche.longitude;
this.arretLatitude = proche.latitude;
this.arretNom = proche.nom;
let azimut: number = proche.azimut(this.longitude, this.latitude);

var dist: string = distance.toFixed(0).toString().toLowerCase();
if (dist == "nan") {
  this.distance = "";
}
else {
  dist = dist + " m";
  this.distance = dist;
  let angle: number = (azimut + this.orientation) % ( 2 * Math.PI);
  let x: number = 35 + Math.cos(angle) * 35;
  let y: number = 35 + Math.sin(angle) * -35;
  let arretBus = document.getElementById("arretBus");
  if (arretBus != null) {
    arretBus.style.top = y + "%";
    arretBus.style.left = x + "%";
    arretBus.style.visibility = "visible";
  }
}
```

L'écran pour choisir l'arrêt de départ et d'arrivé utilise uniquement le capteur GPS pour déterminer l'arrêt le plus proche. Pour notre application, l'arrêt le plus proche est retourné parmi des arrêts en local. Dans une vraie application, ce calcul doit être fait par le serveur qui récupère la position GPS de l'utilisateur. L'orientation quant à elle a du sens d'être calculée du côté du client.

# Web component avec Vue.js

## Outil de développement

Les outils utilisés sont Node.js, Vue.js ainsi que Atom comme IDE.

### Installer le projet

L'installation peut se faire en ligne de commande avec les instructions suivantes :  
(source : <https://vuejs.org/v2/guide/installation.html>)

```
$ npm install vue
```

### Générer le projet

```
# install vue-cli
$ npm install --global vue-cli
# create a new project using the "webpack" template
$ vue init webpack my-project
# install dependencies and go!
$ cd my-project
$ npm install
$ npm run dev
```

Le projet créé aura ainsi l'arborescence suivante :

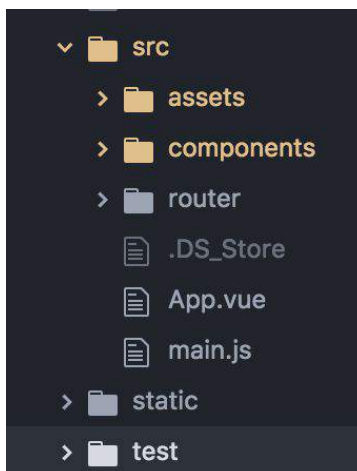


Figure C1 : Arborescence du dossier source

Un dossier source composé d'**assets** (les ressources externes), de **components** (les fichiers .vue que nous allons créer), et de **router** afin d'aiguiller la navigation de l'application.



Il y a de plus un dossier test, qui comporte les tests unitaires. L'outil de test par défaut avec ce générateur est Karma.js

### Exécuter le projet

```
$ npm run dev
```

Le projet sera ainsi accessible au lien suivant : <http://localhost:8080>

### Déployer le projet

```
$ npm run build
```

Cette commande va générer un projet dans le dossier dist/

### Description d'un composant

Un composant est un fichier .vue composé de trois parties simples

- Une partie html

```
<template>
// code html du composant
</template>
```

- Une partie js

```
<script>

export default {
  components: {
    // déclare composants utilisés dans ce composant
  },
  data () {
    return {
      // déclare les données du scope
    }
  },
  methods: {
    // déclare les méthodes liées au composant
  }
}
// d'autres paramètres peuvent être complétés, il faut se référer à la doc
</script>
```

- Une partie css

```
<style scoped>  
// code css  
</style>
```

Le “*scope*” dans la balise signifie que ce css ne s’applique qu’au composant du fichier

## Réalisation

Le projet est divisé en trois vues différentes. Nous supposons que l’utilisateur y accèdera seulement avec un navigateur sur ordinateur.



The image shows a navigation menu for a website. It features the text "Let's Go" in a bold, dark blue font, followed by three menu items: "Accueil", "Navigation", and "Horaires", all in a green font. The menu is set against a light gray background with a subtle shadow effect.

Figure C2 : Menu de navigation

## L’Accueil

Cette page est utilisée lorsqu’un utilisateur souhaite rechercher les trajets possibles entre deux arrêts de bus. Il a la possibilité d’entrer un arrêt de départ et d’arriver, ainsi qu’une heure de départ afin de centrer la recherche.

Welcome to Let's Go App

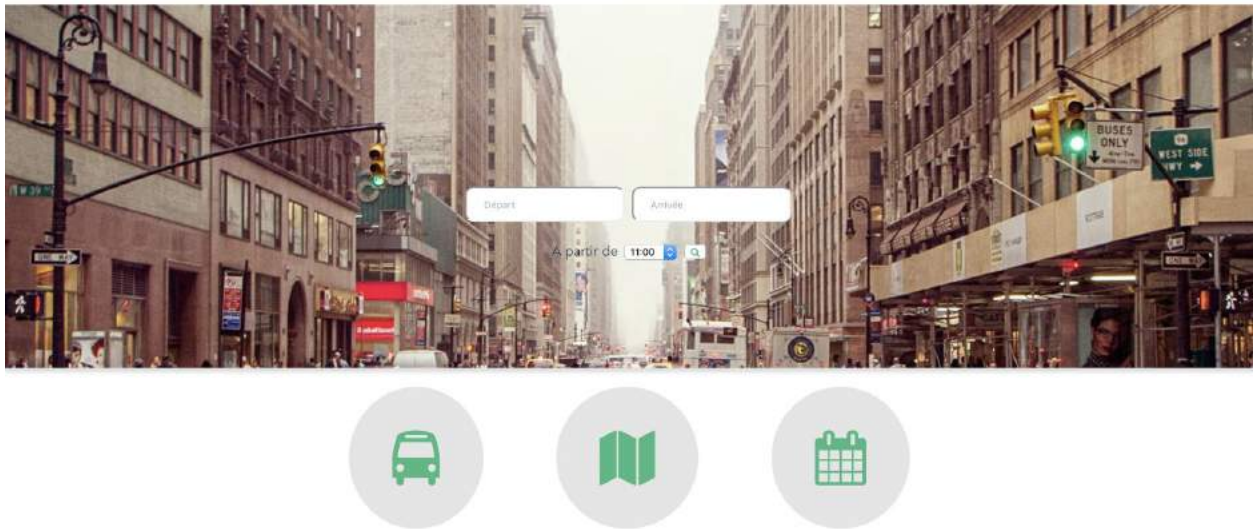


Figure C3 : Page Accueil

## La Navigation

Cette page est utilisée pour afficher les résultats de la recherche qu'à entrer précédemment l'utilisateur dans la page d'**Accueil**. Il affiche à gauche la liste des arrêts possibles, au milieu les arrêts indiqués, et à droite les arrêts et les heures de passage en lien avec la ligne de bus sélectionnée.

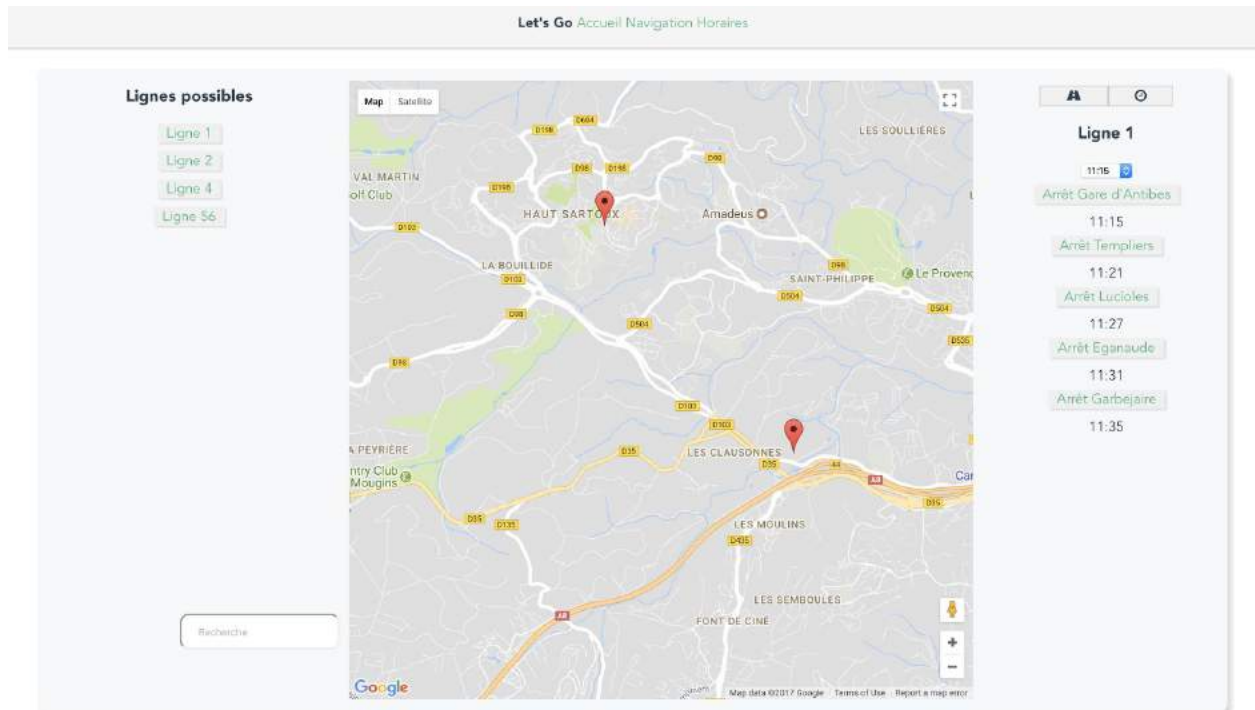


Figure C4 : Page Navigation

## Les Horaires

Cette page concerne différent type d'utilisateur. Les personnes qui utilisent ponctuellement l'application et donc ont accès à toutes lignes; et des utilisateurs qui utilisent l'application régulière et possèdent des lignes et horaires favoris.

Ligne	Horaire	Départ	Arrivée	Détails
Polybus 100	07:25	Amphores	Dugommier	Détails
Polybus 100	07:45	Amphores	Dugommier	Détails
Polybus 100	08:25	Amphores	Dugommier	Détails
Polybus 100	08:45	Amphores	Dugommier	Détails
Polybus 100	09:25	Amphores	Dugommier	Détails
Polybus 100	10:35	Amphores	Dugommier	Détails
Polybus 100	11:35	Amphores	Dugommier	Détails
Polybus 1	07:38	Echanges	Templiers	Détails
Polybus 1	07:58	Echanges	Templiers	Détails
Polybus 1	08:18	Echanges	Templiers	Détails
Polybus 1	08:58	Echanges	Templiers	Détails
Polybus 1	09:18	Echanges	Templiers	Détails
Polybus 1	09:38	Echanges	Templiers	Détails
Polybus 1	09:58	Echanges	Templiers	Détails
Polybus 2	15:56	Amphores	Templiers	Détails
Polybus 3	14:56	Audiberti	Eganaude	Détails
Polybus 4	14:32	Dugommier	Eganaude	Détails
Ligne 5	09:56	Amphores	VSA	Détails
Ligne 6	08:37	Audiberti	Skema	Détails
Ligne 7	16:43	Amphores	VSA	Détails
Ligne 8	16:49	Amphores	Echanges	Détails
Ligne 9	17:54	Combes	VSA	Détails
Ligne Noctibus	00:10	Dugommier	VSA	Détails

Figure C5 : Page Horaires

## Adaptation

### Composants réutilisables

Nous avons ainsi créé divers composants afin de les réutiliser dans des contextes différents. Chaque composant sera décrit avec leur contexte d'utilisation ainsi que le code qui les implémente.

- L'onglet - **Tabs**
  - Contexte de **Navigation** : Il va permettre à l'utilisateur de passer entre les arrêts et les horaires

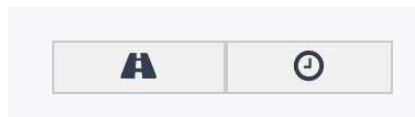


Figure C6 : Tabs appliquée à la page de Navigation

```
<v-tabs>
  <v-tab flink='/#/nav#' name="" ficon="road">
    <!--CONTENT-->
  </v-tab>
  <v-tab flink='/#/nav#' ficon="clock-o">
    <!--CONTENT-->
  </v-tab>
```

</v-tabs>

- Consulter les **Horaires** : Il va permettre à l'utilisateur de choisir les types d'horaires qui lui correspond (toutes, localisée, favorites)

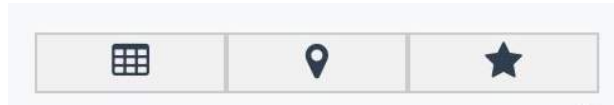
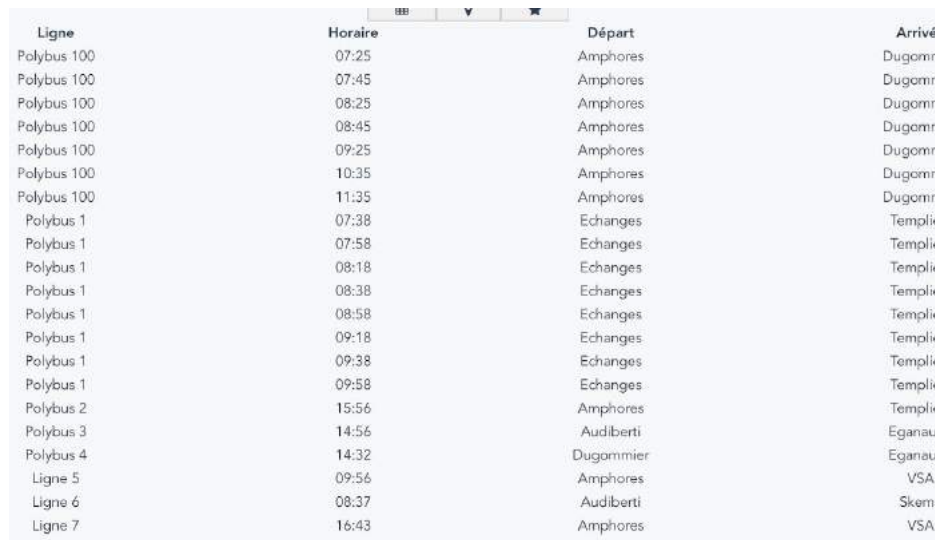


Figure C7 : Tabs appliquée à la page de Horaires

```
<v-tabs>
  <v-tab flink='/#/schedule#' ficon="table">
    <!--CONTENT-->
  </v-tab>
  <v-tab flink='/#/schedule#' ficon="map-marker">
    <!--CONTENT-->
  </v-tab>
  <v-tab flink='/#/schedule#' ficon="star">
    <!--CONTENT-->
  </v-tab>
</v-tabs>
```

- Le tableau de données - **Datatable**

- **Horaires** : Le tableau de données affichera des informations suivant différents contextes
  - Contexte qui s'adapte à l'utilisateur : "Je suis un utilisateur ponctuel, je souhaite consulter les horaires globales"



Ligne	Horaire	Départ	Arrivée
Polybus 100	07:25	Amphores	Dugommi
Polybus 100	07:45	Amphores	Dugommi
Polybus 100	08:25	Amphores	Dugommi
Polybus 100	08:45	Amphores	Dugommi
Polybus 100	09:25	Amphores	Dugommi
Polybus 100	10:35	Amphores	Dugommi
Polybus 100	11:35	Amphores	Dugommi
Polybus 1	07:38	Echanges	Templien
Polybus 1	07:58	Echanges	Templien
Polybus 1	08:18	Echanges	Templien
Polybus 1	08:38	Echanges	Templien
Polybus 1	08:58	Echanges	Templien
Polybus 1	09:18	Echanges	Templien
Polybus 1	09:38	Echanges	Templien
Polybus 1	09:58	Echanges	Templien
Polybus 2	15:56	Amphores	Templien
Polybus 3	14:56	Audiberti	Eganaud
Polybus 4	14:32	Dugommier	Eganaud
Ligne 5	09:56	Amphores	VSA
Ligne 6	08:37	Audiberti	Skema
Ligne 7	16:43	Amphores	VSA

Figure C8 : Datatable appliquée à un utilisateur ponctuel

```
<mydatatable linesJson='all' ></mydatatable>
```

- Contexte qui s'adapte à l'utilisateur : "Je suis un utilisateur régulier, je souhaite consulter mes horaires favoris"

Ligne	Horaire	Départ	Arrivée
Polybus 2	15:56	Amphores	Templiers
Polybus 3	14:56	Audiberti	Eganaude

Figure C9 : Datatable appliquée à un utilisateur régulier

```
<mydatatable linesJson='favorite' ></mydatatable>
```

- Contexte qui s'adapte à l'environnement : "Je suis au bureau et je souhaite consulter les lignes qui passent autour de moi"

Ligne	Horaire	Départ	Arrivée
Polybus 100	07:25	Amphores	Dugommier
Polybus 100	07:45	Amphores	Dugommier
Polybus 100	08:25	Amphores	Dugommier
Polybus 100	08:45	Amphores	Dugommier
Polybus 100	09:25	Amphores	Dugommier
Polybus 100	10:35	Amphores	Dugommier
Polybus 100	11:35	Amphores	Dugommier
Polybus 2	15:56	Amphores	Templiers
Ligne 5	09:56	Amphores	VSA
Ligne 7	16:43	Amphores	VSA
Ligne 8	16:49	Amphores	Echanges

Figure C10 : Datatable appliquée à un utilisateur au travail

```
<mydatatable linesJson='around' ></mydatatable>
```

Code du composant :

```
<template>
<div>

<table width="100%">
  <thead>
    <tr>
      <th>Ligne</th>
      <th>Horaire</th>
      <th>Départ</th>
      <th>Arrivée</th>
      <th>Détails</th>
      <!-- <th>CSS grade</th> -->
    </tr>
  </thead>
  <tbody>
    <tr class="odd gradeX" v-for="line in loadJson(linesJson)">
      <td>{{ line.name }}</td>
      <td>{{ line.time }}</td>
      <td>{{ line.departure }}</td>
      <td>{{ line.arrival }}</td>
      <td>
        <button id="show-modal" @click="showModal = true; updateLine(line)">Détails</button>
      </td>
    </tr>
  </tbody>
</table>
```

```

<mymodal v-if="showModal" @close="showModal = false">
  <h3 slot="header">{{headerModal}}</h3>
  <div slot="body">{{bodyModal}}</div>
</mymodal>
</div>
</template>

<script>
import allLines from '../assets/all-line.json'
import aroundLines from '../assets/around-line.json'
import favoriteLines from '../assets/favorite-line.json'

import Modal from './modal'

export default {
  props: ['linesJson'],
  components: {
    'mymodal': Modal
  },
  data: function () {
    return {
      allLines: allLines,
      aroundLines: aroundLines,
      favoriteLines: favoriteLines,
      showModal: false,
      headerModal: '',
      bodyModal: ''
    }
  },
  methods: {
    loadJson (name) {
      switch (name) {
        case 'all':
          return allLines
        case 'around':
          return aroundLines
        case 'favorite':
          return favoriteLines
        default:
          return allLines
      }
    },
    updateLine (line) {
      this.headerModal = line.name
      this.bodyModal = ''
      for (var i = 0, len = line.stop.length - 1; i < len; i++) {
        this.bodyModal = this.bodyModal + line.stop[i] + ' - '
      }
      this.bodyModal += line.arrival
    }
  }
}

```



```

    }
  }
</script>

```

- Message de détails - Modal

- Dans le cadre de la **navigation**, le modal sert à afficher les détails des horaires d'un trajet



Figure C11 : Modal appliqué à la Navigation

```

<myLi v-for="time in timesSchedule"
v-on:click.native="updateSelected(time.departureTime); showModal = true;
updateModal(time)" :key="time.id" :mylink="prefix_link + time.departureTime"
:text="time.departureTime + ' - ' + time.arrivalTime"></myLi>

```

```

<mymodal v-if="showModal" @close="showModal = false">
<h3 slot="header">{{headerModal}}</h3>
<div slot="body">{{bodyModal}}</div>
</mymodal>

```

- Dans le cadre de la consultation des **horaires**, elle va permettre d'afficher les détails des arrêts d'un trajet



Figure C12 : Modal appliqué aux Horaires

```

<button id="show-modal" @click="showModal = true; updateLine(line)">Détails</button>

```

```

<mymodal v-if="showModal" @close="showModal = false">
<h3 slot="header">{{headerModal}}</h3>
<div slot="body">{{bodyModal}}</div>
</mymodal>

```

Code du composant :

```

<template type="text/x-template" id="modal-template">
<transition name="modal">

```

```

<div class="modal-mask">
  <div class="modal-wrapper">
    <div class="modal-container">

      <div class="modal-header">
        <slot name="header">
          <!-- {{headerModal}} -->
        </slot>
      </div>

      <div class="modal-body">
        <slot name="body">
          <!-- {{bodyModal}} -->
        </slot>
      </div>

      <div class="modal-footer">
        <slot name="footer">
          <!-- {{footerModal}} -->
          <button class="modal-default-button" @click="$emit('close')">
            OK
          </button>
        </slot>
      </div>
    </div>
  </div>
</transition>
</template>

<script>

export default {
  components: {
  },
  data () {
    return {
    }
  },
  methods: {
  }
}

</script>

<style scoped>
.modal-mask {
  position: fixed;
  z-index: 9998;
  top: 0;

```

```

left: 0;
width: 100%;
height: 100%;
background-color: rgba(0, 0, 0, .5);
display: table;
transition: opacity .3s ease;
}

.modal-wrapper {
display: table-cell;
vertical-align: middle;
}

.modal-container {
width: 300px;
margin: 0px auto;
padding: 20px 30px;
background-color: #fff;
border-radius: 2px;
box-shadow: 0 2px 8px rgba(0, 0, 0, .33);
transition: all .3s ease;
font-family: Helvetica, Arial, sans-serif;
}

.modal-header h3 {
margin-top: 0;
color: #42b983;
}

.modal-body {
margin: 20px 0;
}

.modal-default-button {
float: right;
}

.modal-enter {
opacity: 0;
}

.modal-leave-active {
opacity: 0;
}

.modal-enter .modal-container,
.modal-leave-active .modal-container {
-webkit-transform: scale(1.1);
transform: scale(1.1);
}

```

</style>

- Champ de text - **Input**

- **Accueil** : Il servira à recueillir les noms des différents arrêts afin de lancer la recherche

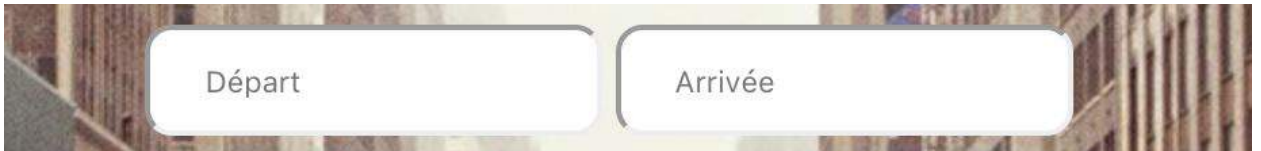


Figure C13 : Input appliqué à l'Accueil

```
<stop nameStop="Départ"></stop>
<stop nameStop="Arrivée"></stop>
```

- **Navigation** : Il sera utilisé pour filtrer les lignes disponibles afin d'effectuer une recherche rapide dans la liste.



Figure C14 : Input appliqué à la Navigation

```
<stop class="onRight" nameStop="Recherche"></stop>
```

- **Horaire** : Il sera utilisé pour filtrer les lignes disponibles afin d'effectuer une recherche rapide dans le tableau.

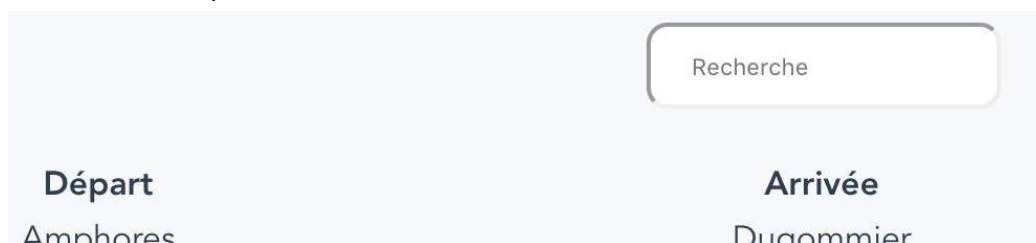


Figure C15 : Input appliqué aux Horaires

```
<stop nameStop="Recherche"></stop>
```

Code du composant :

```
<template>
  <input type="text" :placeholder="nameStop"/>
</template>
```

```
<script>
```

```
export default {
  props: ['nameStop'],
  computed: {
```

```
        usersToRender () {  
            // return this.userList.filter(...);  
        }  
    }  
}  
</script>  
  
<style scoped>  
input[type=text] {  
    /*width: 100%;*/  
    padding: 12px 20px;  
    /*margin: 8px 0;*/  
    box-sizing: border-box;  
    border-radius: 10px;  
  
}  
</style>
```

## Responsive Web Design avec Bootstrap

Cette partie traite d'une technologie de Responsive Web Design.

Pour réaliser ce mini projet, nous avons utilisé Bootstrap (v3.3.7) et un peu de JQuery couplé avec l'IDE WebStorm. Pour le rendu sur smartphone nous avons utilisais l'outil de développement mis à disposition par Google Chrome.

Bootstrap est un framework web permettant de mettre en page assez facilement des sites internet. L'avantage de ce framework est qu'après une rapide prise en main vous pouvez créer des pages Web responsive c'est à dire qui s'adapte sur différent type de périphérique.

### Installation

Il n'y a pas d'installation à proprement parler. Vous allez télécharger les fichiers dont à besoin le site web pour utiliser bootstrap.

Pour obtenir ces fichier vous avez plusieurs méthodes.

- Les télécharger directement depuis le site officiel
- L'installer via Bower si vous utilisez node
- L'installer via npm
- L'installer via composer (conseillé si projet php)

Une dernière solution consiste à ne pas télécharger les sources, mais directement mettre dans votre page web les CDN (content delivery network) de bootstrap qui seront « injectés » lors de la lecture de la page web. À noter que cette dernière solution ne permet pas de travailler hors ligne.

Nous avons opté pour la première solution. Une fois les fichiers téléchargés nous pouvons créer notre premier fichier HTML avec le template de base fourni sur le site :

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
    <title>Bootstrap 101 Template</title>
    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">
    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries --> <!-- WARNING: Respond.js doesn't
work if you view the page via file:// --> <!--[if lt IE 9]> <script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>
<script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script> <![endif]-->
  </head>
  <body>
    <h1>Hello, world!</h1>
    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script> <!-- Include all compiled plugins (below), or
include individual files as needed -->
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

Un des avantages avec bootstrap est qu'il est fourni avec une documentation très précise et énormément d'exemple et de template pour bien démarrer. De plus, il y a une forte communauté derrière ce framework et on ne reste jamais bloqué bien longtemps sur une problématique en cherchant sur internet.

Sans rentrer dans les détails bootstrap s'appuie sur deux principes :

- La division des pages internet en 12 colonnes
- L'utilisation des media-query pour gérer le placement de ces 12 colonnes

Les media-query vont permettre d'effectuer certaines actions selon la taille de l'affichage.

## Réalisation

### La structure

Le site internet est composé de 3 pages l'une permettant de faire une recherche de trajet, une autre montrant le résultat de cette recherche et une dernière permet de consulter la table des horaires des bus en fonction de la ligne.

Les 3 pages ont un comportement différent selon la taille de l'écran. Globalement chaque page à 2 aspects selon si elle est visualisée sur un écran d'ordinateur ou sur un téléphone.

## Les fondations

Pour démarrer ce projet, je suis partie d'un template de base proposée par bootstrap. Il comporte un menu et une page principale. Le css de bootstrap a déjà rendu le menu responsive en mettant des media-query sur la balise HTML <nav>. Ainsi si la largeur de l'écran est < 768 px le menu est iconisé quand on peut le voir dans l'exemple ci-dessous.

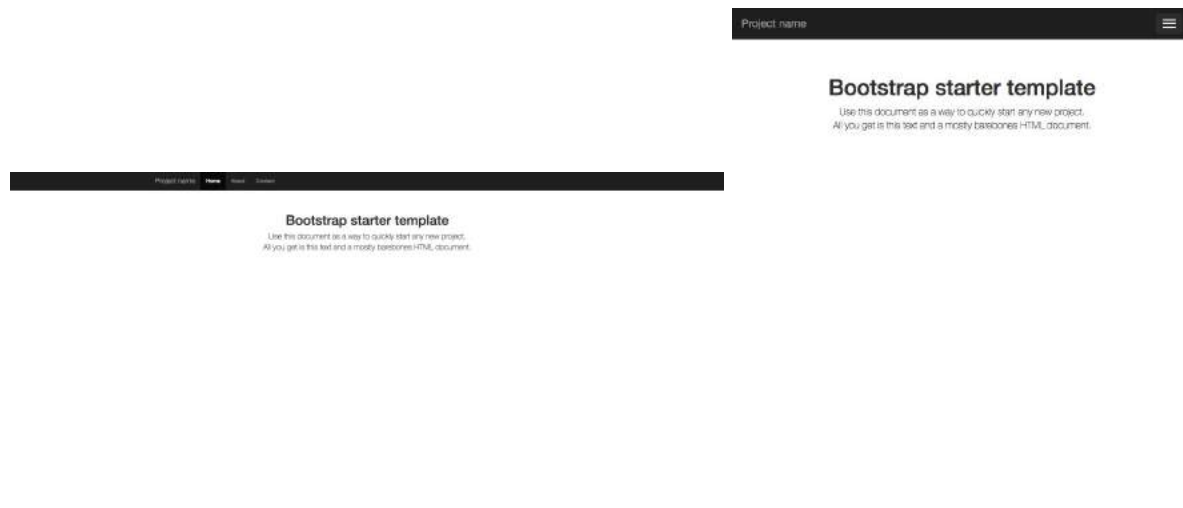


Figure 1 - Template basique sur PC et sur mobile

## Page de recherche

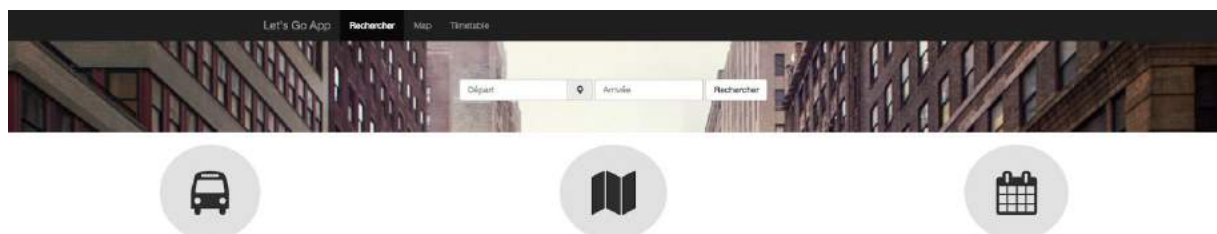


Figure 2 - Page de recherche sur écran d'ordinateur

La page de recherche est composée de 2 champs texte et d'un bouton. Les 3 cercles en bas n'ont qu'un rôle esthétique. On peut noter également un petit bouton à côté du champ texte départ.

## Adaptation au contexte d'usage

Une première adaptation qui a été faite sur cette page fut de permettre à l'utilisateur visualisant la page sur son smartphone (donc écran plus petit) de ne pas scroller horizontalement pour voir les champs de texte. De plus, il fallait que l'image en arrière-plan s'adapte aussi à la taille de l'écran.

Code du formulaire :

```
<form class="form-inline">
  <div class="input-group">
    <input type="text" class="form-control" id="dep" placeholder="Départ"/>
    <span class="input-group-addon" id="position">
      <span class="glyphicon glyphicon-map-marker"></span>
    </span>
  </div>
  <input type="text" class="form-control" id="pwd" placeholder="Arrivée">
  <button type="submit" class="btn btn-default">Rechercher</button>
</form>
```

Par défaut lorsque l'on crée un formulaire avec bootstrap celui-ci n'est pas en une seule ligne. Pour le mettre en une seule ligne nous rajoutons la class form-inline à notre balise form. Cette classe ne va être active que si la taille de l'écran est supérieure à 768 px ainsi quand la taille sera inférieure le formulaire s'affichera bien sur deux lignes. Si nous regardons dans le code source de bootstrap nous pouvons voir la règle qui permet de faire ça :

```
@media (min-width: 768px)
.form-inline .form-control {
  display: inline-block;
  width: auto;
  vertical-align: middle;
}
```

La classe input-group quant à elle permet d'ajouter en plus de l'input une petite icône à côté qui permet parfois d'illustrer le champ de texte. Bootstrap utilise énormément les images vectorielles pour ses icônes. Ces images ne sont pas faites de pixel, mais d'une suite de vecteur permettant d'avoir toujours une qualité optimale quelque soit la taille de l'écran, car l'image s'apparente à du texte. FontAwesome est un framework CSS très connu qui va de pair avec bootstrap et lui permet d'ajouter à sa librairie un certain nombre d'icônes vectorielles.

Pour que l'image d'arrière plan soit responsive nous avons utilisé une propriété de background-size dans mon CSS qui se nomme cover.



## cover

« Un mot-clé dont le comportement est opposé à celui de contain. L'image est redimensionnée pour être aussi grande que possible et pour conserver ses proportions. L'image couvre toute la largeur ou la hauteur du conteneur et les parties qui dépassent sont rognées si les proportions du conteneur sont différentes (il n'y a aucun espace libre sur lequel on verrait la couleur d'arrière-plan). » - [Mozilla](#)

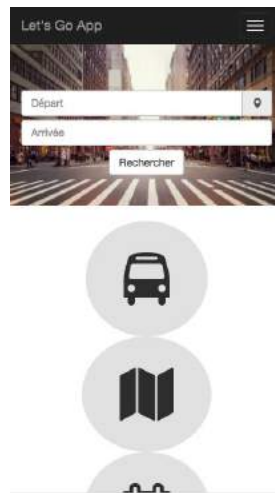


Figure 3 – Rendu mobile

## Adaptation à l'environnement

Étant donné que nous travaillons avec des utilisateurs qui vont rechercher des trajets potentiellement de leurs domiciles ou de là où ils sont il nous a semblé pertinent d'ajouter une fonctionnalité permettant de géolocaliser la personne via son navigateur internet. C'est le rôle du petit bouton à côté de départs.

Cela se fait en JavaScript avec le code suivant :

```
$(function () {
  $('#position').click(function () {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(showPosition);
    } else {
      x.innerHTML = "Geolocation is not supported by this browser.";
    }
  });
});

var x = document.getElementById("dep");
function showPosition(position) {
  document.getElementById("dep").value = position.coords.latitude + "," + position.coords.longitude;
}
```

Une première fonction va vérifier si votre navigateur est compatible et une seconde fonction va traiter les données de géolocalisation pour pouvoir les afficher.

À noter qu'il faut avoir autorisé le site à vous géolocaliser via le navigateur.

Cette fonctionnalité de géolocalisation est compatible avec la plupart des navigateurs sur le marché que ça soit sur PC ou sur Mobile

API					
Geolocation	5.0 - 49.0 (http) 50.0 (https)	9.0	3.5	5.0	16.0

**Note:** As of Chrome 50, the Geolocation API will only work on secure contexts such as HTTPS. If your site is hosted on a non-secure origin (such as HTTP) the requests to get the users location will no longer function.

Figure 4 – Liste navigateur compatible géolocalisation

## Page de la carte

Cette page comporte 3 blocs séparés. Un qui permet de sélectionner la ligne que l'on veut voir, un autre qui permet de voir la carte selon la ligne sélectionnée et un troisième permettant de voir les heures en fonction de la ligne sélectionnée et des arrêts.

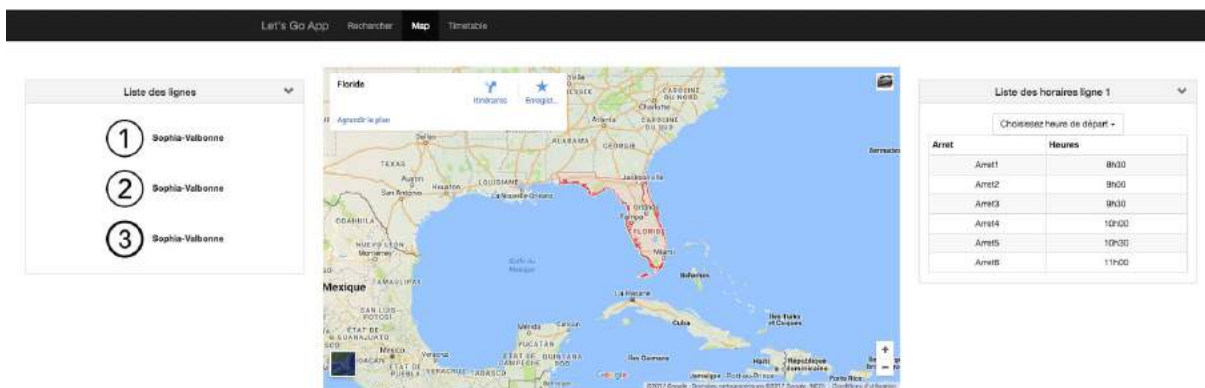


Figure 5 – Page de recherche container-fluid

Cette page a une première particularité par rapport au template de base. Nous avons utilisé la classe container-fluid qui permet au contenu du site de prendre l'entièreté de la largeur dont il dispose. Si nous mettons un container classique, le contenu va se limiter aux 1200 px même si l'écran fait plus. Pour un confort de lecture dans le cas de cette page il vaut mieux occuper tout l'espace de l'écran pour mieux visualiser la carte.



Figure 6 – Page de recherche container classique

On utilise ici une propriété basique de bootstrap avec les colonnes en disant que les deux éléments sur le coté ont chacun une taille de 3 tandis que la carte à une taille de 6. Lors du rétrécissement de l'écran, les 3 éléments se mettront automatiquement les uns à la suite des autres à partir de 1024px.

Cette page utilise l'attribut panel de bootstrap. Un panel est une boîte entourée de bordure que l'on peut personnaliser. Pour le personnaliser nous avons d'abord ajouté une classe panel-group permettant de scinder notre panel en deux parties le heading et le body puis après cela nous avons donné un titre à notre heading et remplis le corps du body. Enfin nous avons couplé tout ceci avec la propriété panel-collapse qui permet de fermer et d'ouvrir le corps d'un panel sans ajouter le moindre code de JavaScript.

```

<div class="panel-group">
  <div class="panel panel-default">
    <div class="panel-heading">
      <h4 class="panel-title">
        <a class="accordion-toggle" data-toggle="collapse" data-parent="#accordion" id="titleHoraire" href="#collapseTwo">Liste des
        horaires ligne 1</a>
      </h4>
    </div>
    <div id="collapseTwo" class="panel-collapse collapse in">
      <div class="panel-body">
        <div class="dropdown">
          /** Le contenu du body **/
        </div>
      </div>
    </div>
  </div>
</div>

```

Nous avons ici une balise <a> qui se trouve dans le header du panel qui va lui indiquer que lorsqu'on clique dessus on doit fermer la div qui à l'id collapseTwo. Cette div à une propriété panel-collapse + collapse qui va lui permettre de communiquer avec le js de bootstrap afin de lancer l'animation de fermeture. L'attribut in permet de dire que le panel est ouvert par défaut.

Ceci est une fonctionnalité très pratique de bootstrap permettant de faire des animations simples qui ont un intérêt pour l'utilisateur.

### Adaptation au contexte d'usage

Nous avons vu précédemment que bootstrap permettait de développer des animations simples sans faire de JavaScript cependant dans notre scénario il se trouve quand l'état si l'utilisateur ouvre l'application sur son smartphone il voit cet affichage.



Figure 7 – Affichage smartphone avant optimisation

Comme vous pouvez le voir il n'y a pas tous les éléments qui sont affichés ceci est dû au fait qu'il faut scroller vers le bas. Une bonne solution pour pallier à ce problème tout en utilisant ce que nous avons déjà développé est que lorsqu'on est sur smartphone les panels soient fermés à l'ouverture de l'application.

Pour cela nous devons écrire ces 2 lignes en JQuery qui dise qu'au-dessus de 1024 px les corps des 2 panels auront l'attribut in permettant de les montrer ouvert en dessous de 1024 ils seront donc fermés.

```
$('#collapseOne').toggleClass('in', $(window).width() > 1024);  
$('#collapseTwo').toggleClass('in', $(window).width() > 1024);
```



Figure 8 – Affichage smartphone après optimisation

Dans le panel listant les horaires on peut remarquer un sélecteur permettant de choisir les heures. L'implémentation par défaut de ce sélecteur faisait que quand on l'ouvrait celui-ci caché des informations qui aurait pu s'avérer utile pour la sélection. Ceci est dû au fait que la partie qui s'étend est en position absolue en float. Pour que l'utilisateur puisse voir les informations de manière correcte nous allons mettre celui-ci en position static sans le float ainsi cela aura pour effet de pousser le contenu en dessous du sélecteur lors de son ouverture.



Figure 9 – Affichage heure avant/après optimisation

## Page des horaires

Cette page comporte deux parties une première où l'on sélectionne une ligne et une seconde qui affiche la fiche horaire de cette ligne.

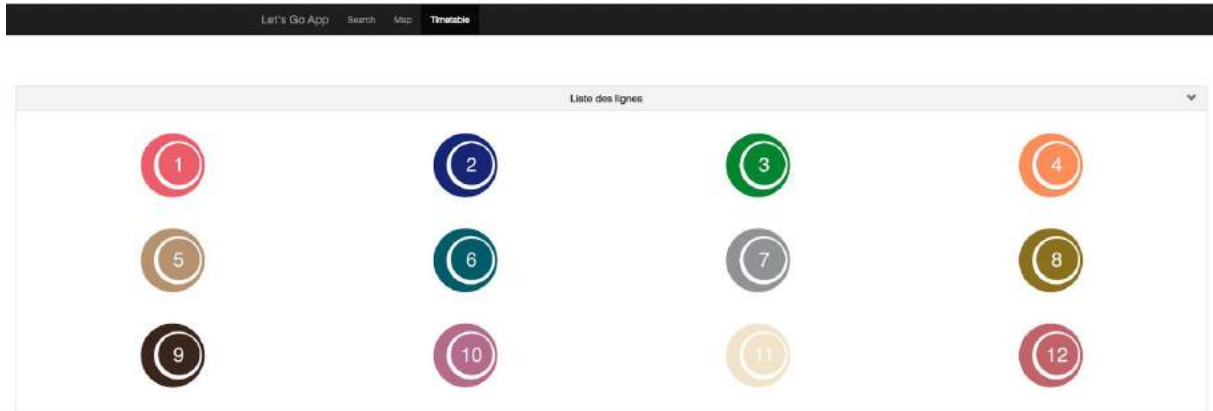


Figure 10 – Première affichage des lignes

Les lignes de bus ont été placées toujours grâce aux systèmes de colonne de bootstrap. Ici nous avons mis ces propriétés à chaque élément.

```
<div class="col-md-3 col-sm-4 col-xs-6 ligne-picto"><i class="icon-ligne" style="color: #EE6A76;"><span class="code-ligne un">1</span></i></div>
```

- col-md-3 veut dire que chaque élément prend 3 colonnes sur 12 (4 éléments par ligne) pour des tailles d'écran  $\geq 992$  px (valeur minimum de la classe md).
- Entre 992 px et 768 px on est en col-sm-4 donc 3 éléments par ligne
- $< 768$  px on est en col-xs-6 donc 2 éléments par ligne

On notera également sur cette partie que chaque forme est une icône vectorielle qui a comme avantage de pouvoir être mise dans n'importe quelle couleur.

Ici pour éviter d'appeler une nouvelle page au clic sur un élément et étant donné que ce n'est pas inclus dans bootstrap j'ai du ajouter du JQuery pour gérer la transition avec l'écran d'après.

Pour cela au click d'un élément j'enlève une propriété display : none que j'avais mise aux deux éléments que je veux afficher et j'utilise la méthode hide de jquery pour cacher la div contenant la liste des lignes de bus.

```

var myDiv = document.querySelector("#collapseOne")
var list = myDiv.querySelectorAll("div");

Array.prototype.slice.call(list).forEach(function(listItem){
  listItem.addEventListener('click', function(e){
    $('#searchButton').removeClass("hidden");
    $('.heures').removeClass("hidden");
    $(".lignes").hide("slow");
  });
});

```

Let's Go App Search Map Timetable

Chercher les horaires d'une autre ligne

Liste des lignes

	Lundi-Vendredi	Samedi	Dimanche
Arrêt1	05:08 05:38 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58
Arrêt2	05:08 05:38 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58
Arrêt3	05:08 05:38 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58
Arrêt4	05:08 05:38 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58
Arrêt5	05:08 05:38 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58
Arrêt6	05:08 05:38 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58
Arrêt7	05:08 05:38 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58

Figure 11 –Affichage des heures version Web

Cet écran présente donc l'heure de passage du bus en fonction de l'arrêt et du jour.

### Adaptation au contexte d'usage

Si nous passons à la version mobile de ce tableau nous pouvons remarquer que nous perdons de la lisibilité quand nous descendons dans le tableau, car ne nous voyons plus l'en tête (bien que celle-ci est assez facile à retenir). Nous avons donc développé une version qui s'est avéré plus pratique pour les utilisateurs (après tests avec eux) qui reprend les en tête à chaque arrêt et du coup permet de ne pas perdre de l'information.

Let's Go App			
	05:53 06:38	06:38 07:23	06:38 07:23
	07:23 08:13	08:13 08:58	08:13 08:58
	08:58 09:43	09:43 10:28	09:43 10:28
	10:28 11:13	11:13 11:58	11:13 11:58
	11:58 12:43	12:43 13:28	12:43 13:28
	13:28 14:13	14:13 14:58	14:13 14:58
Arret5	05:08 05:38	05:20 05:53	05:20 05:53
	05:53 06:38	06:38 07:23	06:38 07:23
	07:23 08:13	08:13 08:58	08:13 08:58
	08:58 09:43	09:43 10:28	09:43 10:28
	10:28 11:13	11:13 11:58	11:13 11:58
	11:58 12:43	12:43 13:28	12:43 13:28
	13:28 14:13	14:13 14:58	14:13 14:58
Arret6	05:08 05:38	05:20 05:53	05:20 05:53
	05:53 06:38	06:38 07:23	06:38 07:23
	07:23 08:13	08:13 08:58	08:13 08:58
	08:58 09:43	09:43 10:28	09:43 10:28
	10:28 11:13	11:13 11:58	11:13 11:58
	11:58 12:43	12:43 13:28	12:43 13:28
	13:28 14:13	14:13 14:58	14:13 14:58
Arret7	05:08 05:38	05:20 05:53	05:20 05:53
	05:53 06:38	06:38 07:23	06:38 07:23
	07:23 08:13	08:13 08:58	08:13 08:58
	08:58 09:43	09:43 10:28	09:43 10:28
	10:28 11:13	11:13 11:58	11:13 11:58
	11:58 12:43	12:43 13:28	12:43 13:28
	13:28 14:13	14:13 14:58	14:13 14:58

Liste des lignes	
Arret1	
<b>Lundi-Vendredi</b>	05:08 05:38 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13
<b>Samedi</b>	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58
<b>Dimanche</b>	05:20 05:53 06:38 07:23 08:13 08:58 09:43 10:28 11:13 11:58 12:43 13:28 14:13 14:58
Arret2	
<b>Lundi-Vendredi</b>	05:08 05:38 05:53 06:38 07:23 08:13

Figure 12 – Adaptation Tableau

Là encore c'est grâce à une media-query appliqué au css que nous pouvons adapter le tableau. Nous avons tiré cette partie du code du codepen de [sergio\\_pinna](#). Néanmoins la solution de répéter les en-têtes pour gagner en lisibilité sur les smartphones n'est pas la bonne lorsqu'il y a peu de colonnes. Par exemple si nous essayons de l'appliquer au tableau de la page précédente nous remarquons que nous perdons énormément d'espace et gagnons en complexité.

Liste des horaires ligne 1	
Choisissez heure de départ ▾	
Arret	Arret1
Heures	8h30
Arret	Arret2
Heures	9h00
Arret	Arret3
Heures	9h30
Arret	Arret4
Heures	10h00
Arret	Arret5
Heures	10h30
Arret	Arret6
Heures	11h00

Liste des horaires ligne 1	
Choisissez heure de départ ▾	
Arret	Heures
Arret1	8h30
Arret2	9h00
Arret3	9h30
Arret4	10h00
Arret5	10h30
Arret6	11h00

Figure 13 – Adaptation tableau qui rend plus complexe la vue



## Avantages

Bootstrap est un excellent outil permettant de faire des mises en page de site internet facilement et rapidement. Une très grande documentation sur internet permet d'approfondir chaque fonctionnalité pour pouvoir en tirer meilleur parti. Ainsi il est possible en plus de la mise en page d'afficher un certain nombre d'animations (diaporama, scrolling automatique...) sans aucune connaissance en JavaScript. Son système de colonne permet d'adapter facilement chaque site internet à tout type de dispositif comme les tablettes et les smartphones. Cette bonne intégration avec les outils vectoriels est un vrai plus pour les designers de site internet.

## Limites

Bootstrap reste un outil de mise en page et n'est pas suffisant pour faire des animations poussées aisément (animation possible en CSS3, mais longue et laborieuse). Le fait que le framework soit aussi facile à maîtriser rend la plupart des sites internet identiques. Nous pouvons voir assez facilement sur un site qu'il utilise Bootstrap. On ne peut pas utiliser que Bootstrap il faut obligatoirement d'autres outils/framework pour pouvoir rendre un code fiable, clair, lisible et testable.

## Bibliographie

<http://api.jquery.com/>

<https://www.w3schools.com/bootstrap/>

<https://developer.mozilla.org/fr/docs/Web/>

<https://getbootstrap.com/docs/3.3/>

# Conclusion

## Swift

Le développement en SWIFT procure plusieurs avantages, notamment une liberté totale lors de l'implémentation des fonctionnalités car nous maîtrisons le code de bout en bout. Nous jouissons aussi d'une rapidité accrue de part la rapidité du langage en lui-même<sup>3</sup>. En contre partie, cela implique un développement qui se limite aux plateformes iOS (macOS, iOS, tvOS et watchOS) et de plus il faut nécessairement disposer d'un compte développeur et d'un ordinateur tournant sous macOS. Enfin, il faut aussi prendre en compte l'accessibilité du langage. Même si Apple a fait de gros efforts pour rendre SWIFT plus accessible et plus lisible que Objective-C, cela reste un langage à part entière avec son propre SDK à apprendre. Par comparaison, faire une application simple en Ionic est plus rapide et facile grâce aux technologies web qu'il y a derrière.

## Ionic

Ionic 3 permet de développer une application mobile relativement aisément et rapidement pour peu que l'on sache utiliser HTML, CSS, Javascript et Angular 3. Pour le développeur, le développement de l'application se rapproche énormément du développement d'un site web. Il n'y a pas de contrainte vis-à-vis d'un IDE, la compilation et l'exécution sont simples avec le serveur Ionic qui nécessite uniquement une page web. Pour pouvoir déployer l'application il faut néanmoins Android Studio et XCode d'installer et configuré pour déployer sur Android et IOS.

Ionic permet de créer des applications pour smartphone qui utilise des capteurs. Il faut installer des modules supplémentaires, selon les capteurs que l'on veut utiliser, mais cette manipulation est facile. Cependant, l'exécution avec le serveur Ionic ne peut pas accéder à certain capteur comme la boussole. Viens alors le problème majeur d'Ionic qui est la performance. Dans notre projet, ce problème c'est traduit majoritairement par un accès relativement long aux capteurs GPS puisque la récupération de la valeur prend environ trois secondes. D'ailleurs, la lenteur du capteur GPS avec Ionic est une critique courante sur internet. De plus, l'application a été détectée comme énergivore par un smartphone Android pendant les tests même si utiliser constamment le capteur GPS à tendance à faire fondre la batterie du dispositif.

## Vue.js

De cette expérience, nous avons pu remarquer quelques défauts quant à l'utilisation des Web Component. Il implique qu'il faille définir en avance les composants. Mais si l'on veut le modifier

---

<sup>3</sup> Sa rapidité lui permet même de s'exécuter côté serveur, source : <https://www.macg.co/logiciels/2016/10/sur-le-serveur-swift-est-nettement-plus-rapide-que-nodejs-95921>

plus tard alors que nous avons utilisé ce dernier à plusieurs reprises dans l'application, cela risque de créer un désordre dans l'aperçu. Aussi, une mauvaise analyse pourrait impliquer une implémentation de certains composants qui auraient pu être fusionnés.

De plus, la communication entre différents composants peut s'avérer fastidieuse. En effet, il faut passer par une procédure particulière qui introduit l'utilisation des **Event.bus**. C'est-à-dire un composant vide avec qui les composants vont communiquer à l'aide `$emit` et `$on`.

Néanmoins, l'utilisation des Web Component s'avère pratique dans le développement d'application Web.

En effet, l'utilisation des composants personnalisables permet de factoriser le code en détectant les éléments communs, et permet donc d'être plus efficace si le composant est répété de nombreuses fois.

Le fait que le code soit découpé en de nombreuses sous-parties permet une légèreté et une meilleure lisibilité, ce qui permet d'identifier plus rapidement les erreurs et rendre le code plus maintenable.

De plus, le fait que tout ce qui est propre au composant soit centralisé dans un seul est même fichier permet un meilleur contrôle et éviter que les feuilles de styles entrent en conflit.

## Bootstrap

Bootstrap est un excellent framework qui nous a permis de construire rapidement et facilement la base du projet. En regardant en détail le framework, celui-ci permet de faire des animations de manière simple et sans connaître le JavaScript. Cependant nous avons bien vu les limites quand il s'agissait de modifier en détail certains éléments graphiques. En effet nous avons dû rajouter du code JQuery pour parfaire quelques petits détail donnant une meilleure adaptation et donc une meilleure expérience utilisateur.

La partie Vue.js aurait permis au projet d'être plus clair dans la construction du code et pouvoir gérer plusieurs composants de manière indépendante notamment les panel-box. On y gagne en clarté du code.

Nous aurions apporté à la partie Vue.js le côté responsive du framework.

Cependant l'intégration des deux technologies aurait été beaucoup plus complexe, car vue.js ne se manipule pas de la même façon que bootstrap.

En termes d'adaptation nous n'avons pas joué sur la même carte avec Vue.js. En effet bootstrap va plus s'adapter au dispositif sur lequel est utilisé le site web tandis que Vue.js va s'adapter au contexte d'utilisation. Ceci est dû au fait que Vue.js n'est pas un framework responsive design et ne peut donc pas s'adapter facilement au smartphone. Du côté de bootstrap, il nous a semblé plus logique de développer l'adaptation au dispositif qui est le point fort de ce framework.

## Général

Enfin, nous pouvons conclure que suivant la problématique nous ne sommes pas confrontés à la même adaptation. En effet, les applications mobiles auront plus tendance à s'adapter à l'environnement et l'utilisateur tandis que le site web s'adapte plus au dispositif et à l'utilisateur. L'avantage de l'application mobile est qu'elle va pouvoir exploiter les capteurs du téléphone pour s'adapter à l'environnement. La petite taille de l'écran oblige aussi à sélectionner l'information que l'on veut afficher.

Côté web l'accès aux capteurs est plus limité (géolocalisation possible selon le navigateur utilisé) mais le contexte d'usage est plus restreint, car un utilisateur visualisant le site sur son portable ne s'attend pas à avoir des fonctionnalités avancées alors qu'il existe une application dédiée pour ça il faut donc jouer sur l'adaptabilité du site sur son smartphone afin qu'il retrouve toutes les informations du site classique tout en ayant un affichage ergonomique.

Nous pouvons donc dire que les enjeux sont différents selon si l'utilisateur utilise l'application mobile ou le site web.