BlablaCar 2.0

Application multiplateforme basée sur le framework Ionic

Branger Jonathan – Ducros Alix

Fonctionnalités de l'application

L'application doit permettre à un utilisateur quel qu'il soit de consulter la liste des covoiturages disponibles. Cette fonctionnalité est représentée par l'écran principal de l'application qui est affiché dès l'ouverture de l'application. Sur cet écran, l'utilisateur peut faire une recherche en filtrant par lieu de départ, d'arrivée et par date de trajet. Plutôt que d'entrer manuellement son lieu de départ, l'utilisateur a la possibilité d'utiliser le capteur GPS du téléphone afin de trouver la ville la plus proche de sa position.

Si l'utilisateur est connecté, il peut réserver un trajet en cliquant sur le bouton « Réserver » dans la tuile du trajet. De plus, il a maintenant accès à 3 autres onglets de l'application.

L'utilisateur connecté a accès à ses paramètres de compte via l'onglet représentant une maison. Sur cet écran il peut modifier son nom, son login et son mot de passe.

Il a aussi accès à la liste des trajets qu'il a réservé ou qu'il a proposé. Il retrouve dans cette liste les informations sur ces trajets et peut soit annuler sa réservation soit annuler le trajet (en fonction de si il est simple passager ou conducteur).

Enfin, dans le dernier onglet, l'utilisateur peut créer un trajet en spécifiant le lieu de départ, le lieu d'arrivée, la date ainsi que le nombre de places disponibles.

Points techniques

Recherche de trajets

Afin de rendre la recherche de trajets plus facile pour l'utilisateur, nous avons décidé que la recherche renverrait des résultats sans prendre en compte la casse et en proposant les lieux dont le nom contient ce que l'utilisateur a écrit.

Plus concrètement, on parcourt la liste des trajets existants et on effectue la vérification suivante :

```
if( trajet.depart.toUpperCase().indexOf(viewModel.searchInputStart.toUpperCase()) != -
1) {
	departMatched = true ;
}
if(trajet.destination.toUpperCase().indexOf(viewModel.searchInputDestination.toUpperCas
e()) != -1) {
	destinationMatched = true ;
}

if(departMatched && destinationMatched) {
	viewModel.matchingTrajets.push(trajet);
}
```

Ainsi, si l'utilisateur cherche un départ de Nice, il lui suffit d'écrire « Nice », « nice » ou même seulement « i ».

Nous avons souhaité donner à l'utilisateur la possibilité de trier les trajets en fonction de leur date (de manière ascendante ou descendante) ; pour mettre cette fonctionnalité en place nous avons utilisé le filtre orderBy proposé par Angular et mis en place les éléments nécessaire à la définition des différents tris :

```
viewModel.ordersList = [
	{
	    id: 1,
	    title: 'Date ascendante',
	    key: 'date',
	    reverse: false
	},
	{
```

```
            id: 2,
            title: 'Date descendante',
            key: 'date',
            reverse: true
        }
    ];

    viewModel.order = viewModel.ordersList[0];
```

Gestion des trajets

Afin de garder le nombre de passagers cohérent, l'utilisateur ne peut réserver que les trajets ayant encore des places disponibles. (Initialement, les trajets complets n'apparaissent tout simplement plus dans l'écran de recherche de trajets, mais à des fins de démonstration, on ne fait qu'empêcher leur réservation.)

Pour permettre la mise en place de ce mécanisme, l'application se repose de manière intensive sur les possibilités du framework Angular et plus précisément le binding afin que les modifications du modèle aient une répercussion immédiate sur la vue et sur l'attribut XML « ng-if » qui nous permet de masquer des possibilités à l'utilisateur en fonction du trajet.

Gestion des utilisateurs non connectés

Si l'utilisateur n'est pas connecté on souhaite limiter ses intéractions possibles avec l'application : il ne doit pas avoir accès aux paramètres de compte, ni à la liste des trajets le concernant et ne doit pas avoir la possibilité de créer un nouveau trajet. Ainsi dans chacun des onglets concernés, le contenu entier de l'onglet est remplacé par un cadre invitant l'utilisateur soit à se connecter soit à s'inscrire.

Quel que soit le choix de l'utilisateur, une fenêtre modale apparait avec un formulaire qui lui permet de faire l'action requise.

Naturellement le mécanisme que nous venons d'exposer est répété 3 fois et un simple copier-coller mènerait à un risque d'erreur bien trop important. Pour pallier à ce problème, nous avons utilisé la balise « ng-include » qui permet d'intégrer du code HTML redondant, nous permettant ainsi d'assainir notre code.

Perspectives

Bien que nous soyons relativement satisfaits par le résultat auquel nous sommes arrivés, notre application manque de certaines fonctionnalités.

- Utilisation des API permettant l'accès aux capteurs du téléphone : nous aurions aimé utiliser la caméra du téléphone pour permettre à l'utilisateur de se prendre en photo ou de choisir une photo dans sa galerie à la création de son compte. Nous aurions aussi aimé ajouter la possibilité de récupérer la ville la proche grâce au GPS afin de remplir le champ de recherche correspondant à au lieu de départ.

- Utilisation du local storage : nous n'avons pas eu le temps de mettre en place une quelconque persistance de donnée qui aurait permis de conserver d'une fois sur l'autre les trajets et les utilisateurs créés ou modifiés.

- Ajout d'un écran permettant de détailler un trajet : nous avons envisagé la création d'un écran de détail d'un trajet qui permettrait à l'utilisateur de connaitre les autres passagers qui voyageront avec lui.

- Mise en place de notifications : nous aurions aimé émettre une notification push à l'utilisateur chauffeur à chaque fois qu'une passager s'ajoute à son trajet.