TP Front-End : React Native

F. Coelho et O. Hermant

20 décembre 2024
6 janvier 2025

1 Environnement de travail

Ce TP utilise la machine virtuelle que nous vous avons fournie ou une installation manuelle (Mac M*). N'oubliez pas d'éteindre votre VM proprement, par le menu interne à la VM, à la fin de votre session. Si tout fonctionne, allez directement à la partie suivante. Sinon, des ordinateurs de prêt disponibles à l'audio-visuel (en L121), que vous pouvez emprunter jusqu'à la fin du PI MobApp. Voici alors les étapes à suivre :

1. Se logger et installer l'image `.ova` dans son répertoire personnel, en sélectionnant ledit répertoire dans l'option d'import "Machine Base Folder".
2. Après l'import, dans la configuration de la VM (VM éteinte), changer si besoin l'option USB à "contrôleur 3.0" et allouer 4Go de RAM à cette machine virtuelle.
3. Il faut alors refaire une partie du TP de prise en main de la tablette, pour connecter celle-ci et la VM. Voir ce TP pour les détails.
4. Surtout, **il faut add/commit/push vos fichiers systématiquement** car la VM est liée à un ordinateur, et vous pouvez être amené à en changer. Il faudra, à chaque changement, refaire cette procédure, en plus de devoir cloner à nouveau le dépot git si besoin.

2 GitLab : configuration, fork et clonage d'un projet

Si ce n'est pas déjà fait, finalisez la configuration de votre compte gitlab CRI : `https://gitlab.cri.minesparis.psl.eu/` en vous rendant sur ce serveur, puis en récupérant votre mot de passe par la procédure d'oubli.

Le projet initial de ce TP reprend les exemples de cours. Pour le récupérer, suivez la procédure décrite ici : `https://gitlab.cri.minesparis.psl.eu/mobapp-2024/kivappa`.

3 Projet existant: installation

Dans la VM, une fois le clonage du projet git effectué :

1. **Dans un terminal**, allez dans le répertoire `kivappa`, correspondant au repository git cloné précédemment. Pour ce faire, utilisez les commandes `cd, ls`.

   Toutes les actions et commandes de ce TP qui utilisent un terminal seront à faire en étant dans ce répertoire `kivappa`. En cas de doute, vérifiez que vous êtes bien dans celui-ci, par exemple avec la commande `pwd`.

2. Installez les modules node requis (les dépendances de votre projet) par:

   ```
   mobapp@vm-lmde:~/.../kivappa/$ npm install
   ```

   Cette étape n'a besoin d'être faite qu'à certains moments précis: juste après un clone ou lorsqu'un ou une de vos camarades a ajouté (et commit/push) l'utilisation d'une nouvelle librairie[1]

3. Vérifiez que la tablette/un téléphone android est bien connecté, avec `adb devices` (cf. tp sur la tablette).

[1] NB : lorsque l'on se met à utiliser `ma-sper-librairie`, en plus de son inclusion dans les fichiers qui l'utilisent, il faut aussi faire l'installation avec `npm install ma-super-librairie`. De même, pensez à `npm remove` les libraries dont vous ne vous servez plus.

4. Augmentez les ressources systèmes allouées à watchman (ou bien relancer le point 5 à chaque fois qu'il plante), en exécutant, dans un terminal, le script

   ```
   mobapp@vm-lmde:~/.../kivappa/$ watchman-fix.sh
   ```

5. **lancer le packager d'application** de react-native (rappel : voir point 1), avec la commande :

   ```
   mobapp@vm-lmde:~/.../kivappa/$ npm start
   ```

6. Enfin, tapez "a" pour lancer la compilation et l'installation de KivAppA sur la tablette/le téléphone.

   Il est normal que cela prenne plusieurs minutes, surtout la première fois. Ensuite, à chaque nouvelle modification et sauvegarde de vos fichiers, l'application mobile devrait se recharger automatiquement. Si ce n'est pas le cas, appuyer sur la touche "r" dans la console (pour **r**eload l'app). Si cela ne marche pas non plus, appuyer de nouveau sur "a". En désespoir de cause, il faudra refaire toute la procédure ci-dessus.

7. être **très** patient et vérifier régulièrement l'état de la compilation du projet *et* de son installation.

   Une fois que c'est terminé, il devrait y avoir sur votre tablette/téléphone une app, nommée `kivappa` avec un bouton et un compteur, comme vu en cours.

4 Extensions de l'app (Décembre 2024)

Vérifiez régulièrement (`git status`) que les fichiers que vous avez modifiés sont bien inclus dans le commit à venir. Faire un **`git add`** de tous ces fichiers, puis un **`git commit`** et **`git push`**.

4.1 Bouton de remise à zéro

Dans ce premier exercice, ajoutez à votre application un bouton de remise à zéro du compteur.
Facultatif : alignez les boutons sur une seule ligne.

4.2 Bouton bonjour

Ajoutez un composant **`<Text>`** et un bouton. Un clic devra afficher (dans le **`<Text>`**) un message de bienvenue.

4.3 Bouton de changement de style

Ajoutez un autre bouton qui permette de changer la couleur de fond de votre application. Étapes suggérées :

- Changez la propriété de style "couleur de fond", en dur dans le code (hardcodé).
- Utilisez une variable d'état (avec `useState`) pour définir dynamiquement cette propriété.

Facultatif :

- récupérez la couleur de fond depuis un champ d'input donné par l'utilisateur (liste déroulante, par exemple)
- ajoutez une option "aléatoire" à cette liste et tirez une couleur au sort dans tout l'espace de couleurs RGB.

4.4 Convertisseur Fahrenheit – Celsius

En dessous de tous ces boutons, développez une fonctionnalité de conversion Celsius – Fahrenheit. Quelques étapes possibles, par degré de difficulté croissant :

1. Développez et testez vos fonctions de conversion, en javascript pur. Rappel :

$$\begin{aligned} X\ ^\circ\mathrm{C} &= \tfrac{9}{5} * (Y\ ^\circ\mathrm{F} - 32) \\ Y\ ^\circ\mathrm{F} &= \tfrac{5}{9} * X\ ^\circ\mathrm{C} + 32 \end{aligned}$$

2. Ajoutez deux boutons, un pour chaque direction $^\circ F \leftrightarrow {}^\circ C$, un **`<TextInput>`** dans lesquel vous lirez la valeur à convertir, et un champ de texte où vous écrirez le résultat de votre conversion.
3. *Facultatif* : faites une conversion *à la volée*, sans aucun bouton. À chaque fois que votre utilisateur modifiera la température dans le champ d'input, la valeur du résultat devra se mettre à jour automatiquement.
4. *Facultatif* : rendez le comportement de la question précédente *symétrique*. Exemple : `https://www.google.com/search?q=fahrenheit+celsius`

4.5 Style

Ajoutez du style à votre application. Il pourra notamment être utile de vous documenter sur la méthode de disposition des "boîtes flexibles" https://reactnative.dev/docs/flexbox.

Attention : React Native impose de remplacer les `<Button>` par des `<Pressable>` contenant un `<Text>` pour que ceux-ci soient bien pris en compte par l'algorithme flexbox. Pensez à faire ce changement, et à ajouter du style pour rendre vos `<Pressable>` plus conviviaux : pour imiter le style des boutons il faudra sans doute intégrer une `<View>` à l'intérieur et lui donner du style.

4.6 Fichiers séparés

Séparez vos composantes, ainsi que le style, en différents fichiers. N'oubliez pas, pour chacun des fichiers que vous créez, de les ajouter à votre repository git avec `git add`. Et, bien entendu, de commit/push.

4.7 S'il vous reste du temps

Faites les parties facultatives de cette section.

5 Requêtes au serveur KiVa (Janvier 2025)

5.1 Bouton de changement d'écran

Faites en sorte de pouvoir basculer à l'envi entre votre travail de la section précédente, qui devrait intégralement être contenu dans le composant `<Decembre>` et le travail de cette section, qui devra être contenu dans le composant `<Janvier>`. Ce dernier est défini dans le fichier `Janvier.js`, et il reprend des éléments du second cours. Éditer ce fichier avec VSCode (`code Janvier.js`), lire et comprendre son organisation.

5.2 Requêtes à un serveur distant

La fonction `sendRequest` utilise la librairie `axios` pour envoyer une requête `GET` au serveur `https://kiva.mobapp.minesparis.psl.eu/api`, avec une authentification basique et le login/mot de passe `kiva/bien`. Elle a deux paramètres :

1. un dictionnaire `params`, qui permet de passer des paramètres à la requête. Essayez, par exemple, avec la valeur `%l%`

2. `updateFunction`, la fonction qui servira à mettre à jour l'application KivAppA en fonction du contenu de la réponse du serveur. Dans le cas présent, il s'agit de mettre à jour une des variables d'état de l'app avec la liste de couples renvoyés, l'argument sera donc `setList` (ligne 45).

   La fonction de mise à jour est appelée ligne 17, au coeur de la fonction de callback enregistrée auprès de (la promesse retournée par) axios, lignes 15 à 21.

   Cette fonction de callback sera automatiquement appelée par axios, qui lui fournira en argument un objet `response`, dont les champs contiennent, entre autres, le statut (`response.status`) et le contenu (`response.data`) de la réponse obtenue du serveur, qui sera ensuite transmise à `updateFunction`.

5.3 Analyse du code

Quels sont les composants et les fonctionnalités qui font partie de cette app ? Les documenter **à l'aide de quelques commentaires** dans les fichiers `App.js` et `Janvier.js`. Ne pas oublier de `git add Janvier.js`, puis `git commit` et `git push`.

5.4 Requête sans paramètre

Modifier le comportement de la fonction `sendRequest` pour que la requête au serveur se fasse **sans** le paramètre `flt`. Puis remettez en place ce paramètre.

5.5 Argument supplémentaire

Ajouter un troisième paramètre à la fonction `sendRequest`, afin que l'on puisse lui spécifier en argument la route de l'API sur laquelle faire la requête, plutôt que de la coder en dur avec `url : '/store'`.

5.6 Mise à jour continue de la liste clefs/valeurs

Faites en sorte que la liste des clefs/valeurs correspondant au filtre se mette à jour *à chaque fois que l'utilisateur tape un caractère*. Le pressable `"Envoyer"`, lui, sera modifié pour sélectionner *tous* les enregistrements.

6 Connexion au serveur Flask KiVa local

1. Modifier votre application cliente pour se connecter à un serveur local `http://localhost:5000`.
2. Reprenez votre serveur flask `kiva`, et lancez-le dans un second terminal. Pour rappel, dans le répertoire `back-end`, exécutez la commande `make clean log`. Vérifiez son bon fonctionnement, par exemple avec la commande `curl -i 'http://0.0.0.0:5000/info'`.
3. Côté client, il faut **explicitement transférer les requêtes HTTP** faites par l'app (sur la tablette/le téléphone) au serveur, qui, lui, est sur la VM/sur votre ordinateur. Pour ceci, il faut demander à `adb` de faire ce qui s'appelle du *reverse port forwarding*. Dans un terminal, taper :

   ```
   adb reverse tcp:5000 tcp:5000
   ```

   Cela a pour effet de transférer les requêtes faites sur le port `5000` de la tablette, vers le port `5000` de la VM, là où justement votre serveur KiVa les attend.

   L'app KivAppA aura toujours l'impression d'interroger `http://localhost:5000` (*localhost* est la tablette/le téléphone, du point de vue de l'application) mais elles seront secrètement redirigées par `adb` là où il faut.

Si tout fonctionne, votre application mobile est désormais capable de faire des requêtes sur le serveur distant aussi bien que sur votre propre serveur local. Testez.

7 Développement full-stack

Il s'agit maintenant de développer conjointement votre API REST KiVa locale et votre application KivAppA. Si vous manquez de routes sur votre API, développez le serveur, et s'il manque des fonctionalités à votre application mobile pour utiliser toutes les routes de votre API, développez votre application.

7.1 Requêtes sur d'autres routes de l'API

Ajoutez tous les boutons, les champs et les fonctions nécéssaires pour que l'application mobile implémente :

- l'insertion d'un nouveau couple clef/valeur,
- la suppression du couple associé à une clef de la table.

Voir le TP sur HTML/CSS/JS pour les routes et méthodes à implémenter dans votre serveur local `kiva`. Encore une fois, ne pas oublier de add/commit/push.

7.2 Configuration dynamique de la connexion au serveur

Ajoutez des champs à votre app, pour que l'utilisateur puisse configurer l'adresse du serveur et les login/mot de passe. Remplissez ces champs à une valeur par défaut, et testez la connexion à votre serveur KiVa local, mais aussi au serveur distant.

Cette configuration devra être sur *un écran séparé*. Il faudra peut-être partager quelques informations entre l'écran de configuration et l'écran principal de l'application à l'aide des `props` de react native. Puis add/commit/push.

7.3 *(Facultatif)* Nettoyage

Vos camarades auront sûrement alimenté la table KiVa (du serveur distant) avec bon goût. Développez une fonctionnalité de nettoyage manuel, mais pas automatique — vos camarades ont aussi besoin de visualiser l'effet des requêtes qu'ils font. Exemple : un bouton "supprimer" à côté de chacun des enregistrement affichés.

Séparez vos fonctionnalités et composants en différents fichiers.