TP Front End: Advanced React Native

Development environment

➔ See TP Front-End: React Native

Ce TP utilise la machine virtuelle que nous vous avons fournie.
Si celle-ci ne fonctionne pas, vous pouvez installer localement cette VM sur les postes fixes qui sont situes en L119 en suivant scrupuleusement les instructions suivantes :

1. choisir un ordinateur pour y installer la VM.
2. Demarrer la machine sous linux ; demarrer virtualbox.
3. Brancher la clef USB qui contient l'image .ova de la VM, ou bien telecharger celle-ci depuis le cloud (cf. le TP d'installation de la VM)

   dans le repertoire /var/tmp

4. importer la VM

   dans les repertoire /var/tmp

   pour ce faire, donner /var/tmp dans l'option d'import "Machine Base Folder".
   L'installer dans votre repertoire personnel pourrait vous causer des problemes de quota disque, aussi nous ne le recommandons pas.
5. apres l'import, dans la configuration de la VM (VM eteinte), changer l'option USB de "controleur 3.0" a "Controleur 1.1".
6. il vous faudra aussi refaire une partie du TP de prise en main de la tablette, pour connecter celle-ci a la VM.
7. enfin, ne pas oublier de sauvegarder ou de commit/push vos fichiers systematiquement.

La VM sera ensuite assignee a l'ordinateur choisi et a vous-meme. Si vous reutilisez le meme ordinateur la prochaine fois, vous n'aurez pas besoin de reinstaller la VM. En principe, vous pourrez la demarrer directement et vos fichiers y seront presents.

Starting application

I suggest starting back from where you were in the previous TP. This can be done via
    git clone git@gitlab.cri.ensmp.fr:[Your project]
where the git@ part can be found by going on the main page of your gitlab project and using Clone > Clone via SSH

Note:

- If you want to store both the front end and the backend on the same git repo, you might have to remove the git submodule. This can be done via
    - `cd front-end`
    - `rm -rf .git`
    - `cd ..`
    - `git rm front-end`
- If you have more files than you wish for when you run “git status”, it might be because some of the build files are getting stored in git. You can add a file called .gitignore at the root of your repository to remove some of those. Feel free to search for “.gitignore react-native” on google for examples. (This also works for the flask backend)

Reminder:

- `adb devices` ensures that the tablette is connected to the VM
- `npm install` ensure that all the dependencies of the project are present. It should be run in the same directory where “package.json” is. It’s usually your front-end folder.
- `watchman_fix.sh` increases the memory limit
- `npx react-native start` starts the react native bundler which is required to compile react native apps
- `npx react-native run-android` starts the react app. It should be run in a dedicated terminal
- `adb reverse tcp:5000 tcp:5000` is required to talk to the backend server on the virtual machine
- `code .` starts vscode in the current directory

START HERE ➔ Extension of the previous TP

The best is to start from the last front end TP.
Otherwise you can clone the starting files again:

- git@gitlab.cri.ensmp.fr:mobapp-2022/eleves/kivappa.git
- https://gitlab.cri.ensmp.fr/mobapp-2022/eleves/kivappa.git

The goal of this TP will be to build a front end system able to handle:

- A logged out view (default)
    - A component to login. This component will make a request to the server to validate a username/password input.
        - If the identifier/password is valid, the app will go to a logged in state
        - If the identifier/password is invalid, a warning message will be displayed
    - A component to create an account. This component will make a request to the server to create a new account.
        - If the account creation is successful, the app will move to the login view
- A logged in view
    - A Component to display all the users in the app (default)
        - The component will load all the users in the store and display their name
        - The component will have a button to reload the users
        - The component will load all the users on the initial render
    - A log out button

Extension:

- Add a way to filter the users in the logged in view
- Support data validation on account creation: What if a user already exists with the same username
- Support for admin users to delete other users

This TP will have a correction ☺

1. Architecture of the app

The first step is to clean up the application architecture into simple component. This will help you ensure that you won't have conflict when you collaborate in git later on.

- Move each react component and javascript function in App.js into their own file.
- Move the common function to a dedicated common folder
- Prepare a folder for the logged out view, and the logged in view

At the end, you should have a folder structure similar to:

```
core/
  | sendRequest.js
  | KeyValues.react.js
  | ApresMidi.react.js
  | Matin.react.js
  | DemiJournee.react.js
loggedOut/
loggedIn/
App.js
```

```
Root.react.js => This will be mostly empty at the moment
```

Notes:

- Root.react.js component will contain the logic to switch between Logged in view and logged out view.
- I usually use camelCase.js for javascript functions, and PascalCase.react.js for react component.
  Example: sendRequest ➔ sendRequest.js
  KeyValues ➔ KeyValues.react.js
- In general, javascript import make it difficult to have deeply nested folder hierarchy. I suggest having 1 or 2 folder depth maximum. The first depth for the current "view" of the app, the second depth for component specific to that view.

Remember imports in javascripts:
https://beta.reactjs.org/learn/importing-and-exporting-components

➢ Gallery.react.js

```
export default function Gallery() {
  return (
    <section>
      <h1>Amazing scientists</h1>
    </section>
  );
}
```

➢ OtherThing.react.js

```
import Gallery from './Gallery.react.js';

export default function OtherThing() {
  return (
    <Gallery />
  );
}
```

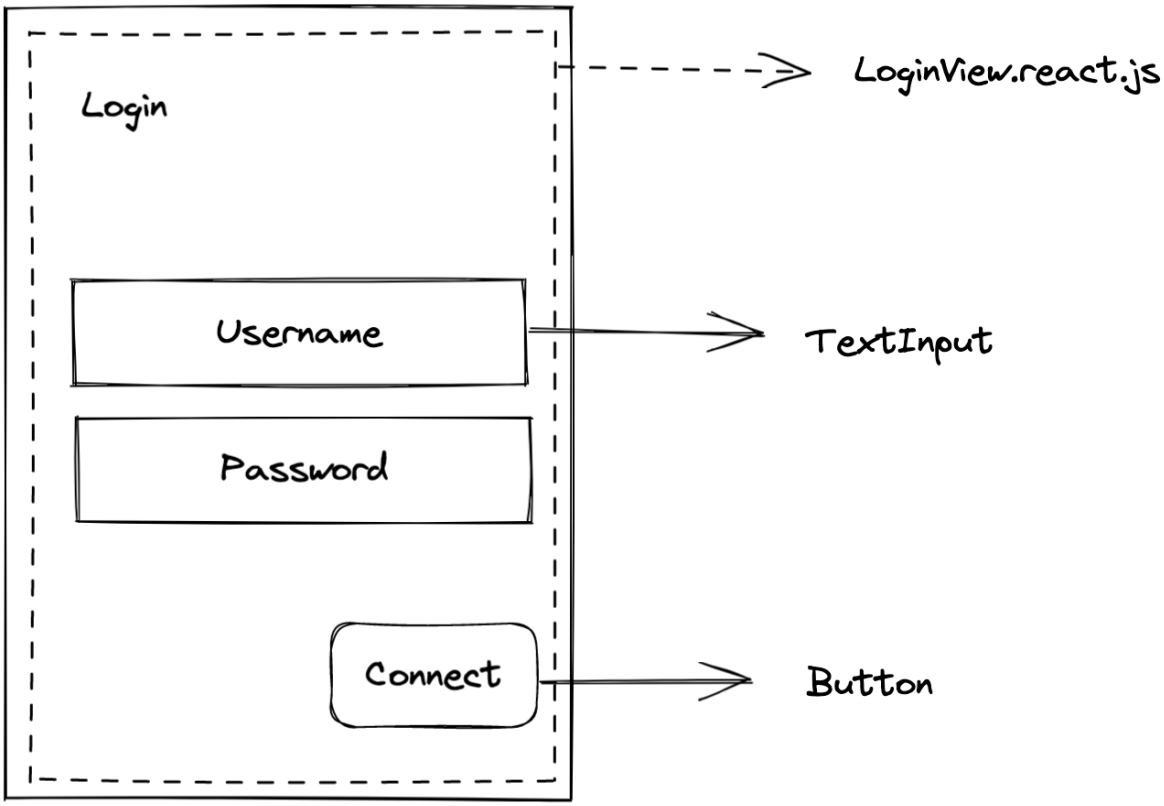
Note:

- ./Gallery.react.js ou ./Gallery.react both work in react
- Default and non-default import are different!
- `export default function Gallery` ➔ `import Gallery from './Gallery.react.js';`
- `export function Gallery` ➔ `import {Gallery} from './Gallery.react.js';`

2. Logged out view – Handling the login inputs

Step 1. Adding a TextInput component for the username

Create a file ‘loggedOut /LoginView.react.js’ which will contain our login component.
Render a TextInput inside the component for the username.



Step 2. Connect the username TextInput component

You will need to a state to update the value of the TextInput.

```
const [username, setUsername] = useState('default username');
```

When the TextInput is modified by the user, it triggers the event `onChangeText`. Use this event to update the value of the Text input with `setUsername`

Reference: https://beta.reactjs.org/reference/react/useState

Step 3. Repeat the previous steps with a TextInput component for the password field

Add this point, you will have 2 independent fields for the username and the password.

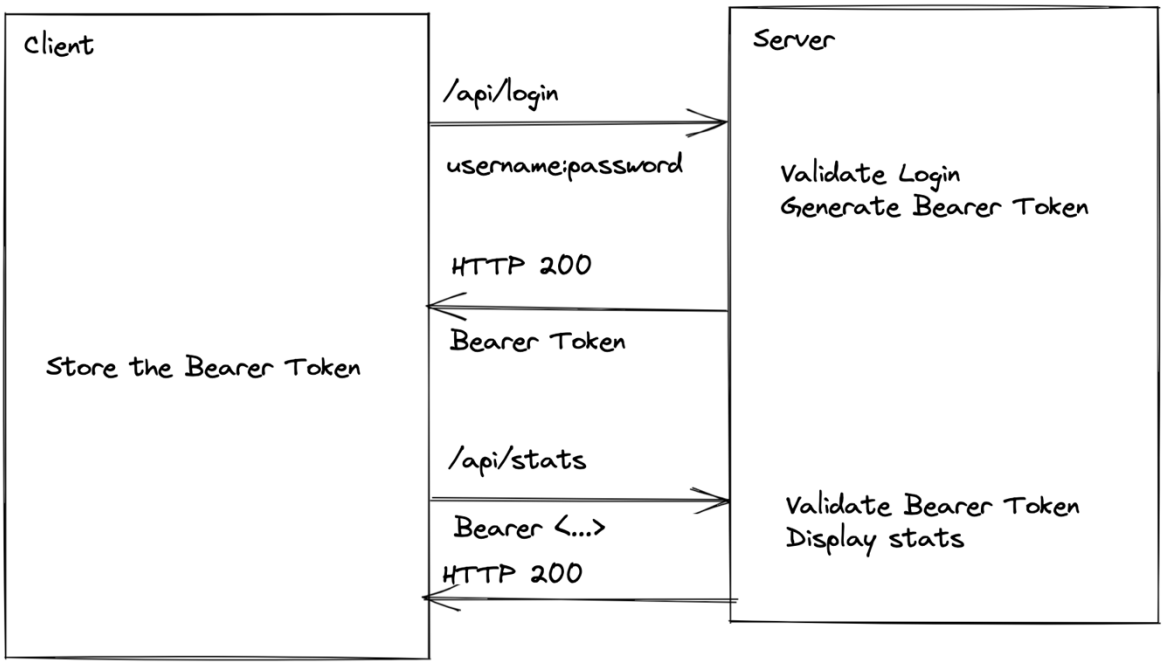
Step 4. Add a Button component to submit the login information

Add a button component which will be used to send a request to the server.

On the server, add a dedicated GET route `/api/login` to submit the user credentials. It should have 2 params (username and password)

Question:

- **What does this GET request return?**
  What happens when the credentials are correct / incorrect? Try the following command line calls




```
curl -i -u kiva:bien https:/kiva.mobapp.mines-paristech.fr/api/login
```

This call should give you a Bearer Token

```
curl -i -H "Authorization: Bearer <ACCESS_TOKEN>"
https:/kiva.mobapp.mines-paristech.fr/api/users
```

The Bearer Token can be re-used to call logged-in function

```
curl -i -u kiva:wrongpassword https:/kiva.mobapp.mines-paristech.fr/api/users
```

- **How are the credentials submitted?**
  Check the definition of sendRequest.

The server should validate that a user exists with those credentials and return the bearer token accordingly.

Step 5. Use the response to give feedback to the user (User Experience)

If the credentials are incorrect, the server returned no bearer token, display an error message.
If the credentials are correct, `console.log` this in the app. We will use the response later.

Note:
- Each one of those steps can (and should be) on a dedicated commit. Having simple commit with frequent merge makes it easier to resolve conflict.
- At this point, it might be useful to create a dedicated component for your inputs as this will make code re-use easier.
- **[I'm feeling lazy] If you don't want to tackle the server aspect today, look at the end of the TP.**

Logged out view – Creating a user

Prepare a second view to create a user. There should be a way to switch back and forth between the Login and the Create User view.

This view will be similar to the login view and have:
- One TextInput for the username
- One TextInput for the password
- One Button to submit the request

On click of the submit button, submit a POST request to the server to create the user.
The server side route is called `/api/users`.
The server should validate that a user does not exists with those credentials and create the user if possible.

If the response is correct, redirect to the Login view. If the response is incorrect, display an error message.

Logged in View – simple view

Add a second view in the application for the logged in page. The goal here is simply to have a visual change when the login action is successful.

This view should display the username & password of the current logged in account.
The login action should redirect to this view if the action is successful.

Note:
- You will need to be able to set the username and password in the Login action of the logged-out view, and also pass them down to the Logged in view. The Logged in and Logged out view don't have to be nested one into the other.

Logged in View – List users on button press

On the logged in – simple view, add an authenticated request to list all the users in the database. This should be triggered by a button.

The endpoint is GET `/api/users`

Logged in View – List users on render

https://beta.reactjs.org/reference/react/useEffect

On the logged in – list user view, add an authenticated request to list all the users in the database. This request should be triggered on the first render of the component.

Note:

- `useEffect(() => fetchMyData(), [])` enables fetching data on the first render of the component.
- `useEffect(() => fetchMyData(filter), [filter])` will fetch data on the first render of the component and every time filter changes.

Logged in View – Logout button

Add a way to log yourself out.

Logged in View – Support for admin users to delete other users

You will need to fetch during the login phase whether the user is an admin and enable deleting users conditionally.

Note:

- Should the data validation only be on the client?

Super Extension – useContext

Look at useContext and use it to share the username & password credentials in the app
https://beta.reactjs.org/reference/react/useContext

I’m feeling lazy – Skipping the server side development for the TP

```
export function maybeFetch(url, method, use_debug = false, debug_data = null) {
    if (use_debug) {
        return new Promise((resolve, reject) => {
            console.log('Hitting ' + url);
            setTimeout(() => resolve(debug_data), 2000);
        })
    }
    return fetch(url, {
            method: method,
        }).then((response) => response.json())
        .catch((e) => {
            alert('Something went wrong ' + e.message);
        });
}
```

The previous function can be used to simulate a response from the server

For the login component, it can be used like this:

```
maybeFetch(url, 'GET', true, { is_login_correct: true, is_admin: true, })
    .then((value) => { doSomething();});
```