TP Front-End : Advanced React Native

E. Tripoul & O. Hermant

15 janvier 2025

The paragraphs denoted with chili peppers are harder to implement. They are optional and should be treated only in a second time.

1 Environment Setup

We suggest starting from where you were in the previous TP. If you are already good with it, have a quick check of the reminder below and you can directly go to section 2.

If your development environment (VM/computer) is not yet fully set up, or you are not able to connect your phone/tablet, see the previous TP for installation instructions.

If you want a fresh copy of your previous work, you can do it via `git clone git@gitlab.cri.ensmp.fr:<pname>/<project>.git newFolder`. If you do so, do not forget to run `npm install` in your `newFolder`.

If your previous TP is in an unstable non-functional state, it is also possible to directly clone the starting files again :

- `git clone git@gitlab.cri.ensmp.fr:mobapp-2024/kivappa.git`
- or `git clone https://gitlab.cri.ensmp.fr/mobapp-2024/kivappa.git`

Beware that, in this case, you will not be able to push your changes on the distant gitlab repo. As said in the notes below, it is not necessary a huge issue.

Reminder :

- `adb devices` ensures that the tablet/phone is connected to your VM/computer.
- `npm install` ensures that all the dependencies of the project are present. It should be run in the directory where the file `package.json` is located, usually it is your front-end folder.
- `npm start` starts the react native bundler, which is required to compile react native apps. Typing `a` when the bundler is started builds, installs and runs your app on the tablet/phone (assuming the latter is connected, see `adb devices` above).
- `adb reverse tcp:5000 tcp:5000` is required for your application (on the tablet/phone) to talk to a locally running back-end server (on your VM/computer).
- `watchman-fix.sh` increases the file watching limits for the debugger of react native.
- `code .` starts vscode in the current directory.

Notes :

- Since this TP is not graded, you may decide not to save your work through `git add/commit/push`, especially if you still feel uncomfortable with `git`. However, it is recommended to use `git`.

In order to save you work without messing up the previous (and graded TP), if you are fluent enough in `git`, you may want to create a new branch.

- If you have a very high degree of `git`-awareness, you may also want to store both the front-end (this TP) and the back-end (your own KiVa server) on the same git repo. Then you might have to remove the git submodule. This can be done via
  - `cd front-end`
  - `rm -rf .git`
  - `cd ..` (assuming that the parent directory corresponds to the git repo of your back-end)
  - `git rm front-end`
  - `git add` whatever you want to add.

  In some cases, `git status` may display a lot of files to store. Most of them are build files, and it is wise not to store them in git. In other words, you need to configure git to ignore some files. Feel free to search for ".gitgnore react-native" for examples. This also applies to the Flask back-end.

2 Specs of the App

The goal of this TP is to build a front-end system able to handle :

- A logged-out view (default), containing
  - A component to log in. 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 succesful, the app will move to the login view.
- A logged-in view, containing
  - a component to display all the users of the app (default) :
    - the component will load all the users data from the server and display their names,
    - the component will have a button to reload the users,
    - the component will load all the users on the initial render;
  - a logout button.

Extensions :

- 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. And this correction will be the starting point of your project.

3 Architecture

Assuming that we start with the previous TP, the first step is to clean up your application architecture into simple components. This will help you reduce the conflicts when you collaborate on git later on.

- Move each react component and javascript function in `App.jsx` into their own file.
- Move the common function to a dedicated common folder.

- Prepare a folder for the logged out view, and a folder for the logged in view.

You should produce a folder structure that is similar to :

```
core/
    sendRequest.js
    KeyValues.jsx
    Janvier.jsx
    Decembre.jsx
    DemiJournee.jsx
loggedOut/
loggedIn/
App.jsx
Root.jsx
```

The last file will be mostly empty, at the moment. Notes :

- Check that your app is still compiling and working.
- `Root.jsx` will contain the logic to switch between the logged in and the logged out views.
- We usually use `camelCase.js` for javascript functions, and `PascalCase.jsx` for react components. Examples above : `sendRequest` and `KeyValues`.
- Javascript `import` makes it difficult to have a deeply nested folder hierarchy. We suggest having a folder depth of 1 or 2 at most. The first depth will be used for the "views" of the app, the second depth will be used for components specific to that view.

Keep in mind the structure of imports in javascript (for instance, as described here `https://beta.reactjs.org/learn/importing-and-exporting-components`) :

- file `Gallery.jsx`:

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

- file `OtherThing.jsx`:

```
import Gallery from './Gallery.jsx';

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

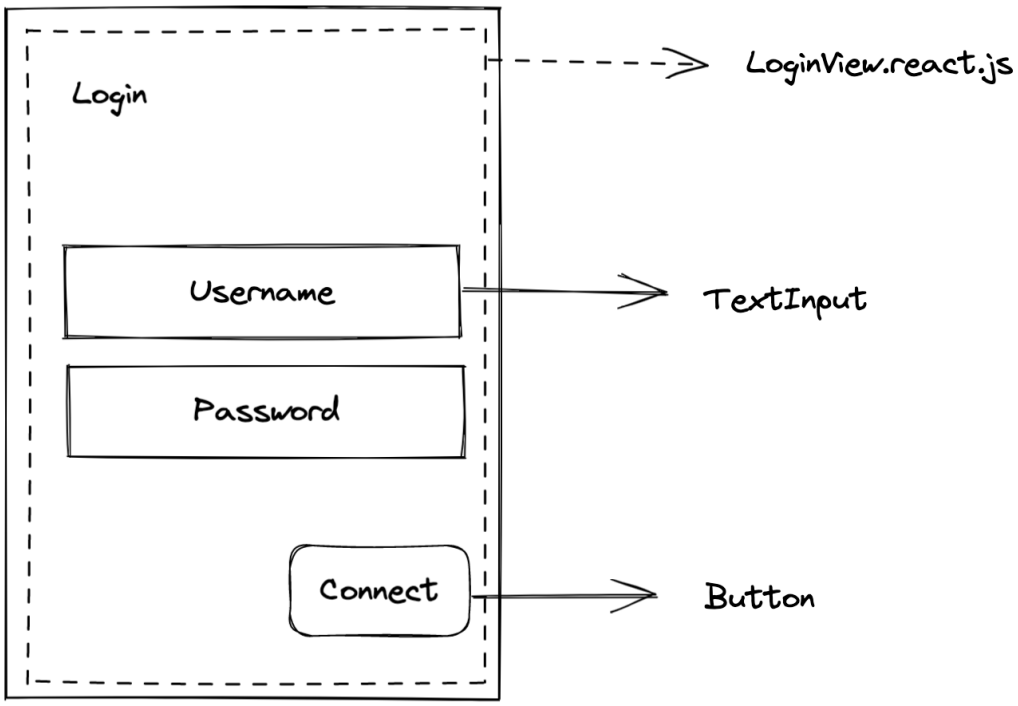
Notes :

- imports from `./Gallery.jsx` and `./Gallery` both work in react
- Default and non-default imports and exports work differently:
  - there can be at most **one** default export
  - `export default function Gallery()` implies `import Gallery from './Gallery.jsx';`
  - `export function Gallery()` implies `import {Gallery} from './Gallery.jsx';`

4 Logged-Out View – Handling the Login Phase

4.1 Step 1. Adding a TextInput Component for the username

Create a file `loggedOut/LoginView.jsx` which will contain our login component. The component should render a single `<TextInput>` component, that will be used for the username.



4.2 Step 2. Connect the username TextInput Component

To store and update the value of the TextInput, you will need a state:

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

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

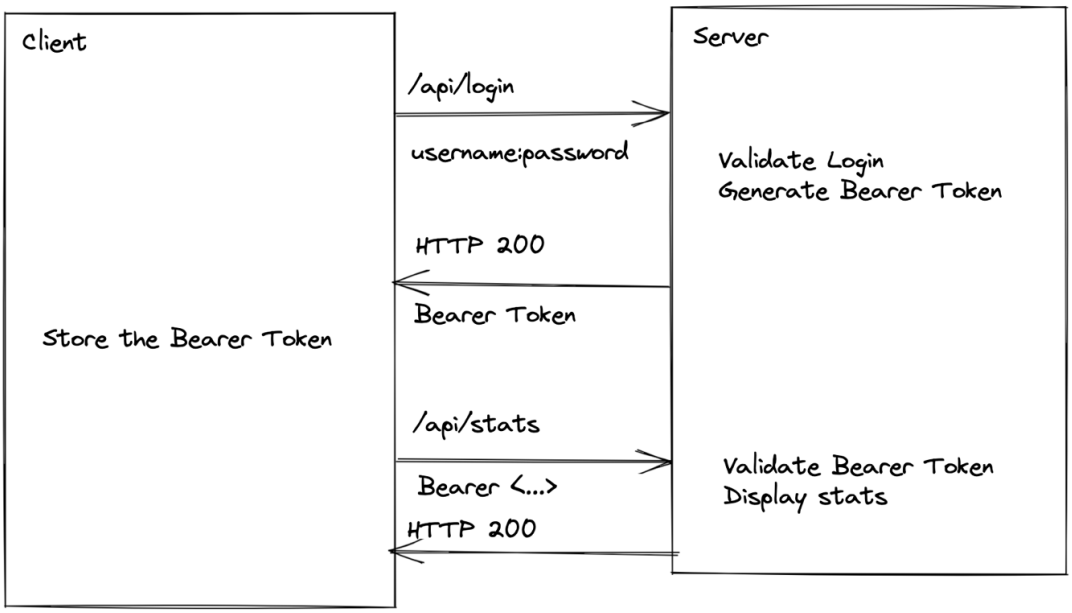
Reference : `https://beta.reactjs.org/reference/react/useState`

4.3 Step 3. Wash, Rinse, Repeat Steps 1 and 2 for the password Field

At this point, you will have two independent fields for the username and the password.

4.4 Step 4. Add a Button Component to Submit the Login Information

Add a button component (or a pressable component) which send a login request to the server. In case of success, the server returns a token, that will be used to perform authentified requests on routes that require authentification, as described by the following picture.



In this step, we are only interested by making our app sending a sound login request to the server. The generated token will be used later.

4.4.1 Remote Common Server

At first, skip the development of you local server, and use the remote KiVa server. You can work with the remote KiVa server for the entire TP if necessary. All your `axios` requests should therefore be submitted to `https://kiva.mobapp.minesparis.psl.eu/api/`, on the corresponding routes. It would be **very** nice to have this URL appearing only **once** in your entire app. *Heavy hint* : factor hardcoded strings in a file located in the `core/` folder.

If you are feeling ready to switch to your local KiVa server, we strongly recommend that you first make absolutely sure that everything is working well with the remote server.

4.4.2 Behavior and Implementation of the Login Request

The GET login route on the server is `/api/login`.

Questions :

- What does this request return ? What happens when the credentials are correct/incorrect ?
- Try the following commandline calls :
  - this call should fail because of a wrong password :

    ```
    curl -i -u kiva:mal https://kiva.mobapp.minesparis.psl.eu/api/login
    ```
  - this call should deliver a bearer token :

    ```
    curl -i -u kiva:bien https://kiva.mobapp.minesparis.psl.eu/api/login
    ```

    Answer :

    ```
    "kiva:kiva:20250104181750:254de2**********a89756"
    ```
  - with this bearer token, you are allowed to make calls to other logged-in routes of the api :

    ```
    curl -i -H "Authorization: Bearer
    ↪ kiva:kiva:20240103181750:254de2**********a89756"
    ↪ https://kiva.mobapp.minesparis.psl.eu/api/users
    ```
- In which part of the HTTP request are the credentials submitted ? How to obtain this behavior with `axios` ?

Now, make the button send the login request on `/api/login` with the credentials taken from our state variables `username` and `password`, and correctly encoded.

4.4.3 Local KiVa Server

If your local KiVa server is sufficiently stable and developed, you can start using it.

On the server side, add a dedicated GET route `/api/login` to submit the user credentials. It should have two parameters (username and password). The server should validate that a user exists with those credentials and return a bearer token if it is the case.

You must also enforce the `/api/users` route to be accessible only to authentified users (by a valid token or username/password).

4.5 Step 5. Use the Response to Give Feedback (User Experience)

If the credentials are incorrect, the server returns no bearer token and answers a `40x` code. Display an error message.

If the credentials are correct and the server's response code is `200`, `console.log` this in the app.

Then, store the bearer token in some global state (that is to say : high-level, to be shared via props to many subcomponents). We will use this token later.

Notes :

- Each of those steps can (and should be) done within a dedicated commit. Having simple commits with frequent merges makes it easier to resolve conflicts.
- Check with `git status` that newly created files and edited files have been included in the commit that you are about to make.
- It might be useful to create a dedicated component for your inputs, as this will make code re-use easier.

5 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 views. 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 a click of the submit button, send a POST request to the route `/api/users` of the server to create the user. The server should validate that a user does not already exists with this username, may validate some password strength criteria, and if everything is ok, should create the requested user.

In the front-end, if the response is correct, redirect to the Login view. If the response is incorrect, display an error message.

6 Logged-in View

Add a new view in the application for the Logged-in users. All the files should be stored in the `logggedIn/` folder.

6.1 Simple View

The goal here is simply to have a visual change when the login action is successful.

This view should display the username and password of the current logged-in account. The login action (of the `Login.react.js` component) should redirect to this view if the login action is successful.

Notes :

- You will need to be able to share the username and password (entered in the Login component of the Logged-out view) with the Logged-in view. Think about how to do it cleanly.
- The Logged-in and Logged-out views do not have to be nested one into another to perform information sharing.

6.2 List Users on Button Press

On the Logged-in simple view of the previous section, add an authenticated request to list all the users of the database. This action should be triggered by a button.

Notes :

- The server endpoint is GET on `/api/users`.
- You will need the authentication bearer token, that is sent to the server when logging in (in the Logged-out view). As for the global sharing of the username and password, it has to be somehow shared between all components.
- Think precisely on what kind of access each component needs to each state variable : read ? write ? both ?

6.3 List Users on First Render

In the Logged-in view – List Users component, add an authenticated request to list all the users in the database. This request should be triggered on the first render of the component.

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

Notes :

- `useEffect(() => fetchMyData(), [])` will call `fetchMyData()` on the first render of the component.
- `useEffect(() => fetchMyData(filter), [filter])` will call `fetchMyData(filter)` on the first render of the component and every time the `filter` state variable changes.

6.4 Logout Button

Add a way to log yourself out.

6.5 Support for Admin Users to Delete Others

Depending on whether the current user is an administrator or not, enable the deletion of users in the user list.

Notes :

- You need to identify whether the user is an admin or not during the login phase that happens in the Logged-out view of section 4.
- You also need to share this information with other components.
- Should data validation and permission checking only be on the client side ?

7 Super Extension – `useContext`

Take a look at `useContext`, and use this hook to share the credentials (username, password, authentication token, etc.) throughout the entire app.

`https://beta.reactjs.org/reference/react/useContext`