

# *P13 - Utilisez une API pour un compte utilisateur bancaire avec React*



*par Thomas RANQUE*

The logo for ArgentBank, featuring the word "ARGENT" in green and "BANK" in dark blue/black, both in a bold, sans-serif font.

## Le projet

ArgentBank est une nouvelle banque en ligne.

*En vue de l'aider à percer dans le secteur, nous sommes recrutés pour mettre en place une application à l'attention de sa clientèle.*

**Phase 1 : Authentification des utilisateurs** – Création d'une application web permettant aux clients de se connecter et de gérer leurs comptes et leur profil.

**Phase 2 : Transactions** – Il s'agirait de spécifier les endpoints d'API nécessaires pour une éventuelle deuxième mission une fois que nous aurons terminé la première.

# La structure du projet


Le projet est développé en tant qu'application React en faisant usage de l'environnement fourni par la commande *create-react-app*.

**React-router v6** est employé pour le routage, et c'est **Axios** qui sert à la récupération de données depuis l'API.

**Redux est employé** pour la gestion d'un store global.


A noter l'emploi de **Sass** pour le css.

**ARGENTBANK**Sign In




**No fees.  
No minimum deposit.  
High interest rates.**

Open a savings account with Argent Bank today!




**You are our #1 priority**

Need to talk to a representative? You can get in touch through our 24/7 chat or through a phone call in less than 5 minutes.



**More savings means higher rates**

The more you save with us, the higher your interest rate will be!



**Security you can trust**

We use top of the line encryption to make sure your data and money is always safe.

Copyright 2020 Argent Bank

# Router & Provider

## Index.js

Mise en place du provider du Store qui vient wrapper l'application.

La librairie FontAwesome fournit les icônes requises.

```
1 import { BrowserRouter, Routes, Route } from 'react-router-dom'
2 import Header from './Header'
3 import Footer from './Footer';
4 import Home from './pages/Home';
5 import Signin from './pages/Signin';
6 import Signup from './pages/Signup.js';
7 import User from './pages/User';
8 import Transactions from './pages/Transactions';
9 import Error404 from './pages/Error404';
10
11
12
13 /**
14  * The Router function returns a BrowserRouter component that contains a Header, Routes, and a Footer
15  * component
16  * @returns A BrowserRouter component with a Header, Routes, and Footer component.
17  */
18 function Router() {
19   return (
20     <BrowserRouter>
21       <Header />
22       <Routes>
23         <Route path="/" element={<Home />} />
24         <Route path="/signin" element={<Signin />} />
25         <Route path="/signup" element={<Signup />} />
26         <Route path="/user/:userId" element={<User />} />
27         <Route path="/user/:userId/transactions" element={<Transactions />} />
28         <Route path="" element={<Error404 />} />
29       </Routes>
30       <Footer />
31     </BrowserRouter>
32   );
33 }
34
35 export default Router;
```

```
1 import React from 'react';
2 import { createRoot } from 'react-dom/client';
3 import Router from './components/Router';
4 import store from './utils/store'
5 import { Provider } from 'react-redux'
6
7 import './style/index.scss'
8 import { faCircleUser, faSignOut, faArrowLeft } from '@fortawesome/free-solid-svg-icons'
9 import { library } from '@fortawesome/fontawesome-svg-core';
10
11 // Global FontAwesome icons lib
12 library.add(faCircleUser, faSignOut, faArrowLeft)
13
14 // App router with Redux store Provider
15 const container = document.getElementById('root');
16 const root = createRoot(container);
17 root.render(
18   <Provider store={store}>
19     <Router tab="home" />
20   </Provider>
21 );
```

## Router.js

La version 6 de *react-router-dom* remplace *Switch* par *Routes*.

- Etablissement des routes de l'application
- Gestion des routes dynamiques ( :userId )
- Gestion des routes erronées ( Error404 )

# Redux & Redux Toolkit

Avec Redux Toolkit, les **actions** et le **reducer** sont créés par *createSlice()*.

Le fichier regroupe aussi le **State** initial et les *Thunks* qui permettent la gestion des calls asynchrones à l'API.

Le **Store** se résume à l'import de la *Slice*.

Les **selectors** facilitent l'accès au **State**.

## Selectors

```
1 import { configureStore } from '@reduxjs/toolkit'
2 import userIdReducer from './slices/userIdSlice'
3
4 export default configureStore({
5   reducer: {
6     user: userIdReducer,
7   }
8 })
```

## Store

## userSlice

```
201 /**
202  * REDUCER and ACTIONS build with Redux Toolkit createSlice()
203  * @param {string} name - Reducer's name
204  * @param {object} initialState - Reducer's initial state
205  * @param {object} reducers - Actions creator
206  * @returns Actions & a Reducer
207  */
208 const { actions, reducer } = createSlice({
209   name: 'user',
210   initialState,
211   reducers: {
212     init: (draft) => {
213       console.log('INITIALISATION');
214       draft.status = 'void'
215       draft.info = initialState.info
216       draft.transactions = initialState.transactions
217       // Remove token from sessionStorage on logout
218       // Token should be managed by a cookie with 'HTMLOnly' parameter served from API
219       sessionStorage.removeItem('ARGENTBANK_token')
220       const oldStorage = JSON.parse(localStorage.getItem('ARGENTBANK_userInfos'))
221       localStorage.setItem('ARGENTBANK_userInfos', JSON.stringify({ email: oldStorage.email }))
222       return
223     },
224     remember: (draft, action) => { draft.rememberMe = action.payload }
225   },
226   fetching: (draft) => {
227     draft.error = null
228     if (draft.status === 'resolved') {
229       draft.status = 'updating'
230     }
231   }
232 })
```

```
import { createSlice } from '@reduxjs/toolkit'
import axios from 'axios'
import { rememberMeSelector, statusSelector } from './selectors'
```

```
// User initial state
const initialState = {
  status: 'void',
  rememberMe: localStorage.getItem('ARGENTBANK_rememberMe') === 'true' || false,
  error: null,
  infos: {
    firstName: null,
    lastName: null,
    id: null,
    email: null,
    createdAt: null,
  },
  transactions: {
    status: 'void',
    data: null,
    error: null
  }
}

/* ----- PROFILE ----- */
/**
 * Manage LOGOUT user
 * Initiate state on logout, but keeps rememberMe state
 * @returns A thunk.
 */
export function initProfile() {
  return async (dispatch, getState) => {
    const status = statusSelector(getState())
    if (status === 'connected') {
      // ----- TRANSACTIONS ----- */
      // ...
    }
  }
}

/* ----- Functions & Middlew ----- */
/* ----- Manage LOG ----- */
/**
 * Manage LOG
 * It returns
 * resolved o
 * @param {st
 * @param {st
 * @param {bo
 * @returns a
 */
export function setRememberMe(dispatch) => {
  dispatch(rememberMeSelector)
  localStorage.setItem('ARGENTBANK_rememberMe', 'true')
}
```

```
export function getUserTransactions(token) {
  console.log('FETCHING TRANSACTIONS')
  return async (dispatch) => {
    dispatch(fetchingTransactions())
    try {
      const response = await axios.post(
        'http://127.0.0.1:3001/api/v1/user/transaction',
        {},
        {
          headers: { Authorization: token }
        }
      )
      console.log('TRANSACTIONS -', response.data.body)
      const data = response.data.body
      dispatch(resolvedTransactions(data))
    } catch (error) {
      console.log('ERROR fetching transactions -', error)
      dispatch(rejectedTransactions(error))
    }
  }
}

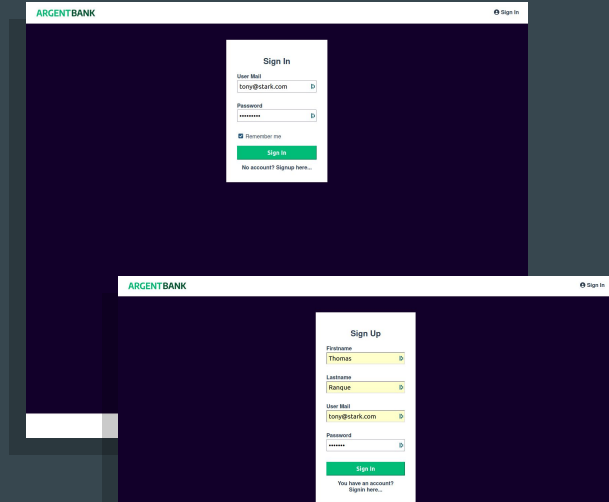
/* ----- Manage FETCHING user transaction (ID) details ----- */
/**
 * It returns a thunk that dispatches a fetching action, then makes an API call, then
 * resolved or rejected action based on the result of the API call
 * @param {string} token - The token to access the API
 * @param {string} id - The transaction ID
 * @returns A thunk
 */
export function getTransactionDetails(token, id) {
  console.log('FETCHING TRANSACTION n°$id')
  return async (dispatch) => {
    try {
      const response = await axios.post(
        'http://127.0.0.1:3001/api/v1/user/transaction/$id',
        {},
        {}
      )
    }
  }
}
```

# SignIn / SignUp

## sécurisation de l'accès

L'accès au service est sécurisé par un login( email ) et un password.

- Une fonction valide le format des inputs
- Un *JWT* est fourni en retour de `signinUser()` qui permet l'authentification à l'API.
- **getUserProfile()** récupère les infos utilisateur
- *localStorage* et *sessionStorage* stockent certaines informations en duplicata du Store, pour assurer la persistance des données en cas de reload utilisateur.



```
63 // Validate each input and sets value for email & password
64 function validateForm(type, value) {
65   const emailRegex = /^[\\w-\\.]+@[\\w-]+\\.+[\\w-]{2,4}$/
66   switch (type) {
67     case 'email':
68       setEmail(value)
69       if (!emailRegex.test(value)) {
70         emailError.current.className = 'error-msg error-show'
71         setFormValidator(false)
72         return
73       } else {
74         emailError.current.className = 'error-msg'
75       }
76       break
77     default:
78       setPassword(value)
79       if (value.length < 6) {
80         passwordError.current.className = 'error-msg error-show'
81         setFormValidator(false)
82         return
83       } else {
84         passwordError.current.className = 'error-msg'
85       }
86       break
87   }
88 }
```

En pratique, le *JWT* devrait être géré à minima par l'API via un *cookie* disposant de l'option `httpOnly` ( non accessible au JS côté utilisateur ).

ref de librairies : <https://jwt.io/>

```
26 // Check token to grant access or throw to /signin page
27 useEffect(() => {
28   if (!token) {
29     dispatch(initProfile())
30     navigate('/signin')
31   }
32   else {
33     try {
34       dispatch(getUserProfile(token))
35     } catch (error) {
36       console.log('ERROR GETTING USER DATA -', error)
37       dispatch(initProfile())
38       navigate('/signin')
39     }
40   }
41 }, [token])
42
43 // Secure userId route
44 useEffect(() => {
45   console.log('PARAMID-', userId, 'ID-', id);
46   if (userId !== id) {
47     dispatch(initProfile())
48     navigate('/signin')
49   }
50 }, [id])
```

# Logout

Le chargement de la route '/' provoque automatiquement **initProfile()** et le changement de **status** ( connected -> void ) de l'utilisateur.

Le *JWT* est retiré de *sessionStorage*.

L'utilisateur est alors obligé de s'authentifier à nouveau.

```
9  /**
10   * It returns a header with a logo, a link to the home page, and a link to the signin page if the user
11   * is not connected, and a link to the home page and a link to sign out if the user is connected
12   * @returns A header with a logo and a link to the signin page.
13   */
14  const Header = () => {
15    const connected = useSelector(state => state.user.status === 'connected')
16    const { firstName } = useSelector(state => userInfosSelector(state))
17
18    return (
19      <header className='main-nav'>
20        <Link to="/" className='main-nav-logo'>
21          <img className='main-nav-logo-image' src={logo} alt='logo' />
22        </Link>
23        {connected ? (
24          <Link to="/" className='main-nav-item'>
25            <FontAwesomeIcon className='fa fa-circle-user' icon="circle-user" />
26            {firstName}
27            <FontAwesomeIcon className='fa fa-sign-out' icon="sign-out" />
28            Sign Out
29          </Link>
30        ) : (
31          <Link to="/signin" className='main-nav-item'>
32            <FontAwesomeIcon className='fa fa-circle-user' icon="circle-user" />
33            Sign In
34          </Link>
35        )}
36      </header>
37    )
38  }
39
40  export default Header
```

```
43  /* ----- PROFILE ----- */
44
45  /**
46   * Manage LOGOUT user
47   * Initiate state on logout, but keeps rememberMe state
48   * @returns A thunk.
49   */
50  export function initProfile() {
51    return async (dispatch, getState) => {
52      const status = statusSelector(getState())
53      if (status === 'connected') {
54        console.log('DISCONNECTING - Empty User Credentials')
55        dispatch(init())
56      }
57      return
58    }
59  }
60
61  /**
62   * Manage LOGIN user
63   * It returns a thunk that dispatches a fetching action, then makes an API call, then dispatches a
64   * resolved or rejected action based on the result of the API call
65   * @param {string} email - The email address of the user
66   * @param {string} password - The password of the user
67   * @param {boolean} rememberMe
68   * @returns A thunk
69   */
70  export function signinUser(email, password, rememberMe) {
71    return async (dispatch, getState) => {
72      if (!rememberMe) {
73        // ...
74      }
75    }
76  }
```



```

52  /**
53   * It takes the form data, and updates the user's profile
54   * @param e - the event object
55   * @callback updateUserProfile - Dispatch new profile
56   */
57  function updateProfile(e) {
58    e.preventDefault()
59    closeProfileForm()
60    const values = {
61      firstName: firstName,
62      lastName: lastName,
63      email: email
64    }
65    Object.values(e.target).forEach((obj, index) => {
66      if (obj.value === undefined) {
67        return
68      }
69      if (obj.value !== "") {
70        values[Object.keys(values)[index]] = Object.values(e.target)[index].value
71      }
72    })
73    dispatch(updateUserProfile(token, values))
74  }
75
76  function closeProfileForm() {
77    profileForm.current.style.top = '-100%'
78    profileForm.current.style.opacity = '0'
79  }
80
81  function showProfileForm() {
82    profileForm.current.style.top = '0'
83    profileForm.current.style.opacity = '1'
84  }
85
86  function consultAccount(e) {
87    dispatch(getUserTransactions(token))
88    navigate('transactions')
89  }

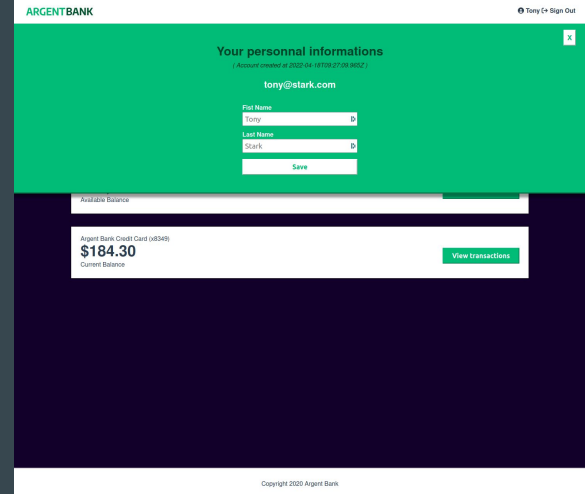
```

# User profile

## Accès & modifications

Les données récupérées à la connection au service sont accessibles sur le Dashboard par **show/closeProfileForm()**.

Un **dispatch** déclenche l'action de modification du Store à la demande de l'utilisateur ( **updateUserProfile** ).





# Modélisation de l'API

## Swagger

En vue de l'extension des capacités de l'application à la gestion des transactions, des **endpoints** supplémentaires ont été envisagés :

- GET – /user/transaction  
Consulter les transactions du mois.
- POST – /user/transaction/{id}  
Consulter les détails d'une transaction.
- PUT – /user/transaction/{id}  
Modifier les détails d'une transaction.
- DELETE – /user/transaction/{id}  
Modifier les détails d'une transaction.

Swagger Editor

```

193 - $ref: '#/definitions/Transaction'
194 -
195 - produces:
196 -   - application/json
197 -
198 - responses:
199 -   '200':
200 -     description: Transaction details updated successfully
201 -     schema:
202 -       $ref: '#/definitions/TransactionUpdateResponse'
203 -   '400':
204 -     description: Invalid Fields
205 -   '500':
206 -     description: Internal Server Error
207 -
208 - delete:
209 -   security:
210 -     - Bearer: []
211 -   tags:
212 -     - Transaction Module
213 -   summary: Transaction API
214 -   description: API for deleting a specific transaction details
215 -   parameters:
216 -     - in: header
217 -       type: string
218 -       name: Authorization
219 -       description: Attach Bearer JWT token
220 -       required: true
221 -     - in: path
222 -       name: id
223 -       type: string
224 -       description: Attach transaction ID to URL
225 -       required: true
226 -   produces:
227 -     - application/json
228 -   responses:
229 -     '200':
230 -       description: Transaction details deleted successfully
231 -       schema:
232 -         $ref: '#/definitions/TransactionUpdateResponse'
233 -     '400':
234 -       description: Invalid Fields
235 -     '500':
236 -       description: Internal Server Error
237 -
238 - securityDefinitions:
239 -   Bearer:
240 -     type: apiKey
241 -     name: Authorization
242 -     in: header
243 -
244 - definitions:
245 -   User:
246 -     properties:
247 -       email:
248 -         type: string
249 -         description: user email
250 -       password:
251 -         type: string
252 -         description: user password
253 -       firstName:
254 -         type: string
255 -         description: user first name
256 -       lastName:
257 -         type: string
258 -         description: user last name
259 -
260 -   Login:
261 -     properties:
262 -       email:
263 -         type: string
264 -         description: user email
265 -       password:
266 -         type: string
267 -         description: user password
268 -
269 -   Transaction:
270 -     properties:
271 -       type:
272 -         type: string
273 -         description: transaction type
274 -       category:
275 -         type: string
276 -         description: transaction category
277 -       notes:

```

## Bank Argent API documentation 1.1.0

[ Base URL: localhost:3001/api/v1 ]

Contains all available API endpoints in this codebase

[Terms of service](#)

Schemes

HTTP

### User Module

**POST** /user/login Login

**POST** /user/signup Signup

**POST** /user/profile User Profile API

**PUT** /user/profile User Profile API

### Transaction Module

**GET** /user/transaction Transaction API

**POST** /user/transaction/{id} Transaction API

**PUT** /user/transaction/{id} Transaction API

**DELETE** /user/transaction/{id} Transaction API

### Models

User >

Login >

# Fonctionnalités des transactions

Afin d'éprouver les méthodes et routes envisagées pour la **phase 2**, l'API et l'application ont été développées.

Il s'agit d'une proposition technique fonctionnelle qui est hors des pré-requis de l'exercice.

← Back

Argent Bank Checking (x8349)

\$2,082.79

Fleming & Co. (TS0001-0605)

**\$40**

Amount

**\$2147.79**

Available Balance

Type

Electronic

Category

Ping receiver

Notes

5 more to Davis!

June 5th, 2021

Save Details

Delete Details

Golden Sun Bakery (TS0002-0612)

**\$30**

Amount

**\$2187.79**

Available Balance

June 12th, 2021

View Details

Golden Sun Bakery (TS0003-0620)

**\$5**

Amount

**\$2217.79**

Available Balance

Type

Representation

Category

Food

Notes

Pickles & wine...

June 20th, 2021

Save Details

Delete Details

Oil factory Inc. (TS0004-0627)

**\$10**

Amount

**\$2223.79**

Available Balance

June 27th, 2021

View Details

Golden Sun Bakery (TS0005-0630)

**\$20**

Amount

**\$2233.79**

Available Balance

June 30th, 2021

View Details

# Contrôle de l'identité

## sécurisation des données

```
26 // Check token to grant access or throw to /signin page
27 useEffect(() => {
28   if (!token) {
29     dispatch(initProfile())
30     navigate('/signin')
31   }
32   else {
33     try {
34       dispatch(getUserProfile(token))
35     } catch (error) {
36       console.log('ERROR GETTING USER DATA -', error)
37       dispatch(initProfile())
38       navigate('/signin')
39     }
40   }
41 }, [token])
42
43 // Secure userId route
44 useEffect(() => {
45   console.log('PARAMID-', userId, 'ID-', id);
46   if (userId !== id) {
47     dispatch(initProfile())
48     navigate('/signin')
49   }
50 }, [id])
```

Il est important de vérifier à plusieurs niveaux la validité de l'identité de l'utilisateur, les données bancaires étant sensibles par essence.

**useEffect (1)** – Vérifie la présence du *JWT* et déconnecte le cas échéant.

**useEffect (2)** – Compare *userId* de la route à celui du Store.

# Proptypes

## sécurisation des données

```
1 import chat from '../assets/icon-chat.png'
2 import money from '../assets/icon-money.png'
3 import security from '../assets/icon-security.png'
4 import PropTypes from 'prop-types';
5
6 /**
7  * Build the 3 features on Homepage
8  * @param {object} value - Values to build a feature component
9  * @returns A React component.
10 */
11 const Feature = ({ value }) => {
12   const { type, title, text } = value
13   const imgSrc = {
14     chat: chat,
15     money: money,
16     security: security
17   }
18
19   return (
20     <div className="feature-item">
21       <img src={imgSrc[type]} alt={`${type} Icon`} className="feature-icon" />
22       <h3 className="feature-item-title">{title}</h3>
23       <p>
24         {text}
25       </p>
26     </div>
27   )
28 }
29
30 export default Feature
31
32 Feature.propTypes = {
33   value: PropTypes.objectOf(PropTypes.string)
34 }
```

Proptypes permet la **vérification du type des paramètres** fournis aux composants afin de limiter les risques de bugs, en particulier si plusieurs développeurs travaillent sur le même projet.

*Il est à noter que des solutions de typage plus robustes telles que **Flow** et **Typescript** existent.*

# DOC inline & README

*Afin de permettre une bonne maintenabilité du code, il est nécessaire de documenter clairement le projet.*

- Un **cartouche** présentant les fonctions importantes, leurs paramètres et leur retour.
- Définir le **{type}** pour chaque paramètre abordé.
- Emploi de la langue anglaise.

Le fichier **README.md** à la racine du repository GitHub est fondamental pour permettre une bonne installation de l'application et un usage approprié.

The screenshot shows a GitHub repository for 'Peanuts-83 / P13-Bank-Front'. The repository is public and has 22 commits. The file list includes 'public', 'src', '.gitignore', 'README.md', 'Ranque\_Thomas\_1\_swagger\_phas...', 'hp.png', 'hp\_vignette.png', 'made-with-create-react-app.svg', 'package-lock.json', 'package.json', 'pdf.png', 'swagger.png', 'swagger\_vignette.png', 'uses-react-router-v6.svg', and 'uses-react.svg'. The README.md file is open, showing a 'P13 - ArgentBank - Front-end application' section. The README includes a description of the application and a list of technologies used: React, React-Router V6, and Create-React-App.

```
61 /**
62  * Manage LOGIN user
63  * It returns a thunk that dispatches a fetching action, then makes an API call, then dispatches a
64  * resolved or rejected action based on the result of the API call
65  * @param {string} email - The email address of the user
66  * @param {string} password - The password of the user
67  * @param {boolean} rememberMe
68  * @returns A thunk
69  */
70 export function signInUser(email, password, rememberMe) {
71   return async (dispatch, getState) => {
72     if (!rememberMe) {
73       rememberMe = rememberMeSelector(getState())
74     }
75     dispatch(fetching())
```

# GitHub

Le projet est hébergé sur Github avec un fichier README.md explicite :

- Contexte / technos
- Installation
- Utilisation
- Paramétrage

Les commits sont réguliers et explicites.

The image displays two GitHub repository pages side-by-side. The left page is for the repository 'Peanuts-83 / P13-Bank-Front' (Public). It shows a file list with items like 'public', 'src', '.gitignore', 'README.md', and various image files. The 'README.md' is partially visible, showing a header 'P13 - ArgentBank - Front-end applicat' and a 'USES' section with 'REACT' and 'REACT-ROUTER V6'. The right page is for the repository 'Peanuts-83 / P13-Bank-API' (Public). It shows a file list with items like '.github/ISSUE\_TEMPLATE', '.vscode', 'designs', 'server', '.env', '.gitignore', 'CONTRIBUTING.md', 'README.md', 'made-with-javascript.svg', 'made-with-nodejs.svg', 'package-lock.json', 'package.json', 'swagger.yaml', and 'use-mongodb.svg'. The 'README.md' is also visible, showing a header 'Project #13 - Argent Bank API' and a description 'This codebase contains the code needed to run the backend for Argent Bank.'