01-login

En este módulo trabajaremos con una aplicación más completa, vamos a simular un portal de banca online, mostraremos un listado de movimientos de cuenta, así como un formulario de transferencia.

Vamos a trabajar a partir del proyecto `00-boilerplate` el cual tiene todo lo necesario para poder implementar los siguientes pasos.

Pasos a realizar

Lo primero sería instalar todas las dependencias que hay registradas en el fichero `package.json`. Para ello, abrimos el terminal en la carpeta donde se encuentra dicho fichero y ejecutamos el siguiente comando:

```
npm install
```

Este comando instalará tanto las dependencias que tiene nuestro proyecto de cliente, como las dependencias de servidor (carpeta `server`). Ahora ya podemos ejecutar el comando `npm start` para arrancar el servidor web:

> NOTA: Si tenemos algun problema y vemos que no se han instalado las dependencias del server, podemos abrir el terminal en la carpeta de server e instalar manualmente:
>
> `cd ../server npm install`

```
npm start
```

Comenzaremos por la página principal `login.html`. Ésta es la página de login, donde vamos a simular llamadas a servidor y validar los datos introducidos del usuario.

- Vamos a crear el fichero principal `login.js`:

./src/pages/login/login.js

```
console.log('login page');
```

- Y referenciarlo en el html:

./src/pages/login/login.html

```
...
      <h4>Está Usted en un <strong>sitio seguro</strong></h4>
    </div>
```

```
+   <script src="login.js"></script>
  </body>
</html>
```

- Vemos que ya sale en la consola el mensaje. Vamos a continuar recogiendo los datos que el usuario va a introducir en el formulario:

./src/pages/login/login.js

```
- console.log('login page');
+ import { onUpdateField, onSubmitForm } from '../../common/helpers';

+ let login = {
+   user: '',
+   password: '',
+ };

+ onUpdateField('user', event => {
+   const value = event.target.value;
+   login = { ...login, user: value };
+ });

+ onUpdateField('password', event => {
+   const value = event.target.value;
+   login = { ...login, password: value };
+ });

+ onSubmitForm('login-button', () => {
+   console.log({ login });
+ });
```

> NOTA: En la carpeta `common/helpers` tenemos todos los métodos de ayuda para poder asignar valores a los elementos html, asignar los errores de validacion, ejecutar una función cuando el usuario teclea en un elemento input, cuando hace un submit del formulario (pulsando el botón enviar), etc.

- Ya que sabemos como recoger el valor de los inputs, vamos a hacer la llamada a servidor:

./src/pages/login/login.api.js

```
import Axios from 'axios';

const url = `${process.env.BASE_API_URL}/login`;

export const isValidLogin = login =>
  Axios.post(url, login).then(({ data }) => data);
```

> NOTA: el usuario válido es `admin@email.com` y password `test`.
>
> Mirar el fichero login.middleware.js del server: `./server/src/login.middleware.js`

- Ahora toca usarlo:

./src/pages/login/login.js

```
import { onUpdateField, onSubmitForm } from '../../common/helpers';
+ import { isValidLogin } from './login.api';

...

onSubmitForm('login-button', () => {
- console.log({ login });
+ isValidLogin(login).then(isValid => {
+   console.log({ isValid });
+ });
});
```

- Actualmente, siempre que le demos al botón enviar, se hace una petición a servidor, aunque esté vacio. Vamos a realizar validaciones en cliente con la libreria fonk:

> Documentacion fonk-doc

```
npm install @lemoncode/fonk --save
```

- Simplemente necesitamos crear un fichero para las validaciones:

./src/pages/login/login.validations.js

```
import { Validators, createFormValidation } from '@lemoncode/fonk';

const validationSchema = {
  field: {
    user: [Validators.required, Validators.email],
    password: [Validators.required],
  },
};

export const formValidation = createFormValidation(validationSchema);
```

- Este `formValidation` ya nos provee de los métodos necesarios para validar:

./src/pages/login/login.js

```
```

```
import {
  onUpdateField,
  onSubmitForm,
+ onSetError,
+ onSetFormErrors,
} from '../../common/helpers';
import { isValidLogin } from './login.api';
+ import { formValidation } from './login.validations';

onUpdateField('user', event => {
  const value = event.target.value;
  login = { ...login, user: value };

+ formValidation.validateField('user', login.user).then(result => {
+   onSetError('user', result);
+ });
});

onUpdateField('password', event => {
  const value = event.target.value;
  login = { ...login, password: value };

+ formValidation.validateField('password', login.password).then(result => {
+   onSetError('password', result);
+ });
});

onSubmitForm('login-button', () => {
+ formValidation.validateForm(login).then(result => {
+   onSetFormErrors(result);
+   if (result.succeeded) {
      isValidLogin(login).then(isValid => {
        console.log({ isValid });
      });
+   }
+ });
});
```

- Podemos poner los mensajes en español:

./src/pages/login/login.validations.js

```
...

const validationSchema = {
  field: {
-   user: [Validators.required, Validators.email],
+   user: [
+     {
+       validator: Validators.required,
```

```
+       message: 'Campo requerido',
+     },
+     {
+       validator: Validators.email,
+       message: 'Email no válido',
+     },
+   ],
-   password: [Validators.required],
+   password: [
+     {
+       validator: Validators.required,
+       message: 'Campo requerido',
+     },
+   ],
  },
};
...
```

- Y por último navegamos cuando el login sea válido:

./src/pages/login/login.js

```
...
import { formValidation } from './login.validations';
+ import { history, routes } from '../../core/router';

...

+ const onNavigate = isValid => {
+   if (isValid) {
+     history.push(routes.accountList);
+   } else {
+     alert('Usuario y/o contraseña no válidos');
+   }
+ };

onSubmitForm('login-button', () => {
  formValidation.validateForm(login).then(result => {
    onSetFormErrors(result);
    if (result.succeeded) {
      isValidLogin(login).then(isValid => {
-       console.log({ isValid });
+       onNavigate(isValid);
      });
    }
  });
});
```