02 Login form

Vamos a implementar un formulario de login y conectarlo con un servicio mock de validación de usuario y clave.

Pasos

- Copiate el ejemplo anterior *01-routes y haz un _npm install*

```
npm install
```

- Tenemos una escena de login, pero comentamos que las escenas deberían tener lo mínimo:
    - Elección de layout.
    - Pods que usa.
- Así que vamos a crearnos un POD de login.

Vamos a por el componente presentacional, sólo nos preocuparemos de pintar y recibir de las props la info y los callback que hagan falta:

./src/pods/login/login.component.tsx

```
import React from 'react';

export const LoginComponent: React.FunctionComponent = () => {
  return <h2>Hello from Login Component</h2>;
};
```

Vamos a por el contenedor, esto consumirá el componente presentacional y expondra los enganches con los servicios de lógica de negocio, acceso a servicios.

./src/pods/login/login.container.tsx

```
import React from 'react';
import { LoginComponent } from './login.component';

export const LoginContainer: React.FunctionComponent = () => {
  return (
    <>
      <h1>Hello from Login Container</h1>
      <LoginComponent />
    </>
  );
};
```

- Vamos a crear un barrel en el pod de login

./src/pods/login/index.ts

```
export * from './login.container';
```

- Antes de conectarlo con la escena vamos a evitar los path relativos en los imports los "../", vamos a añadirlo a nuestro tsconfig y webpack.config

./tsconfig.json

```
    "paths": {
      "core": ["core"],
      "scenes": ["scenes"],
+      "pods": ["pods"]
    }
```

./config/wepback/base.config.json

```
    resolve: {
      alias: {
        core: helpers.resolveFromRootPath('src/core'),
        scenes: helpers.resolveFromRootPath('src/scenes'),
+       pods: helpers.resolveFromRootPath('src/pods'),
      },
```

- Y conectarlo con la escena:

./src/scenes/login.scene.tsx

```
import React from 'react';
- import { Link } from 'react-router-dom';
- import { routes } from 'core/router';
+ import { LoginContainer } from 'pods/login';

export const LoginScene: React.FC = () => {
-  return (
-    <>
-      <h1>Hello from Login Scene!</h1>
-      <Link to={routes.submoduleList}>Navigate to submodule list</Link>
-    </>
+  return <LoginContainer />;
  );
};
```

- Comprobamos que está funcionando:

```
npm start
```

- Es hora de maquetar un diálogo de login, vamos a usar *material-ui* para tener una aspecto profesional, lo instalamos así como su colección iconos:

```
npm install @material-ui/core @material-ui/icons --save
```

- Para estilar usaremos Emotion (CSS in JS), vamos a instalarlo:

```
npm install @emotion/css --save
```

- Montemos el diálgo básico de login, de momento no nos centramos demasiado en estilado (material-ui trae un montón de ejemplos en vivo, por ejemplo los del componente card: https://material-ui.com/components/cards/), ya que estamos a la espera del diseño final del creativo:

./src/pods/login/login.component.tsx

```
import React from 'react';
+ import Card from '@material-ui/core/Card';
+ import CardHeader from '@material-ui/core/CardHeader';
+ import CardContent from '@material-ui/core/CardContent';
+ import TextField from '@material-ui/core/TextField';
+ import Button from '@material-ui/core/Button';

export const LoginComponent: React.FunctionComponent = () => {
-  return <h2>Hello from Login Component</h2>;
+  return (
+  <Card>
+      <CardHeader title="Login" />
+      <CardContent>
+        <form>
+          <div style={{display: 'flex',flexDirection: 'column',
justifyContent:'center',}}>
+            <TextField label="Name" margin="normal" />
+            <TextField label="Password" type="password" margin="normal"
/>
+            <Button type="submit" variant="contained" color="primary">
+              Login
+            </Button>
+          </div>
+        </form>
```

```
+      </CardContent>
+  </Card>
+  );
};
```

- Vamos a quitar el texto de login container:

./src/pods/login/login.container.tsx

```
export const LoginContainer: React.FunctionComponent = () => {
  return (
    <>
-      <h1>Hello from Login Container</h1>
      <LoginComponent />
    </>
  );
};
```

- No esta mal el aspecto, pero se ve raro pegado a la izquierda, ¿ No podríamos centrarlo? Correcto vamos a generar un layout que centre el diálogo:

Primero lo definimos en la zona de layouts, aquí usaremos emotion y las medidas que trae el tema de Material:

Estilos

./src/layouts/centered.layout.styles.ts

```
import { css } from '@emotion/css';

export const root = css`
  display: grid;
  grid-template-columns: 1fr;
  align-items: center;
  margin-top: 2rem;
  @media (min-width: 800px) {
    justify-items: center;
  }
`;
```

> Si te fijas hemos dejado harcodead un punto de de corte de la media query, más adelante pasaremos esa información a un tema para que sea homogeneo en toda la aplicación.

Código

./src/layouts/centered.layout.tsx

```
import React from 'react';
import * as classes from './centered.layout.styles';

export const CenteredLayout: React.FunctionComponent = (props) => {
  const { children } = props;
  return <div className={classes.root}>{children}</div>;
};
```

Y vamos a crear el barrel para layout:

./src/layouts/index.ts

```
export * from './centered.layout';
```

Y añadirlo en los alias:

./tsconfig.json

```
    "paths": {
+     "layout": ["layout"],
      "core": ["core"],
      "scenes": ["scenes"],
      "pods": ["pods"]
    }
```

./config/base.config.js

```
    resolve: {
      alias: {
+       layout: helpers.resolveFromRootPath('src/layout'),
        core: helpers.resolveFromRootPath('src/core'),
        scenes: helpers.resolveFromRootPath('src/scenes'),
        pods: helpers.resolveFromRootPath('src/pods'),
      },
```

> Acuerdate de parar webpack-dev-server si esta arrancado para actualizar los cambios.

Después se lo asignamos en la escena:

./src/scenes/login.scene.tsx

```
import React from 'react';
import { LoginContainer } from 'pods/login';
+ import { CenteredLayout } from 'layout';
```

```
export const LoginScene: React.FC = () => {
-  return <LoginContainer />;
+  return (
+     <CenteredLayout>
+        <LoginContainer/>
+     </CenteredLayout>
+  )
};
```

Acabamos de ver como se definen los layouts, estos son genéricos para cualquier ventana, también podríamos definir ese layout a nivel de router (ventaja no redibujamos al cambiar de página).

- Si ejecutamos podemos ver como aparece nuestro formulario de login centrado, y responsivo:

```
npm start
```

¿Te apuntas a nuestro máster?

Si te ha gustado este ejemplo y tienes ganas de aprender Front End guiado por un grupo de profesionales ¿Por qué no te apuntas a nuestro Máster Front End Online Lemoncode? Tenemos tanto edición de convocatoria con clases en vivo, como edición continua con mentorización, para que puedas ir a tu ritmo y aprender mucho.

Y si tienes ganas de meterte una zambullida en el mundo *devops* apuntate nuestro Bootcamp devops online Lemoncode