**Arquitectura proyecto Incremental**

**Contenido carpetas src:**

* **Assets:** contiene ficheros gráficos como: ico, svg, png.
* **Transversal:** es la capa que contiene funcionalidades que se puede usar en el resto de capas del proyecto.
* **Utilities:** contiene funciones útiles que se pueden reaprovechar.
* **Resto carpetas… users, Project, roles, accesTarget**: cada una contiene el código de un tipo de objeto, al crear una carpeta de este tipo, la forma de averiguar el nombre de la carpeta es preguntarse “quien eres”.

**Contenido carpetas src/transversal:**

* **Cache:** contiene el código que hace funcionar la cache del proyecto.
* **Context:** contiene el código que hace funcionar el context (el equivalente al redux).

**Nota:** El fichero contextStateConfig.js contendrá la carga de todos los reduces creados para usar a nivel global en todo el proyecto. El fichero contextActionsGeneric.js contendrá las acciones a usar en todos los reduces creados para usar a nivel global en todo el proyecto.

* **Error:** contiene
  + el gestor de erros que se encargara de mostrar un toast por pantalla si se ha producido un error.
  + El error Boundary que devolverá una pantalla de error si se ha producido un fallo en el renderizado.
* **Language:** contiene el código que hace funcionar la traducción de la web y los mensajes de error.

**Nota:** Las traducciones están en el fichero translations.js

* **Routes:** contiene el código que hace funcionar la gestión de rutas de la web y también contiene todas las rutas de las páginas.
* **Fichero isAuthenticated.js:** contiene el código que verifica si estas autentificado en la web.

**Contenido carpetas src/utilities:**

* **allLayers:** contiene funciones o componentes útiles que se usaran en varias capas del proyecto.
* **application:** contiene funciones o componentes útiles que se usaran solo en la capa de application.
* **domain:** contiene funciones o componentes útiles que se usaran solo en la capa de domain.
* **infrastructure:** contiene funciones o componentes útiles que se usaran solo en la capa de infrastructure.
* **interface:** contiene funciones o componentes útiles que se usaran solo en la capa de notainterface.

**Nota:** si en algún momento del desarrollo una función o componente que está ubicada en una de las capas: application, domain, infrastructure o interface, pasa a usarse en varias capas, la función o componente deberá moverse a la carpeta allLayers y refactorizar cambiando las rutas afectadas.

**Contenido carpetas src/allLayers:**

* **Fichero adaptersAllLayers.js:** contiene funciones de adaptación de datos que se usaran en varios tipos de objeto/casosUso del proyecto ej: users, projects.
* **Fichero utilsAllLayers.js:** contiene funciones o componentes útiles que se usaran en varias capas del proyecto.
* **Fichero createResource.js:** contiene el código para poder crear un componente Suspense, se ha exportado del proyecto TwinXWeb (evaluar si es necesario este fichero ya que el suspense ya es estable en esta versión de node).
* **Nota:** si en algún momento del desarrollo una función o componente útil es demasiado grande o especial se deberá crear un fichero propio, como en el caso del fichero createResource.js.

**Contenido carpetas src/users (tomada esta carpeta como ejemplo para el resto que se deban crear):**

* **Adapters:** contiene funciones que se usan para adaptar/formatear datos como por ej:
  + Función para ordenar json por nombre campo.
  + Función para ordenar array.
  + Función para eliminar de json un campo.
  + Función para agrupar datos de un json o array.

**Nota:** la llamada a las funciones adapter se harán desde las carpetas “controllers” o “userCases”. Lo usual será adaptar datos obtenidos de la carpeta “crud” y que:

* Vienen de la interface y van a infraestructura para guardarse en bds.
* Vienen de infraestructura y van a interface para verse en pantalla.

En caso de que las funciones de adaptación no sean para usar únicamente por ej: en “src/users” se pasarían a “src/utilities/adaptersAllLayers.js

* **Context:** contiene el código encargado de almacenar datos en el contexto, ej: de “src/users” contendrá solo código para almacenar en users.
* **Infrastructure:** contiene el código encargado obtener datos de servicios externos como: api de bds, micro servicios, correo… etc.
* **Domain:** contiene el código encargado de ejecutar otros casos de uso diferentes a los de la carpeta users.

Ej: al dar de alta un usuario nuevo se debe de crear una tarjeta de acceso (para ello se llamarán a las funciones necesarias de la carpeta accesTarget para realizar dicho proceso).

* **Application:** contiene el código encargado de ejecutar los casos de uso del tipo de objeto en el que estamos, por ej: estando en “src/users” tendremos los casos “userLogin” y “userRegister”.

**Interface:** contiene el código encargado de generar el html de users (SOLO CODIGO PARA GENERAR HTML DE USERS).

**Contenido carpetas src/users/context:**

* **~~contextActionsUser.js:~~** ~~contiene funciones de acciones específicas de usuario que se llamaran para guardar datos en el reducer del caso de uso que tiene por nombre la carpeta ej: users.~~

**Nota:** no ha de ser necesario crear este fichero “**contextActionsUser.js”** ya que con las acciones del fichero “src/transversal/context/contextActionsGeneric.js” ha de ser suficiente para todas las acciones que se necesiten usar en el proyecto.

* **contextReducerUser.js:** contiene función con acciones de ej: users, se ejecutará una de las acciones para modificar el context ej: userLogin.
* **~~contextUser.js:~~** ~~contiene las funciones de dispatch de ej: users. Estas funciones se ejecutarán des de “aplication/controller” o en “domain/userCases”.~~

**Nota:** los ficheros de esta carpeta solo se usarán dentro de las capas application y domain.

**Contenido carpetas src/users/interface:**

* **css:** contiene ficheros con los estilos ccs únicos para los layouts de la carpeta “layouts”. Los ficheros contenidos dentro de la carpeta serán:
  1. Un fichero genérico de estilos que se usaran para cualquiera de los ficheros contenidos en la carpeta “layouts”.
  2. Un fichero de estilos único para un fichero de la carpeta “layout”.
* **hooksView:** contiene funciones o componentes que solo tendrán código html, para ejecutar en los layouts ej:

|  |  |  |
| --- | --- | --- |
| **Correcto** | **Incorrecto** | **Incorrecto** |
| function DropDown(items) {  return (  <select>  {items.map(item => (  <option  key={item.value}  value={item.value}  >  {item.label}  </option>  ))}  </select>  );  } | function DropDown() {  const [items] = React.useState([  {label: "Jedi",  value: "Luke"},  {label: "Robot",  value: "C-3PO"}  ]);  return (  <select>  {items.map(item => (  <option  key={item.value}  value={item.value}  >  {item.label}  </option>  ))}  </select>  );  } | Function async DropDown() {  const items = await axios.get("items");  return (  <select>  {items.map(item => (  <option  key={item.value}  value={item.value}  >  {item.label}  </option>  ))}  </select>  );  } |

**Nota:** estas funciones o componentes solo serán llamadas desde los layouts.

* **hooksCode:** contiene funciones o componentes que solo tendrán código que no sea html, para ser llamadas desde la carpeta “routes”, ej: en hooksCodeInUserLogin:
  1. se obtiene de la base de datos los datos para ser mostrados en el layoutUserLogin.js.
  2. se llama a layoutUserLogin.js pasándole los datos recuperados de la base de datos.
* **layout:** contiene los layous que se visualizarán en la web y los ficheros que contengan dependerán de la carpeta raíz, ej: users tendrá, layoutUserLogin.js y layoutUserRegister.js

**Nota:** un fichero de layout contendrá una única función y esa función solo contendrá código html y useState para almacenar el estado local de la función de layout.

Los valores globales del context los recibirá la función del fichero layout por parámetros de entrada.

Los valores locales de estado se podrán recibir desde las funciones que ejecutan los objetos de la web o las acciones que se ejecutan en la web.

Video de interés: <https://youtu.be/6h9S_IhHdZI>

**Contenido carpetas src/users/application:**

* **controllers:** existirá solo un fichero controller por layout, contendrá las funciones que serán llamadas desde el fichero layout al pulsar sobre un objeto o producirse un evento ej: btnLogin.

**Nota:** las funciones aquí creadas serán como orquestadores, encargadas de llamar a:

* 1. Recoger los datos en formato json de “application/crud”.
  2. Recoger los datos en formato json o estados de resultado de “domain/userCases”.
  3. Adaptar los datos llamando a las funciones de “adapters”.
  4. Guardar los datos en el context llamando a las funciones de “context”.
  5. Retornar los datos y/o estado obtenidos al layout.
* **crud:** contiene funciones que llamaran a las funciones ubicadas en la capa de infraestructura de: crear, leer, modificar, eliminar.

**Nota:** las funciones se llamarán desde la carpeta “controllers”. Las funciones devolverán los datos en formato json y NO harán ningún proceso que no sea relacionado con trabajar con la bds como:

* 1. Guardar en el context.
  2. Guardar en el state.
  3. Generar html.
  4. Adaptar datos.
  5. Hacer llamadas a otras funciones que estén fuera de la carpeta crud.
* **hooksCode:** contiene funciones o componentes que solo tendrán código que no sea html, para ser llamadas desde la carpeta “controllers”.

**Nota:** estas funciones NO tendrán código para almacenar datos en:

* 1. context.
  2. State (los estados se guardan en el layout).
  3. También todas las funciones de crud: crear, leer, modificar, eliminar.

**Contenido carpetas src/users/domain:**

* **userCases:** los ficheros de esta carpeta tendrán una única función que será la de orquestadora, encargada de llamar a las funciones necesarias ej: al dar de alta un usuario es necesario crearle una tarjeta de acceso al edificio.
  1. estamos en la carpeta “src/user/domain/userCases/caseAccesTargetCreate.js”.
  2. desde caseAccesTargetCreate.js llamamos a las funciones necesarias para el proceso existentes en “src/accesTarget/…” o “src/users/adapters”.
  3. desde caseAccesTargetCreate.js llamamos a las funciones necesarias para el proceso que no existen y hemos tenido que crear en src/user/domain/[crud/hooksCode/state].
  4. desde caseAccesTargetCreate.js llamamos a otro case en la misma carpeta como: de “src/user/domain/userCases/caseAccesTargetCreate.js” a “src/user/domain/userCases/caseProject.js”.
  5. flujos de código correctos:

|  |  |
| --- | --- |
| **Correcto** | **Incorrecto** |
| “src/users/application/controllers/controllerUserLogin.js” 🡪  ”src/ user/domain/userCases/caseAccesTargetCreate.js” 🡪  “src/accesTerget/…” | “src/users/application/controllers/ controllerUserLogin.js” 🡪  “src/accesTerget/…” |
| “src/users/application/controllers/ controllerUserLogin.js” 🡪  ”src/ user/domain/userCases/caseProjectsUser.js” 🡪  “src/projects/…” | “src/users/…[sitios diferentes a un userCase de users]” 🡪  “src/accesTerget/…” |
| “src/users/application/controllers/ controllerUserLogin.js” 🡪  ”src/ user/domain/userCases/caseAccesTargetCreate.js” 🡪  ”src/ user/domain/userCases/caseProjectsUser.js” 🡪  “src/projects/…” |  |

* **Nota:** en caso de tener que guardar datos en el contexto se subirán los datos a la función de la carpeta “users/application/controllers” que ha llamado al caso de uso y desde allí se llamara a la función de context para guardar.
* **crud:** contiene funciones que llamaran a las funciones ubicadas en la capa de infraestructura de: crear, leer, modificar, eliminar pero que están relacionadas con los casos de uso de la carpeta “userCases” y que no existen.

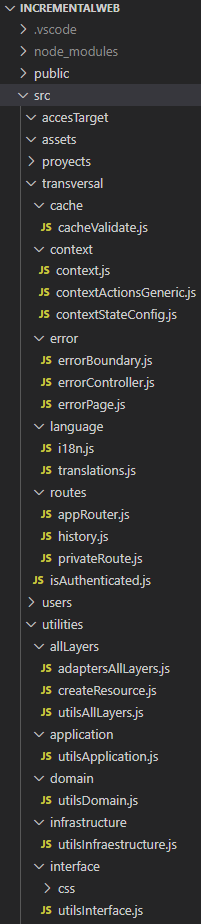
**Nota:** las funciones crud creadas aquí NO se pondrán encontrar en la carpeta crud de otros casos de usos.

* **states:** las funciones de esta carpeta guardarían el estado usando useState o useReducer para utilizarlas en los ficheros de la carpeta UserCase.

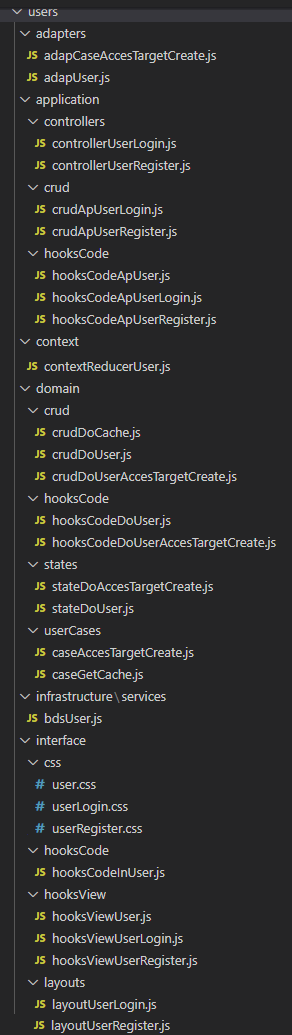
**Contenido carpetas src/users/infrastructure:**

* **services:** contiene un fichero con las funciones de uso con la bds ej: si estamos en “src/users” solo contendría las funciones que usan la tabla users.

Captura árbol proyecto estructura general:



Captura árbol proyecto estructura tipo objeto:



**Nota:** recordar que los archivos que su nombre termina igual a la carpeta principal de la raíz de la estructura son archivos que contienen funciones genéricas y el uso de sus funciones no está limitado a un caso de uso concreto ej:

* Carpeta raíz: users.
* Archivo con funciones genérico: “usr/application/hooksCode/hooksCodeApUser.js”.
* Archivos con funciones no genéricas: “usr/application/hooksCode/[hooksCodeApUserLogin.js, hooksCodeApUserRegister.js]”.

Teniendo en cuenta lo anterior mente dicho, es posible que en algún momento del desarrollo se tenga que refactorizar una función teniendo que moverla de un archivo no genérico a uno genérico.