Documentación arquitectura front

Documento para explicar a las personas implicadas en el desarrollo de front como se ha de trabajar para que no haya problemas en el desarrollo de la aplicación ni en la distribución de las librerías que van a componer nuestro producto.

INSTALACIÓN

Requisitos:

- Node 8.12.0 o superior
- npm 6.4.1 p superior
- SVN (cualquier cliente svn con la linea de comando instalada)

Al bajar nuestro proyecto del respositorio, debemos de ir al directorio raiz del proyecto y lanzar un install

```
>npm install
```

Para bajar los ejemplos de los mocks hacemos uso del siguiente script

```
>npm run checkout-raml
```

Nos pedirá nuestro usuario y pass del SVN.

Arrancaremos nuestra aplicación:

```
>npm run start
```

Tendremos accesible nuestra app en localhost:4200 por defecto.

MANERA DE TRABAJAR DESARROLLANDO EL PRODUCTO

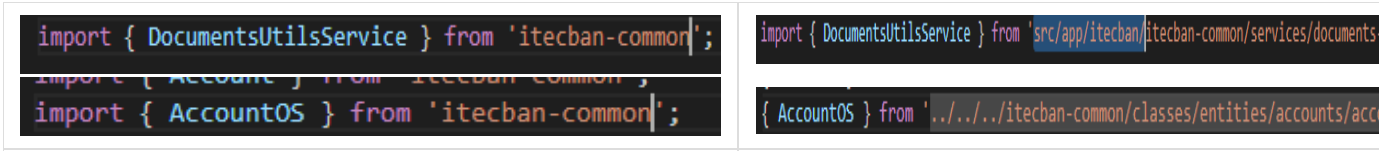
La idea es trabajar en un producto que va a estar compuesto por un grupo de librerías que dotaran de funcionalidad a nuestra aplicación. De esta forma, para cierto cliente se cargarían las librerías que fueran necesarias (ej: common, accounts, alerts,...) y se eliminar la referencia a las que no se usen (ej: directdebits, confirming,....)

Para simular estar trabajando con librerías sobre nuestra aplicación, vamos a trabajar con declarando los paths que van a componer nuestra app y dandoles el nombre de la librería que se va a utilizar. Todo esto se indica en el tsconfig.ts de nuestra aplicación principal.

```
"paths": {
  "itecban-environment": [
    "./src/app/itecban/itecban-environment"
  ],
  "itecban-accounts": [
    "./src/app/itecban/itecban-accounts"
  ],
  "itecban-directdebits": [
    "./src/app/itecban/itecban-directdebits"
  ],
  "itecban-common": [
    "./src/app/itecban/itecban-common"
  ],
```

IMPORTANTE NO USAR NUNCA REFERENCIAS ENTRE LIBRERIAS ACCEDIENDO A ELLAS POR RUTA RELATIVA A NUESTRO PROYECTO, SIEMPRE SE HA DE HACER ACCEDIENDO POR EL NOMBRE DE NUESTRA LIBRERIA

BIEN REFERENCIADO	MAL REFERENCIADO

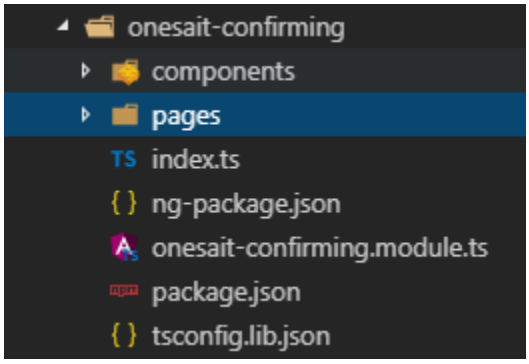


Si trabajamos de esta manera, para la aplicación es como si estuviéramos utilizando librerías bajadas del node, que es como se trabajaría para un cliente específico que hiciera uso de nuestro producto bancario.

CREACIÓN Y COMPOSICIÓN DE MODULO

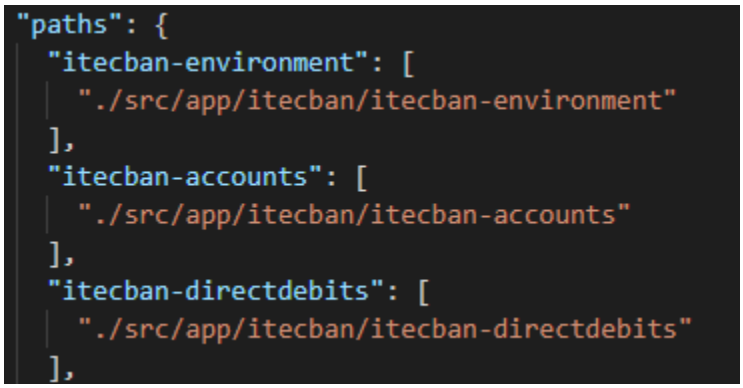
Vamos a ver como sería la manera de generar un nuevo módulo y gestionar los elementos que lo componen.

Se declaran los ficheros, index.ts, ng-package.json, librería.module.ts, package.json y tsconfig.lib.json en la raíz del directorio que va contendrá. Por poner un ejemplo:



Una vez tenemos estos ficheros, y hemos modificado lo necesario dentro de ellos (cogiendo como ejemplo otro módulo ya hecho) y modificando los datos que sean necesarios por el nombre de nuestro nuevo módulo.

A continuación habría que declarar nuestro nuevo módulo como un nuevo path para nuestra aplicación y así poder hacer uso de el como si fuera una librería externa, esto lo marcamos en el tsconfig.ts



Comenzaríamos a trabajar sobre los componentes, servicios, etc que darán funcionalidad a nuestro nuevo módulo. Se declararán, importarán y proveerán de manera normal en el fichero my-library.module.ts

```
@NgModule({
  imports: [
    CommonModule,
    OnesaitCoreModule,
    TranslateModule,
  ],
  declarations: [
    ConfirmingPageComponent,
    IssueOrdersComponent,
    OrdersRemittancesComponent,
    AdvancesComponent,
    RequestAdvanceComponent,
    ContractConditionsComponent
  ],
  exports:[
    ConfirmingPageComponent
  ],
  providers: [
    TranslatePipe
  ]
})
export class OnesaitConfirmingModule { }
```

A su vez, en el fichero index.ts hacemos la exportación de nuestros componentes, servicios, clases, etc para que puedan ser utilizados de manera externa otras librerías. Comenzando siempre por la exportación del propio módulo.

```
export { OnesaitConfirmingModule } from './onesait-confirming.module';

export { ConfirmingPageComponent } from './pages/confirming-page/confirming-page.component';

export { AdvancesComponent } from './components/advances/advances.component';
export { ContractConditionsComponent } from './components/contract-conditions/contract-conditio
export { RequestAdvanceComponent } from './components/request-advance/request-advance.component
export { IssueOrdersComponent } from './components/issue-orders/issue-orders.component';
export { OrdersRemittancesComponent } from './components/orders-remittances/orders-remittances.
```

Con esto conseguimos que nuestros elementos sean accesibles de manera directa al hacer un import de nuestra librería, como ejemplo:

```
import { UtilsService } from 'onesait-core';
import { RequestService } from 'onesait-core';
```

HACEMOS INCAPIÉ EN QUE PARA HACER USO DE ELEMENTOS DE OTRA LIBRERÍA HAY QUE HACERLO DE ESTA MANERA

Para poder hacer uso de la compilación con ng-packagr, debemos de dar de alta nuestro nuevo módulo como parte de un sub-proyecto de nuestra aplicación. Para ello en el angular.json tenemos que declarar lo siguiente:

```
"onesait-confirming": {
  "root": "src/app/onesait/onesait-confirming",
  "sourceRoot": "src/app/onesait/onesait-confirming",
  "projectType": "library",
  "prefix": "lib",
  "architect": {
    "build": {
      "builder": "@angular-devkit/build-ng-packagr:build",
      "options": {
        "tsConfig": "src/app/onesait/onesait-confirming/tsconfig.lib.json",
        "project": "src/app/onesait/onesait-confirming/ng-package.json"
      }
    }
  }
},
```

GENERACIÓN DE LIBRERIAS

En este apartado vamos a explicar como se genera la distribución de nuestras librerías. En el package.json de nuestra aplicación tenemos unas tareas declaradas para generar nuestras librerías.

```
    "packagr-libs": "node ./hooks/packagr-libs.js",
```

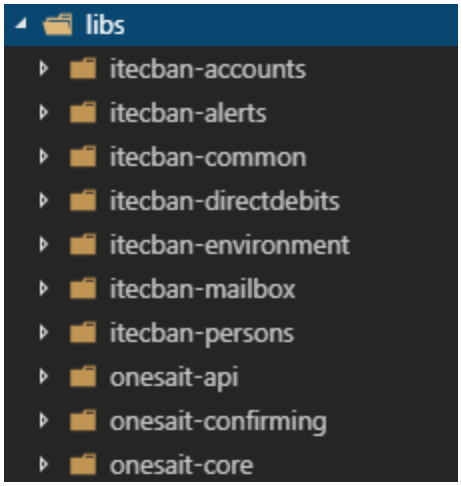
Haciendo uso de ng-packagr, vamos a ir compilando cada una de nuestros módulos, iremos generándolos por orden de dependencia en función de si un módulo requiere de otro. Esto es así, ya que no se ha podido indicar mediante el path en el tsconfig.lib.ts o en el ng-package donde ha de resolver la dependencia al solicitar otro módulo, por ejemplo onesait-core. Para resolver esto, lo que hacemos es ir compilando cada uno de ellos y se alojan en el node_modules temporalmente, para que cuando haya un referencia a una de nuestras librerías la resuelva cogiendo la compilación de nuestro módulo.

La versión con la que se van a generar son la versión que se indica dentro de nuestro package.json de la app, se hace un reemplazo dentro de cada package.json especifico de cada módulo.

En nuestro fichero packagr-libs, en la variable modules, declararemos el orden que vamos a seguir para la compilación, teniendo en cuenta las referencias internas que hay entre los módulos.

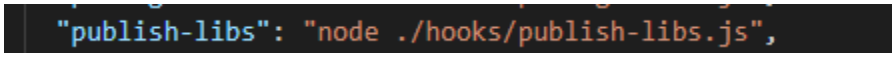
```
let modules = [
    {origin:"itecban\\itecban-environment", name:"itecban-environment"},
    {origin:"onesait\\onesait-api", name:"onesait-api"},
    {origin:"onesait\\onesait-core", name:"onesait-core"},
    {origin:"itecban\\itecban-common", name:"itecban-common"},
    {origin:"itecban\\itecban-alerts", name:"itecban-alerts"},
    {origin:"itecban\\itecban-mailbox", name:"itecban-mailbox"},
    {origin:"itecban\\itecban-persons", name:"itecban-persons"},
    {origin:"itecban\\itecban-directdebits", name:"itecban-directdebits"},
    {origin:"itecban\\itecban-accounts", name:"itecban-accounts"},
    {origin:"onesait\\onesait-confirming", name:"onesait-confirming"}
];
```

Cuando finaliza la compilación de todo nuestro código, se crea un directorio libs donde se deja toda la generación.



PUBLICACIÓN DE LIBRERIAS

La idea es que una vez generadas las librerías con una versión específica se publiquen a un node o a un nexus, tenemos una tarea en el package json para realizar esto.



Este publish esta montado para distribuirlas en un Nexus, pero aun esta pendiente de configurarse sobre el Nexus de mercado.

Se puede modificar todo lo necesario y hacer que despliegue también sobre un Git.

MANERA DE TRABAJAR HACIENDO USO DE LAS LIBRERIAS DEL PRODUCTO

Con las librerías publicadas, la idea sería hacer referencia en el package.json a nuestras librerías y a la versión específica sobre la que vamos a trabajar. Igual que se hace con cualquier otra librería que se baja haciendo uso del npm install.

PENDIENTE DE DEFINIR