

ENTREGABLE TÉCNICO

El siguiente documento tiene como objetivo realizar una entrega técnica de todo lo desarrollado en la prueba.

Base de Datos

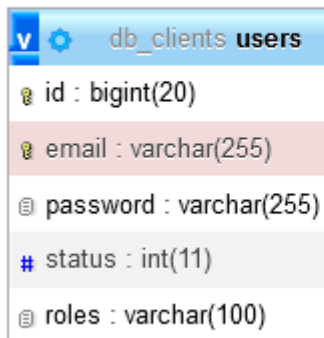
Tecnologías:

- MySQL

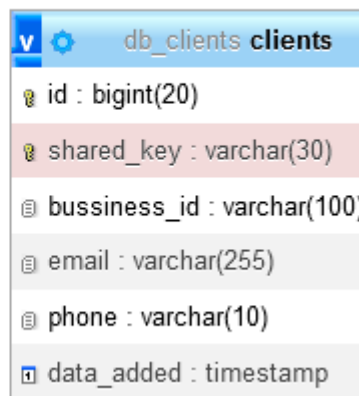
Descripción:

Se creó una base de datos sencilla con dos tablas, una donde estarían almacenados los usuarios del sistema y la otra donde se almacenarían los clientes con los que se haga persistencia.

Evidencias:



db_clients users
id : bigint(20)
email : varchar(255)
password : varchar(255)
status : int(11)
roles : varchar(100)



db_clients clients
id : bigint(20)
shared_key : varchar(30)
bussiness_id : varchar(100)
email : varchar(255)
phone : varchar(10)
data_added : timestamp

Backend:

Tecnologías:

- Java17
- Spring Boot 3.2.3
- Spring WebFlux
- Spring Data R2DBC
- Spring Security
- JWT
- DevTools
- MySQL Connector

- Lombok
- Programación Reactiva
- Programación Funcional
- RxJava2
- OpenAPI Swagger
- Jakarta
- Maven
- ModelMapper
- OpenCSV
- Clean Code
- CamelCase

Descripción:

Se desarrollaron 2 Api's o endPoints, el primero es el correspondiente al login del usuario con el cual se implementó niveles de seguridad con Spring Security y JWT. El segundo es el correspondiente a todo el CRUD de clientes solicitado dentro del cual se desarrolló lo siguiente:

- Listar Clientes
- Buscar Clientes por SharedKey
- Búsqueda Avanzada de Clientes
- Creación de Clientes
- Edición de Clientes
- Eliminación de Clientes
- Exportar Clientes a CSV

Para el desarrollo de todo el Backend se trabajó con programación reactiva y funcional, con el objetivo de obtener en lo mayor posible, las ventajas de la reactividad en la Api Rest. Se trabajaron los DTO junto con el ModelMapper para la transferencia de los datos. Se implementó un CustomExceptions para las excepciones personalizadas. Se implementaron Logs de la aplicación con Logger con descripciones muy detalladas. Se documentaron los endPoints y/o servicios haciendo uso del OpenAPI Swagger, la ruta local de la documentación es [http://localhost:9090/v1/clients-crud/webjars/swagger-ui/index.html#/. Se trabajó la persistencia con Spring Data R2DBC. Se usó Postman para las pruebas de los servicios Rest.](http://localhost:9090/v1/clients-crud/webjars/swagger-ui/index.html#/)

Evidencias:

Servers

http://localhost:9090 - Generated server url

Clients

GET /api/v1/clients/advanced-search/

POST /api/v1/clients/created/

DELETE /api/v1/clients/deleted/

GET /api/v1/clients/list/

PUT /api/v1/clients/modified/

GET /api/v1/clients/search/

Users

POST /api/v1/users/login/

Schemas

AdvancedFilterDTO >

ClientsDTO >

Clients >

LoginDTO >

TokenDTO >

```
clients [boot] [devtools]
├── src/main/java
│   ├── com.project.api.clients
│   ├── com.project.api.clients.security
│   ├── com.project.api.clients.security.config
│   ├── com.project.api.clients.security.constants
│   ├── com.project.api.clients.security.dto
│   ├── com.project.api.clients.security.entities
│   ├── com.project.api.clients.security.exceptions
│   ├── com.project.api.clients.security.handler
│   ├── com.project.api.clients.security.jwt
│   ├── com.project.api.clients.security.repositories
│   ├── com.project.api.clients.security.routes
│   ├── com.project.api.clients.security.services
│   └── com.project.api.clients.security.utilities
├── src/main/resources
├── src/test/java
├── JRE System Library [JavaSE-17]
├── Maven Dependencies
├── target/generated-sources/annotations
├── target/generated-test-sources/test-annotations
├── src
├── target
└── ...
```

Boot Dashboard

Type tags, projects, or working set names to match (incl. * and ? wildcards)

- auth [devtools]
- clients [devtools] [:9090]

Frontend:

Tecnologías:

- Angular17
- TypeScript
- JavaScript
- PrimeNG
- PrimeIcons
- PrimeFlex
- Sass
- HTML5
- RxJS
- File-Saver
- CompoDoc
- Formularios Reactivos
- Programación Reactiva y Funcional
- Directivas, Pipes, Componentes, Módulos, Interceptores, Guardianes, Servicios e Interfaces
- Suscripciones y Observables
- Atomic Web Design
- Responsive Web Design
- Clean Code
- CamelCase

Descripción:

Se desarrollo el proyecto con una modularización bien estructurada, siendo la siguiente:

- Atomic Design -> Módulo que hace referencia a la atomicidad de los componentes
 - Atoms -> Carpeta donde se desarrollaron los componentes tipo átomos o individuales (Loader, SideBar y TopBar).
 - Molecules -> Carpeta donde se desarrollaron los componentes compuestos por varios átomos (FormAdvancedSearch, ModalNewModified y TableCrud).
- Config -> Módulo que hace referencia a la configuración general y de seguridad de la aplicación, por lo general donde se encuentran los interceptores y guardianes.
 - Guards -> Carpeta donde van los guardianes de la aplicación encargados de la seguridad de las rutas (ProtectedGuard).

- Interceptors -> Carpeta donde van los interceptores de la aplicación encargados de interceptar las solicitudes a enviar al Backend y automatizar tareas en base a ellas (RequestInterceptor).
- Core -> Módulo en donde se encuentra el core de la aplicación, es decir, las interfaces y los servicios que nos proveerán la data.
 - Interfaces -> Carpeta donde van las interfaces representando la información manipulada, es decir, haciendo referencia a los DTO del Backend (ClientsInterfaces y UsersInterfaces).
 - Services -> Carpeta donde van los servicios que nos proveerán la data o información a manipular en la aplicación (ClientsServices, LoaderServices y UsersServices).
- Pages -> Módulo en donde se encuentran las rutas o páginas de la aplicación.
 - Crud -> Componente que hace referencia a la página o pantalla del crud o administración de clientes.
 - Login -> Componente que hace referencia a la página o pantalla del login del sistema.
- Shared -> Módulo que contiene recursos compartidos entre todos los módulos de la aplicación (Pipes, Directives etc).

Se implementó seguridad en la ruta /clients haciendo uso de un Guard, adicional, se usó el sessionStorage para almacenar el token recibido después del login.

Se implementó documentación técnica del proyecto Angular haciendo uso de la tecnología CompoDoc.

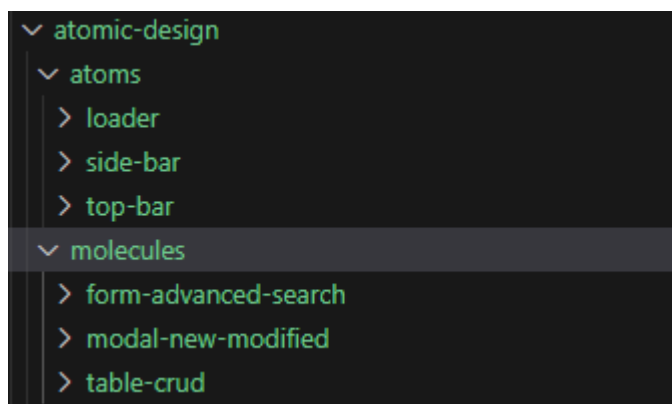
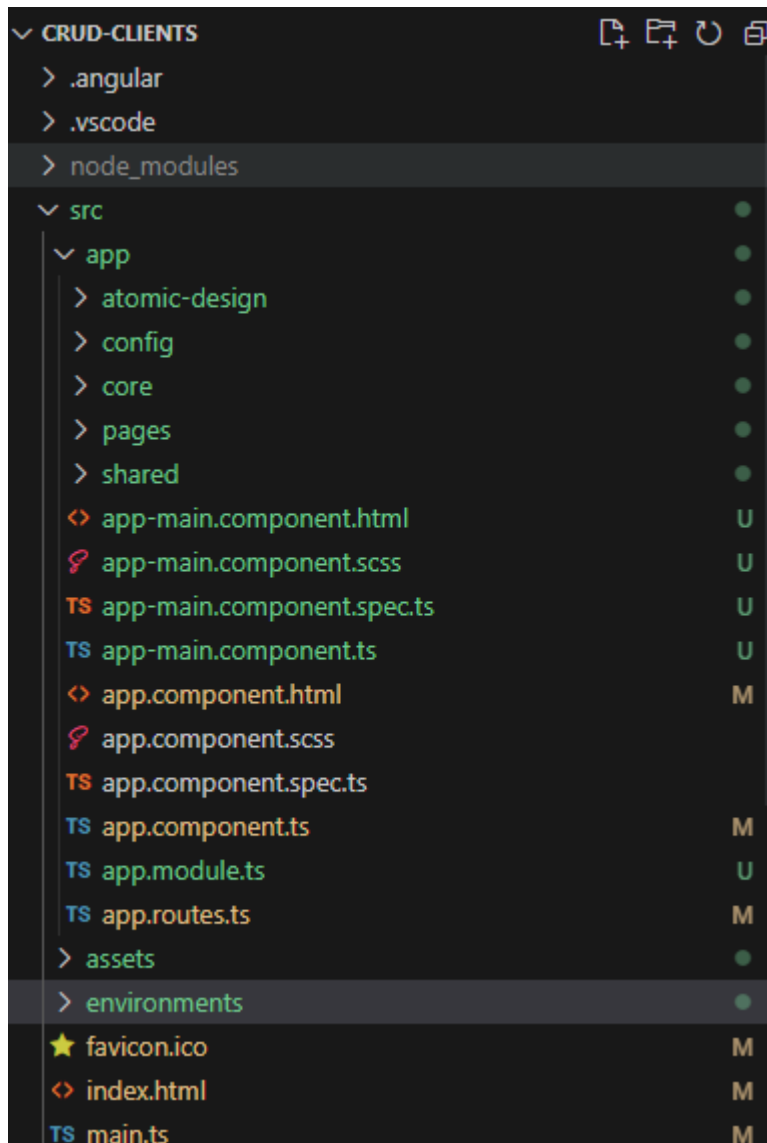
Se trabajaron formularios reactivos haciendo uso de FormBuilder.

Se trabajó con programación reactiva y funcional con el objetivo de obtener en lo mayor posible, sus ventajas.

Se implementó la creación y destrucción de suscripciones para liberar memoria.

Se implementó el lazy loading o carga perezosa.

Evidencias:



```

  ✓ config
    ✓ guards
      TS protected.guard.spec.ts
      TS protected.guard.ts
    ✓ interceptors
      TS request.interceptor.spec.ts
      TS request.interceptor.ts

```

```

  ✓ core
    ✓ interfaces
      TS clients.interfaces.ts
      TS users.interfaces.ts
    ✓ services
      TS clients.service.spec.ts
      TS clients.service.ts
      TS loader.service.spec.ts
      TS loader.service.ts
      TS users.service.spec.ts
      TS users.service.ts

```

```

  ✓ pages
    > crud
    > login
    TS pages-routing.module.ts
    TS pages.module.ts

```

```

  ✓ shared
    ✓ pipes
      TS date.pipe.spec.ts
      TS date.pipe.ts
      TS phone.pipe.spec.ts
      TS phone.pipe.ts
      TS shared.module.ts

```

```
TS app.routes.ts M X
src > app > TS app.routes.ts > routes > children
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { AppMainComponent } from './app-main.component';
4 import { LoginComponent } from './pages/login/login.component';
5 import { AuthGuard } from './config/guards/protected.guard';
6 const routes: Routes = [
7   {
8     path: 'clients', component: AppMainComponent,
9     children:[
10      {
11        path:'', loadChildren:()=>import('./pages/pages.module').then(m=>m.PagesModule),
12        canActivate: [AuthGuard]
13      }
14    ]
15  },
16  {
17    path: 'login',
18    component: LoginComponent
19  },
20  {
21    path: '',
22    component: LoginComponent
23  }
24 ]
25
26 @NgModule({
27   imports: [RouterModule.forRoot(routes)],
28   exports: [RouterModule],
29 })
30 export class AppRoutingModule {}
31
```

