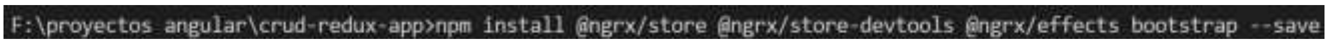
Instalando las dependencias (store, devtools, bootstrap)

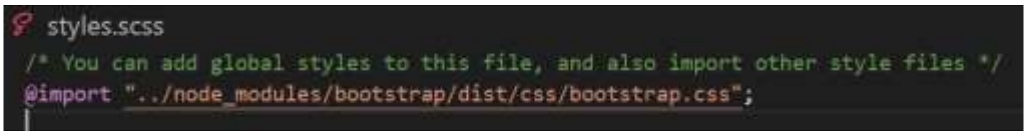
npm install @ngrx/store @ngrx/store-devtools @ngrx/effects bootstrap --save



```
F:\proyectos angular\crud-redux-app>npm install @ngrx/store @ngrx/store-devtools @ngrx/effects bootstrap --save
```

Importando los estilos de bootstrap:

```
@import "../node_modules/bootstrap/dist/css/bootstrap.css";
```

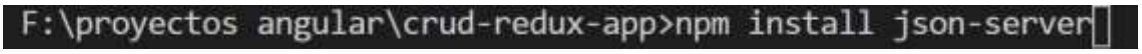


styles.scss

```
/* You can add global styles to this file, and also import other style files */
@import "../node_modules/bootstrap/dist/css/bootstrap.css";
```

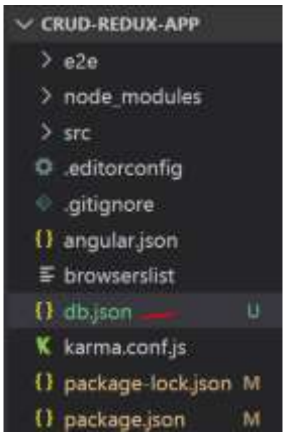
Instalando json-server (Servidor local)

npm install json-server



```
F:\proyectos angular\crud-redux-app>npm install json-server
```

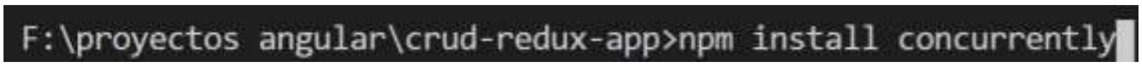
Crear el archivo db.json en la raíz de la aplicación:



```
CRUD-REDUX-APP
  > e2e
  > node_modules
  > src
  .editorconfig
  .gitignore
  angular.json
  browserslist
  db.json            U
  karma.conf.js
  package-lock.json  M
  package.json       M
```

```
{
  "usuarios": [
    {
      "name": "Henry",
      "age": 22,
      "email": "henry@email.com",
      "id": 1
    },
    {
      "id": 3,
      "name": "Carmen",
      "age": 30,
      "email": "carmen@email.com"
    },
    {
      "id": 4,
      "name": "Miguel",
      "age": 28,
      "email": "miguel@email.com"
    },
    {
      "name": "Esteban",
      "age": 40,
      "email": "esteban@email.com",
      "id": 1572300257635
    },
    {
      "name": "Mario",
      "age": 25,
      "email": "mario@email.com",
      "id": 1572371378938
    }
  ]
}
```
Para ejecutar el servidor de angular junto con el servidor local, instalar concurrently:

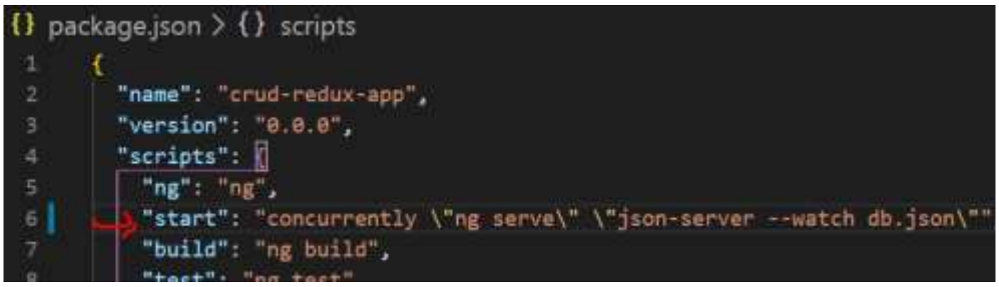
npm install concurrently



```
F:\proyectos angular\crud-redux-app>npm install concurrently
```

Modificar el archivo package.json en la ejecución del comando start:

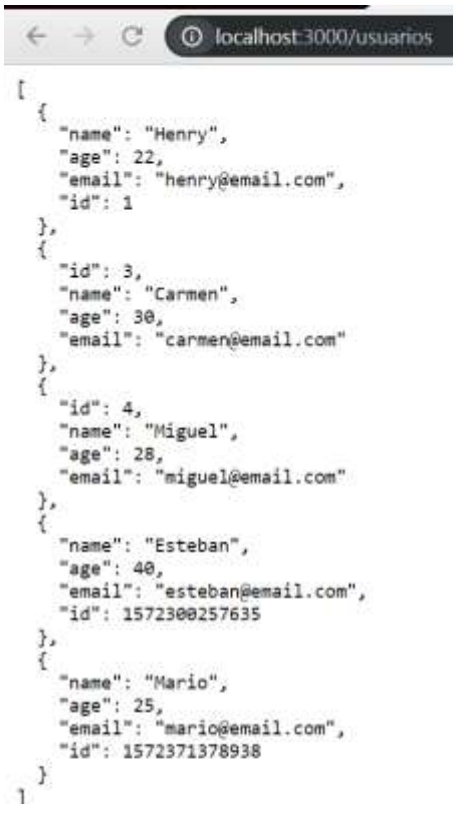
```
"start": "concurrently \"ng serve\" \"json-server --watch db.json\"",
```



```
{} package.json > {} scripts
{
  "name": "crud-redux-app",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "concurrently \"ng serve\" \"json-server --watch db.json\""
    "build": "ng build",
    "test": "ng test"
```

Ejecutar la aplicación:

npm start



localhost:3000/usuarios

```
[
  {
    "name": "Henry",
    "age": 22,
    "email": "henry@email.com",
    "id": 1
  },
  {
    "id": 3,
    "name": "Carmen",
    "age": 30,
    "email": "carmen@email.com"
  },
  {
    "id": 4,
    "name": "Miguel",
    "age": 28,
    "email": "miguel@email.com"
  },
  {
    "name": "Esteban",
    "age": 40,
    "email": "esteban@email.com",
    "id": 1572300257635
  },
  {
    "name": "Mario",
    "age": 25,
    "email": "mario@email.com",
    "id": 1572371378938
  }
]
```

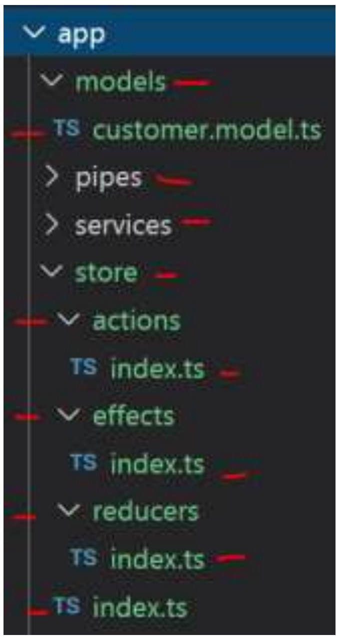
Configuración de estructura del proyecto

EL archivo styles.scss quedaría de la siguiente manera:

```scss
/* You can add global styles to this file, and also import other style files */
@import "../node_modules/bootstrap/dist/css/bootstrap.css";

.overlay {
  background-color: rgba(0, 0, 0, 0.5) !important;
  position: fixed !important;
  top: 0 !important;
  z-index: 999;
  left: 0 !important;
  width: 100% !important;
  height: 100%;
}

a > .btn {
  cursor: pointer;
}

div.modal {
  height: auto;
  z-index: 999;
  color: #000;
  animation: movercaja 0.4s ease-out forwards;
}

@keyframes movercaja {
  from {
    top: -10px;
    opacity: 1;
  }
  to {
    top: 100px;
    opacity: 1;
  }
}

@-webkit-keyframes movercaja {
  from {
    top: -10px;
    opacity: 1;
  }
  to {
    top: 100px;
    opacity: 1;
  }
}

#filterIcon {
  position: relative;
  z-index: 1;
  left: -25px;
  top: 8px;
  color: #7b7b7b;
}

input[type="text"] {
  margin: 2px !important;
}

input[type="text"]:focus {
  outline-width: 0;
}
```

```
input[id="filter"] {
  width: 30% !important;
}
```

Creando los directorios:



```
app
  models
    customer.model.ts
  pipes
  services
  store
    actions
      index.ts
    effects
      index.ts
    reducers
      index.ts
    index.ts
```

Modelo:

```
export interface Customer {
  id?: number;
  name?: string;
  age?: number;
  email?: string;
}
```

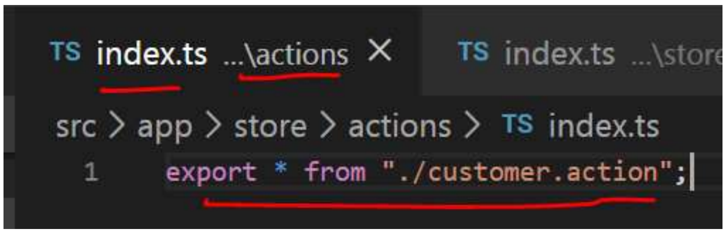
Store Reducer, Actions y más

Dentro del documento index de la carpeta store, se importarán todos los archivos de “actions”, “effects” y “reducers”.

En la carpeta “actions”, crear el archivo “customer.action.ts”:

```
import { Action } from "@ngrx/store";

export const LOAD_CUSTOMER = "[Customer] Load Customer";

export class LoadCustomer implements Action {
  readonly type = LOAD_CUSTOMER;
}

export type CustomerActions = LoadCustomer;
```

Y exportarlo en el index de la carpeta “actions”:



```
TS index.ts ...\actions ×    TS index.ts ...\store
src > app > store > actions > TS index.ts
export * from "./customer.action";
```

Luego, en “reducers”, crear “app.reducer.ts”:

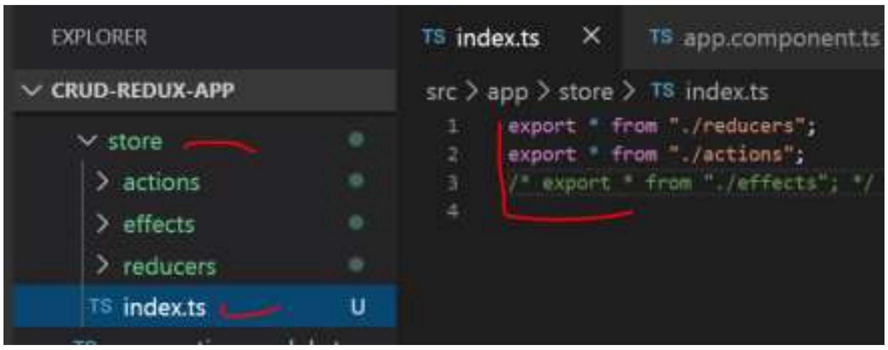
```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [],
  loaded: false,
  loading: false,
  error: ""
};

// Función reducer
export function reducer(state = initState, action) {
  switch (action.type) {
    case fromCustomerActions.LOAD_CUSTOMER:
      return { ...state, loading: true };

    default:
      return state;
  }
}
```

Luego, en el index de la carpeta “reducers”, irán todos los reducers creados (en este caso “customer.reducer“ teniendo una interface (AppState):

```
// Referencia a "app.reducer"
import * as fromCustomerReducer from "./app.reducer";

// Estado de toda la aplicación (modelo)
export interface AppState {
  /* Estados de toda la aplicación */
  // Dando estado
  customer: fromCustomerReducer.CustomerState; // tipado
}

export const reducers = {
  customer: fromCustomerReducer.reducer
};
```

Por último, registrar todos los “reducers” a la aplicación (app.module).

Nota: Como no hay archivos en la carpeta “effects”, comentarla del index de store



```
export * from "./reducers";
export * from "./actions";
/* export * from "./effects"; */
```

```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";

import { AppRoutingModule } from "./app-routing.module";
import { AppComponent } from "./app.component";
// Importando los reducers
import { StoreModule } from "@ngrx/store";
import { reducers } from "./store";

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, AppRoutingModule, StoreModule.forRoot(reducers)],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Ejemplo del uso en el app.component.ts

Para cargar los datos del json, ir a app.reducer.ts (CustomerState)

```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [
    {
      name: "Henry",
      age: 22,
      email: "henry@email.com",
      id: 1
    },
    {
      id: 3,
      name: "Carmen",
      age: 30,
      email: "carmen@email.com"
    },
    {
      id: 4,
      name: "Miguel",
      age: 28,
      email: "miguel@email.com"
    }
  ],
  loaded: false,
  loading: false,
  error: ""
};

// Función reducer
export function reducer(state = initState, action) {
  switch (action.type) {
    case fromCustomerActions.LOAD_CUSTOMER:
      return { ...state, loading: true };

    default:
      return state;
  }
}
```

Luego, en el app.component.ts:

```
import { Component } from "@angular/core";
import { Store } from "@ngrx/store";
import * as fromStore from "./store"; // referencia al archivo index.ts de "store"

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]
})
export class AppComponent {
  title = "crud-redux-app";

  constructor(private store: Store<fromStore.AppState>) {
    store.select("customers").subscribe(response => {
      console.log(response);
    });
  }
}
```

Select - Load Customers y más

Implementar font awesome (index):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>CrudReduxApp</title>
    <base href="/" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="icon" type="image/x-icon" href="favicon.ico" />
    <link
      href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css"
      rel="stylesheet"
      integrity="sha384-
wvfXpqpZZVQGK6TAh5PVlGOfQNHSoD2xbE+QkPxCAFlNEevoEH3Sl0sibVcOQVnN"
      crossorigin="anonymous"
    />
  </head>
  <body>
    <app-root></app-root>
  </body>
</html>
```

Usar devtools

Ir a “app.module.ts”:

```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";

import { AppRoutingModule } from "./app-routing.module";
import { AppComponent } from "./app.component";
// Importando los reducers
import { StoreModule } from "@ngrx/store";
import { reducers } from "./store";
import { StoreDevtoolsModule } from "@ngrx/store-devtools";

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    AppRoutingModule,
    StoreModule.forRoot(reducers),
    StoreDevtoolsModule.instrument({})
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

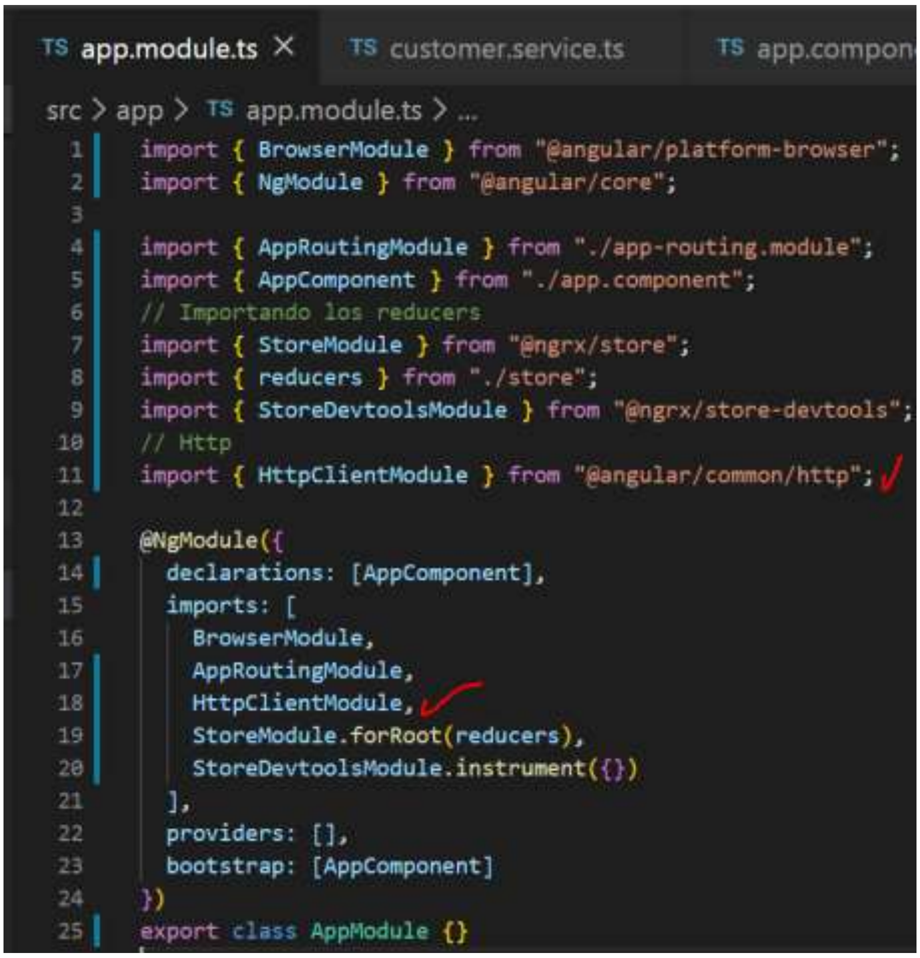
Listar los datos iniciales:

```
import { Component } from "@angular/core";
import { Store } from "@ngrx/store";
import * as fromStore from "./store"; // referencia al archivo index.ts de "store"
import { Customer } from "./models/customer.model";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]
})
export class AppComponent {
  customers: Customer[];

  constructor(private store: Store<fromStore.AppState>) {
    store.select("customers").subscribe(response => {
      this.customers = response.data;
    });
  }
}
```

```
<div class="container">
  <form
    style="background:#428bca;padding-top:20px;margin-top:20px;padding-bottom:2px;"
  >
    <div class="form-group">
      <div
        class="input-group"
        style="padding-left: 10px; width: 100% !important;"
      >
        <input type="text" id="filter" name="text" placeholder="Search" />
        <i class="fa fa-search" id="filterIcon"></i>
      </div>
    </div>
  </form>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Name</th>
        <th>Age</th>
        <th>Email</th>
        <th>Remove</th>
        <th>Edit</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let customer of customers">
        <td>{{ customer.name }}</td>
        <td>{{ customer.age }}</td>
        <td>{{ customer.email }}</td>
        <td><button class="btn btn-danger fa fa-trash-o"></button></td>
        <td>
          <button class="btn btn-warning fa fa-pencil-square-o"></button>
        </td>
      </tr>
    </tbody>
  </table>
</div>
```

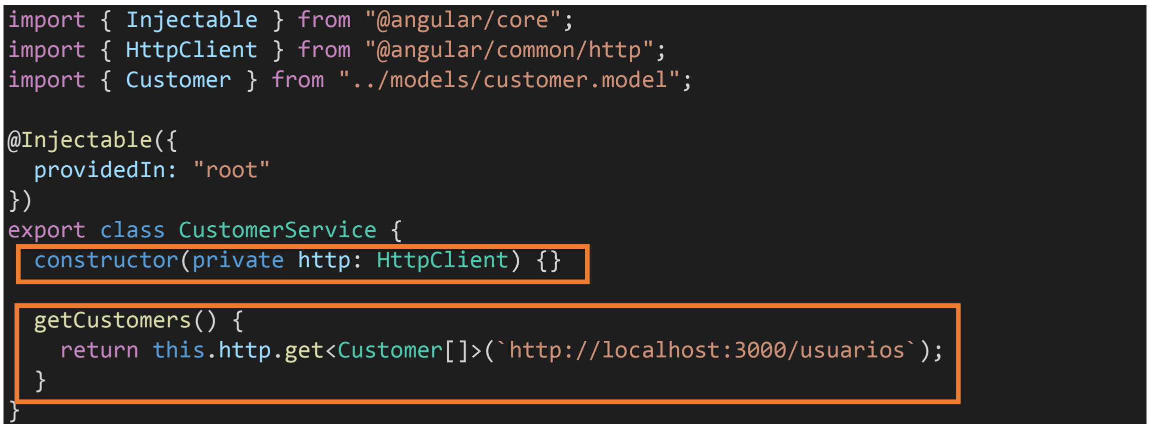
Servicios - HTTP Services

Importar el módulo http en app.module.ts:



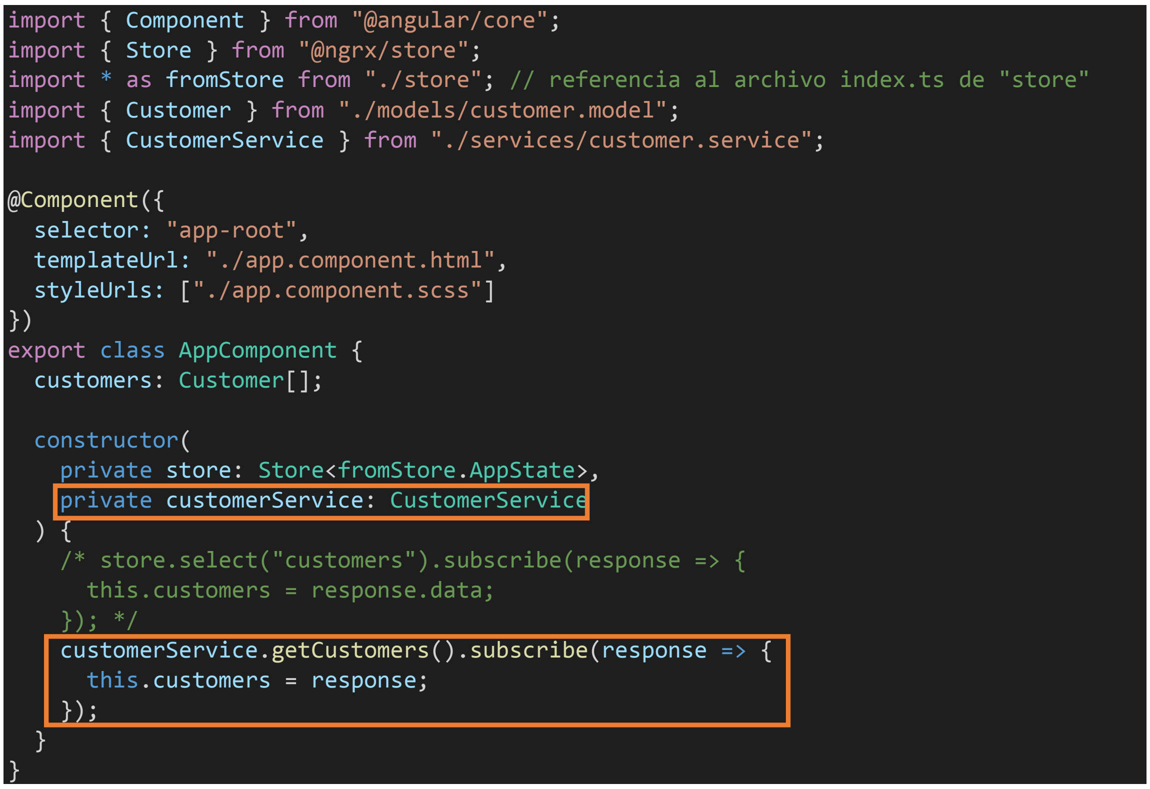
```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";

import { AppRoutingModule } from "./app-routing.module";
import { AppComponent } from "./app.component";
// Importando los reducers
import { StoreModule } from "@ngrx/store";
import { reducers } from "./store";
import { StoreDevtoolsModule } from "@ngrx/store-devtools";
// Http
import { HttpClientModule } from "@angular/common/http";

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    StoreModule.forRoot(reducers),
    StoreDevtoolsModule.instrument({})
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Creando el servicio “customer.service.ts”:



```
import { Injectable } from "@angular/core";
import { HttpClient } from "@angular/common/http";
import { Customer } from "../models/customer.model";

@Injectable({
  providedIn: "root"
})
export class CustomerService {
  constructor(private http: HttpClient) {}

  getCustomers() {
    return this.http.get<Customer[]>(`http://localhost:3000/usuarios`);
  }
}
```

Usar el servicio en el componente:

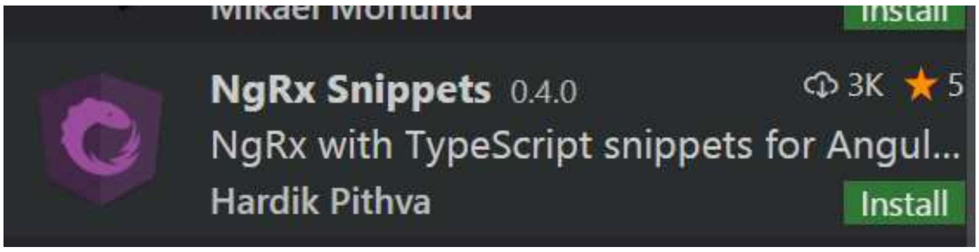


```
import { Component } from "@angular/core";
import { Store } from "@ngrx/store";
import * as fromStore from "./store"; // referencia al archivo index.ts de "store"
import { Customer } from "./models/customer.model";
import { CustomerService } from "./services/customer.service";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]
})
export class AppComponent {
  customers: Customer[];

  constructor(
    private store: Store<fromStore.AppState>,
    private customerService: CustomerService
  ) {
    /* store.select("customers").subscribe(response => {
      this.customers = response.data;
    }); */
    customerService.getCustomers().subscribe(response => {
      this.customers = response;
    });
  }
}
```

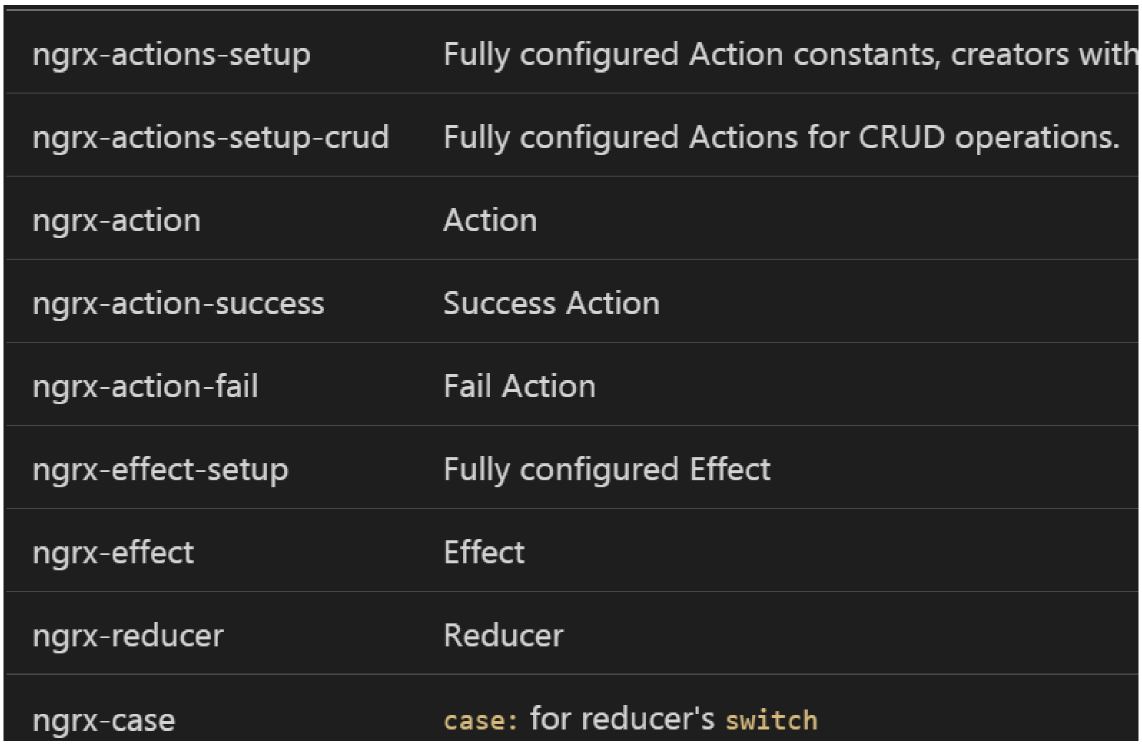
Effects

Permite realizar peticiones hacia un API, no será necesario usar un servicio directamente desde el componente.

Instalando snippets de ngRx:

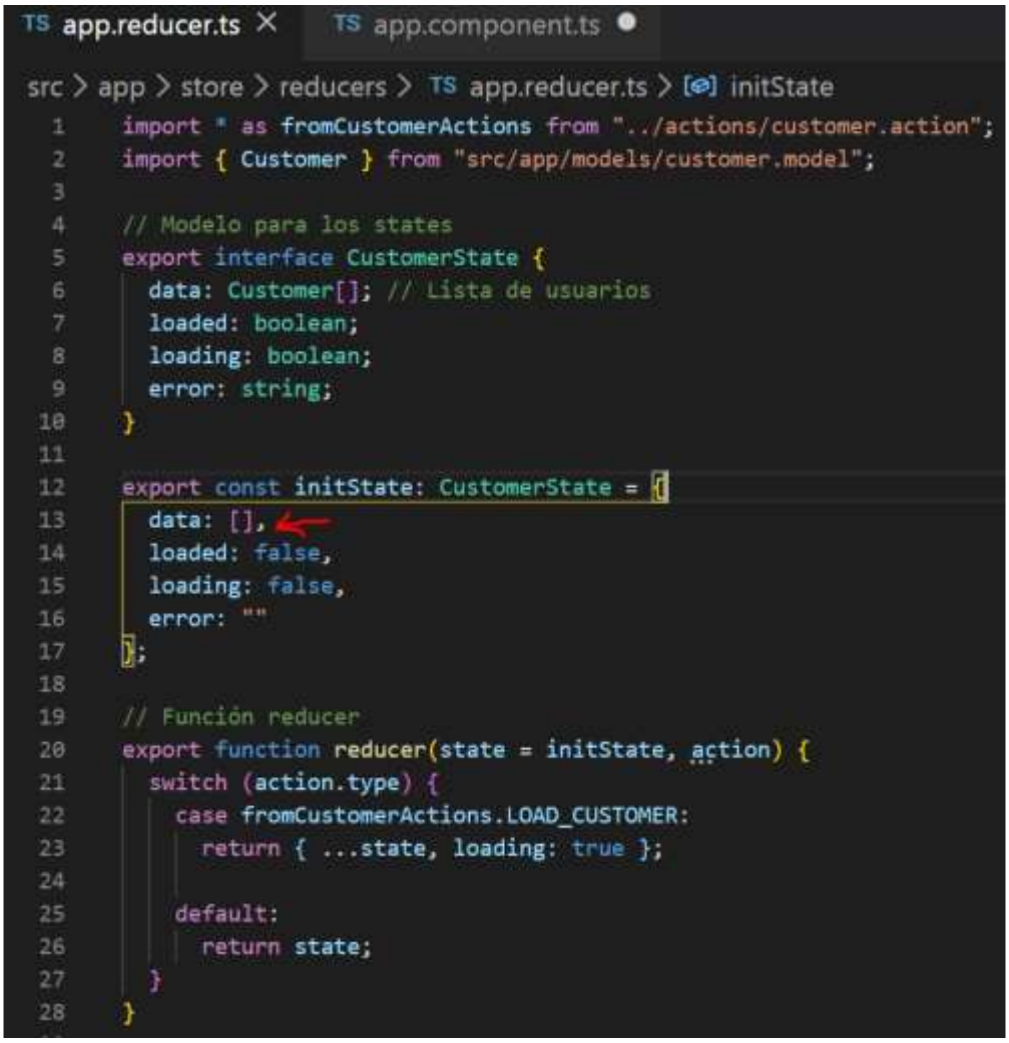


NgRx Snippets 0.4.0
3K ★ 5
NgRx with TypeScript snippets for Angul...
Hardik Pithva
Install



ngrx-actions-setup	Fully configured Action constants, creators with
ngrx-actions-setup-crud	Fully configured Actions for CRUD operations.
ngrx-action	Action
ngrx-action-success	Success Action
ngrx-action-fail	Fail Action
ngrx-effect-setup	Fully configured Effect
ngrx-effect	Effect
ngrx-reducer	Reducer
ngrx-case	`case:` for reducer's `switch`

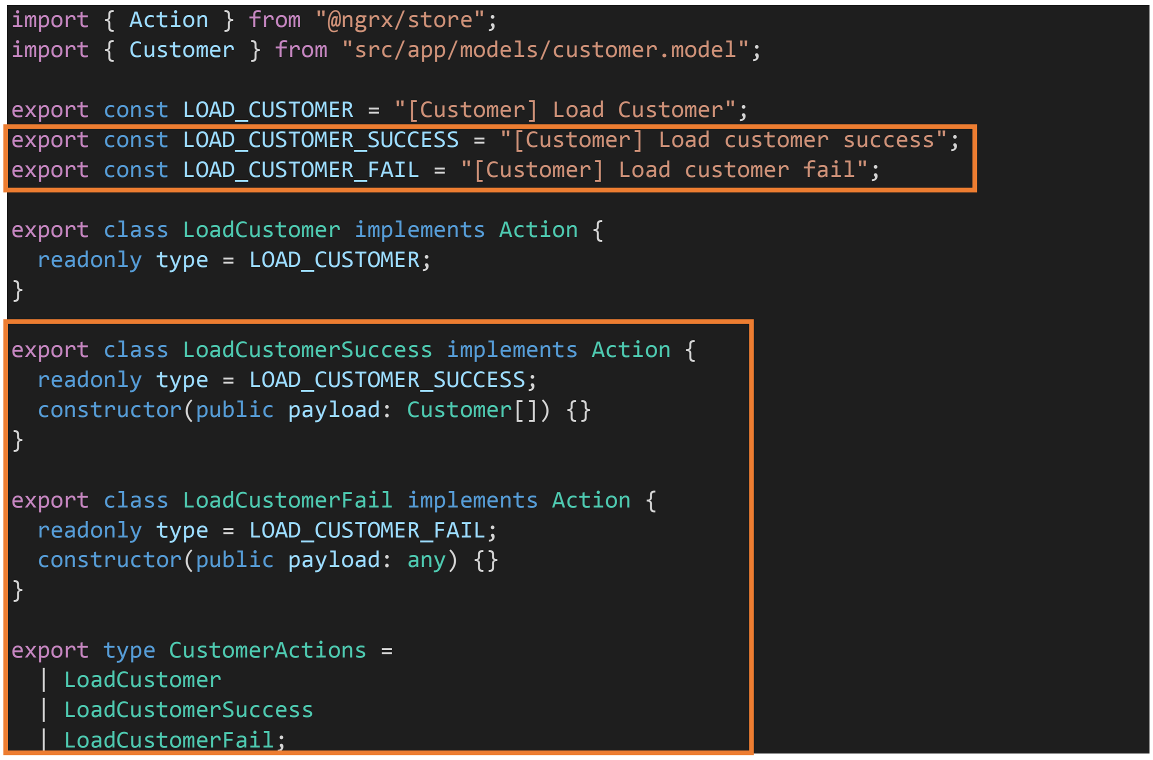
En el archivo app.reducer borrar el estado inicial con los datos de prueba:



TS app.reducer.ts
TS app.component.ts
src > app > store > reducers > TS app.reducer.ts > initState

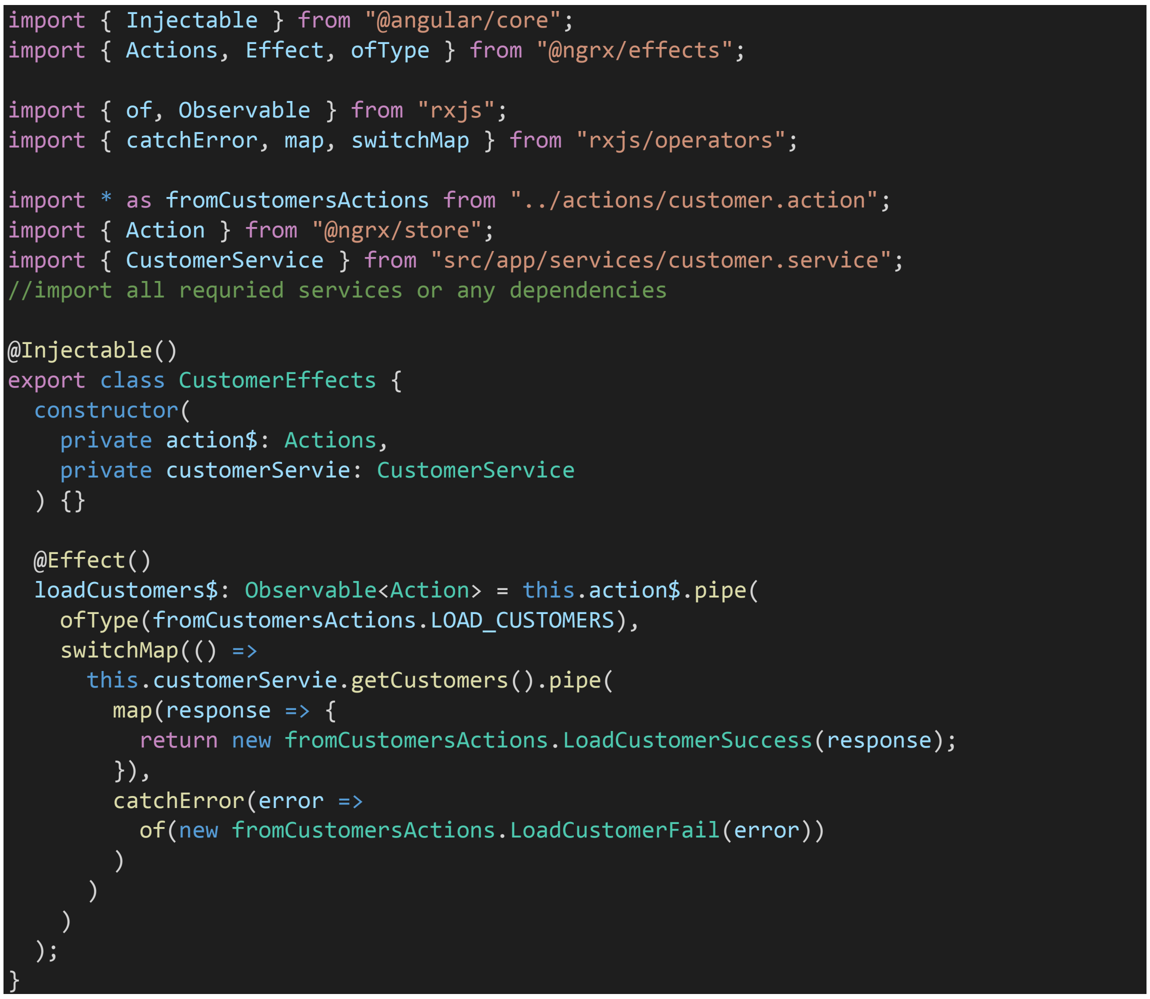
```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [],
  loaded: false,
  loading: false,
  error: ""
};

// Función reducer
export function reducer(state = initState, action) {
  switch (action.type) {
    case fromCustomerActions.LOAD_CUSTOMER:
      return { ...state, loading: true };

    default:
      return state;
  }
}
```

Usando el snippt, crear una nueva acción:



```
import { Action } from "@ngrx/store";
import { Customer } from "src/app/models/customer.model";

export const LOAD_CUSTOMER = "[Customer] Load Customer";
export const LOAD_CUSTOMER_SUCCESS = "[Customer] Load customer success";
export const LOAD_CUSTOMER_FAIL = "[Customer] Load customer fail";

export class LoadCustomer implements Action {
  readonly type = LOAD_CUSTOMER;
}

export class LoadCustomerSuccess implements Action {
  readonly type = LOAD_CUSTOMER_SUCCESS;
  constructor(public payload: Customer[]) {}
}

export class LoadCustomerFail implements Action {
  readonly type = LOAD_CUSTOMER_FAIL;
  constructor(public payload: any) {}
}

export type CustomerActions =
  | LoadCustomer
  | LoadCustomerSuccess
  | LoadCustomerFail;
```

Creando el effect que interactuará con el API:



```
import { Injectable } from "@angular/core";
import { Actions, Effect, ofType } from "@ngrx/effects";

import { of, Observable } from "rxjs";
import { catchError, map, switchMap } from "rxjs/operators";

import * as fromCustomersActions from "../actions/customer.action";
import { Action } from "@ngrx/store";
import { CustomerService } from "src/app/services/customer.service";
//import all requried services or any dependencies

@Injectable()
export class CustomerEffects {
  constructor(
    private action$: Actions,
    private customerServie: CustomerService
  ) {}

  @Effect()
  loadCustomers$: Observable<Action> = this.action$.pipe(
    ofType(fromCustomersActions.LOAD_CUSTOMERS),
    switchMap(() =>
      this.customerServie.getCustomers().pipe(
        map(response => {
          return new fromCustomersActions.LoadCustomerSuccess(response);
        }),
        catchError(error =>
          of(new fromCustomersActions.LoadCustomerFail(error))
        )
      )
    )
  );
}
```

El index de la carpeta “effects” manejará los effects:

```
import { CustomerEffects } from "./app-effect";

export const effects: any[] = [CustomerEffects];

export * from "./app-effect";
```

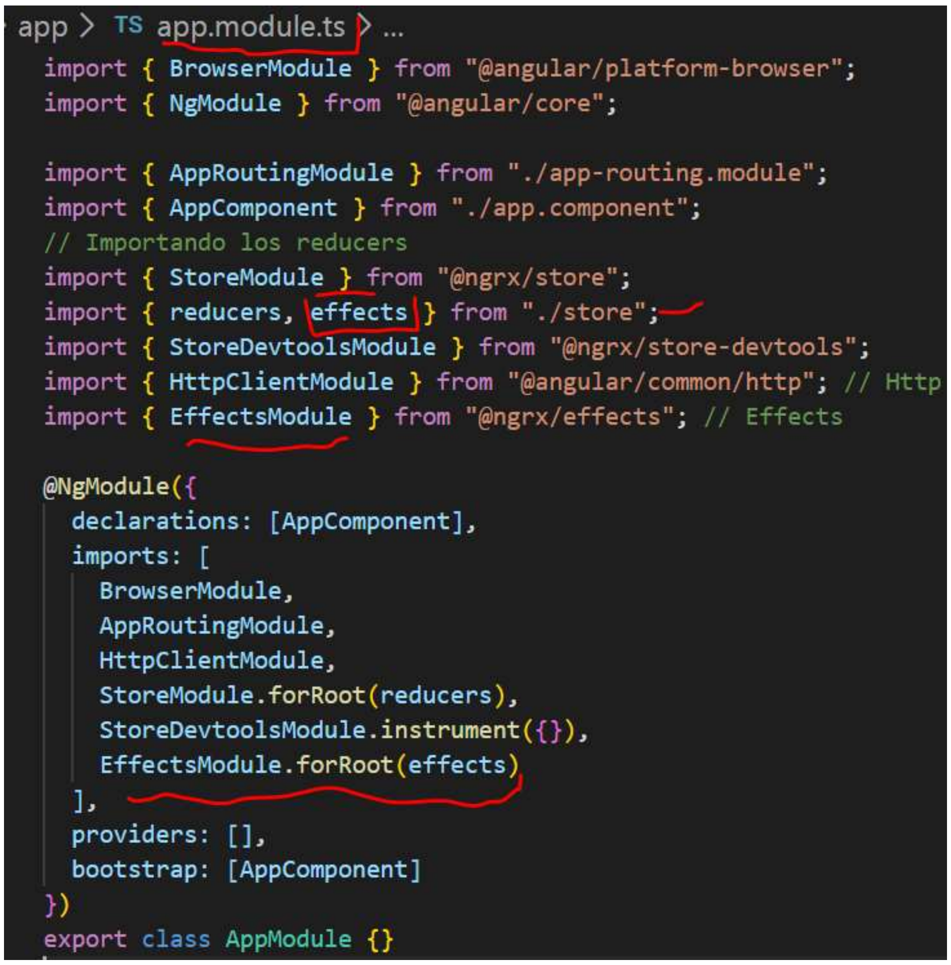
Exportar effects desde el store:

```
export * from "./reducers";
export * from "./actions";
export * from "./effects";
```

En el reducer:

```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [],
  loaded: false,
  loading: false,
  error: ""
};

// Función reducer
export function reducer(
  state = initState,
  action: fromCustomerActions.CustomerActions
) {
  switch (action.type) {
    case fromCustomerActions.LOAD_CUSTOMERS:
      return { ...state, loading: true };
    case fromCustomerActions.LOAD_CUSTOMERS_SUCCESS:
      const data = action.payload; // Lista de clientes enviada por el api
      return { ...state, loading: false, loaded: true, data: data };
    case fromCustomerActions.LOAD_CUSTOMERS_FAIL:
      return { ...state, loading: false, loaded: false, error: action.payload };
    default:
      return state;
  }
}
```

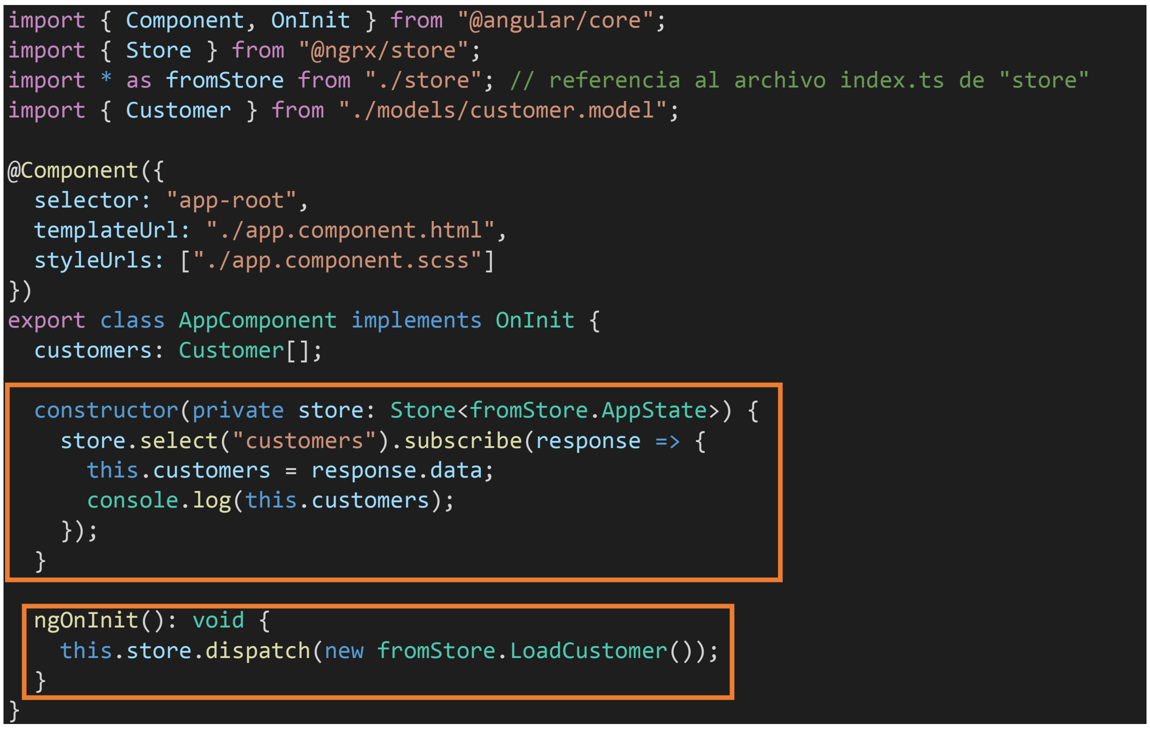
Registrar effects al módulo:



app > TS app.module.ts > ...

```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";

import { AppRoutingModule } from "./app-routing.module";
import { AppComponent } from "./app.component";
// Importando los reducers
import { StoreModule } from "@ngrx/store";
import { reducers, effects } from "./store";
import { StoreDevtoolsModule } from "@ngrx/store-devtools";
import { HttpClientModule } from "@angular/common/http"; // Http
import { EffectsModule } from "@ngrx/effects"; // Effects

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    StoreModule.forRoot(reducers),
    StoreDevtoolsModule.instrument({}),
    EffectsModule.forRoot(effects)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

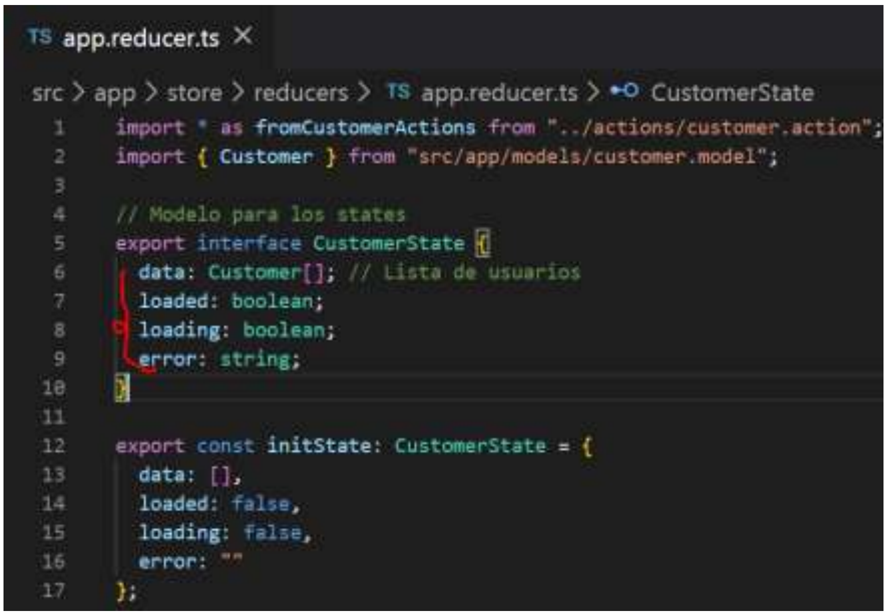
Usarlo en el componente:



```
import { Component, OnInit } from "@angular/core";
import { Store } from "@ngrx/store";
import * as fromStore from "./store"; // referencia al archivo index.ts de "store"
import { Customer } from "./models/customer.model";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]
})
export class AppComponent implements OnInit {
  customers: Customer[];

  constructor(private store: Store<fromStore.AppState>) {
    store.select("customers").subscribe(response => {
      this.customers = response.data;
      console.log(this.customers);
    });
  }

  ngOnInit(): void {
    this.store.dispatch(new fromStore.LoadCustomer());
  }
}
```

createSelector – createFeatureSelector

Para acceder a los estados creados y a sus propiedades (ejemplo: estado “customer” tiene las propiedades “data”, “loading”, “loaded” y “error”).


TS app.reducer.ts ✕

src > app > store > reducers > TS app.reducer.ts > CustomerState

```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [],
  loaded: false,
  loading: false,
  error: ""
};
```

Creando los selectores en app.reducer:

```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [],
  loaded: false,
  loading: false,
  error: ""
};

// Función reducer
export function reducer(
  state = initState,
  action: fromCustomerActions.CustomerActions
) {
  switch (action.type) {
    case fromCustomerActions.LOAD_CUSTOMERS:
      return { ...state, loading: true };
    case fromCustomerActions.LOAD_CUSTOMERS_SUCCESS:
      const data = action.payload; // Lista de clientes enviada por el api
      return { ...state, loading: false, loaded: true, data: data };
    case fromCustomerActions.LOAD_CUSTOMERS_FAIL:
      return { ...state, loading: false, loaded: false, error: action.payload
 };
    default:
      return state;
  }
}
```

```
// Creando selector que retorne la propiedad "data"
export const getCustomers = (state: CustomerState) => state.data;
export const getCustomersLoaded = (state: CustomerState) => state.loaded;
export const getCustomersLoading = (state: CustomerState) => state.loading;
export const getCustomersError = (state: CustomerState) => state.error;
```

En el index de la misma carpeta:

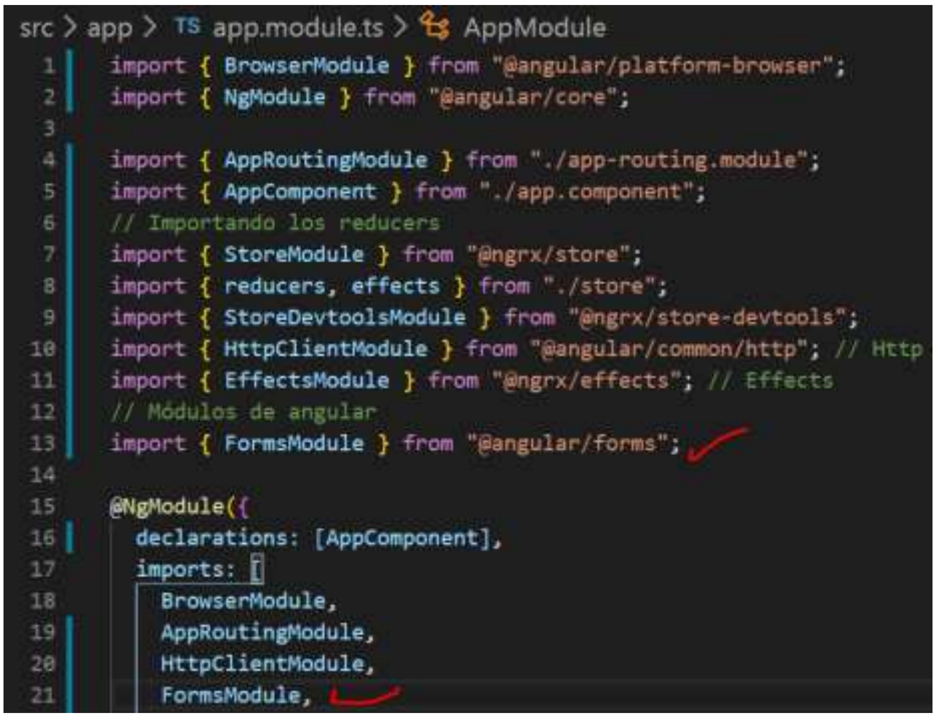
```
// Referencia a "app.reducer"
import * as fromCustomerReducer from "./app.reducer";
import { createFeatureSelector, createSelector } from '@ngrx/store';

// Estado de toda la aplicación (modelo)
export interface AppState {
  /* Estados de toda la aplicación */
  // Dando estado
  customers: fromCustomerReducer.CustomerState; // tipado
}

export const reducers = {
  customers: fromCustomerReducer.reducer
};

export const getState = state => state; // Acceso a todas las propiedades
// Acceso a customers
export const getCustomersState = createFeatureSelector<fromCustomerReducer.CustomerSt
ate>("customers");
// Acceso a la propiedad "data"
export const getCustomers = createSelector(getCustomersState, fromCustomerReducer.get
Customers);
// Acceso a un cliente por id
export const getCustomerById = (id) => createSelector(getCustomers, (customers) => {
  if (getCustomers) {
    var customerFound = customers.find(persona => {
      return persona.id === id;
    })
    return customerFound || {};
  } else {
    return {};
  }

});
```

Usando los selectors en el componente:

```
import { Component, OnInit } from "@angular/core";
import { Store } from "@ngrx/store";
import * as fromStore from "./store"; // referencia al archivo index.ts de "store"
import { Customer } from "./models/customer.model";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]
})
export class AppComponent implements OnInit {
  customers: Customer[];

  constructor(private store: Store<fromStore.AppState>) {
    store.select(fromStore.getCustomers).subscribe(response => {
      this.customers = response;
    });
store.select(fromStore.getCustomerById(2)).subscribe(response => {
      console.log(response);
    });
  }

  ngOnInit(): void {
    this.store.dispatch(new fromStore.LoadCustomer());
  }
}
```

Window - Pop-up – Overlay

Importar el FormsModule en el app.module:



src > app > app.module.ts > AppModule

```
import { BrowserModule } from "@angular/platform-browser";
import { NgModule } from "@angular/core";

import { AppRoutingModule } from "./app-routing.module";
import { AppComponent } from "./app.component";
// Importando los reducers
import { StoreModule } from "@ngrx/store";
import { reducers, effects } from "./store";
import { StoreDevtoolsModule } from "@ngrx/store-devtools";
import { HttpClientModule } from "@angular/common/http"; // Http
import { EffectsModule } from "@ngrx/effects"; // Effects
// Módulos de angular
import { FormsModule } from "@angular/forms";

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule,
    FormsModule,
```

En el app.component:

```
import { Component, OnInit } from "@angular/core";
import { Store } from "@ngrx/store";
import * as fromStore from "./store"; // referencia al archivo index.ts de "s
tore"
import { Customer } from "./models/customer.model";
import { NgForm } from "@angular/forms";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]
})
export class AppComponent implements OnInit {
  customers: Customer[];
  display: string = "none";
  isEditModeEnabled: boolean = false;

  constructor(private store: Store<fromStore.AppState>) {
    store.select(fromStore.getCustomers).subscribe(response => {
      this.customers = response;
    });

    store.select(fromStore.getCustomerById(2)).subscribe(response => {
      console.log(response);
    });
  }

  ngOnInit(): void {
    this.store.dispatch(new fromStore.LoadCustomer());
  }

  openModelDialog() {
    this.display = "block";
  }

  closeModal(myForm: NgForm) {
    this.display = "none";
  }

  addCustomer(myForm: NgForm) {}

  updateCustomer(myForm: NgForm) {}
}
```

En el template:

```
<div class="overlay" [ngStyle]="{ display: display }"></div>
<!-- Modal -->
<div class="modal" [ngStyle]="{ display: display }" tabindex="-
1" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">
          {{ isEditModeEnabled ? "Edit" : "Add" }} customer
        </h5>
        <button
          (click)="closeModal(myForm)"
          type="button"
          class="close"
          data-dismiss="modal"
          aria-label="Close"
        >
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        <form #myForm="ngForm">
          <div class="form-group">
            <input
              type="text"
              class="form-control"
              placeholder="Enter a name"
              name="name"
              #name
            />
          </div>
          <div class="form-group">
            <input
              type="number"
              class="form-control"
              placeholder="Enter your age"
              name="age"
              #age
            />
          </div>
          <div class="form-group">
```

```
            <input
              type="email"
              class="form-control"
              placeholder="Enter your email"
              name="email"
              #email
            />
          </div>
          <div class="form-group">
            <input type="text" class="form-control" name="id" #id hidden />
          </div>
        </form>
      </div>
      <div class="modal-footer">
        <button
          class="btn btn-secondary"
          data-dismiss="modal"
          (click)="closeModal(myForm)"
        >
          Close
        </button>
        <input
          *ngIf="!isEditModeEnabled"
          type="button"
          class="btn btn-primary"
          (click)="addCustomer(myForm)"
          value="Add Client"
        />
        <input
          *ngIf="isEditModeEnabled"
          type="button"
          class="btn btn-success"
          (click)="updateCustomer(myForm)"
          value="Update"
        />
      </div>
    </div>
  </div>
</div>
<!-- Tabla -->
<div class="container">
  <form
    style="background:#428bca;padding-top:20px;margin-top:20px;padding-
bottom:2px;"
  >
    <div class="form-group">
      <div
        class="input-group"
        style="padding-left: 10px; width: 100% !important;"
      >
        <input type="text" id="filter" name="text" placeholder="Search" />
        <i class="fa fa-search" id="filterIcon"></i>
      </div>
    </div>
  </form>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Name</th>
        <th>Age</th>
        <th>Email</th>
        <th>Remove</th>
        <th>Edit</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let customer of customers">
        <td>{{ customer.name }}</td>
        <td>{{ customer.age }}</td>
        <td>{{ customer.email }}</td>
        <td><button class="btn btn-danger fa fa-trash-o"></button></td>
        <td>
          <button class="btn btn-warning fa fa-pencil-square-o"></button>
        </td>
      </tr>
    </tbody>
  </table>
  <!-- Mostrar el modal -->
  <button (click)="openModelDialog()" class="btn btn-primary">
    Add customer
  </button>
</div>
```

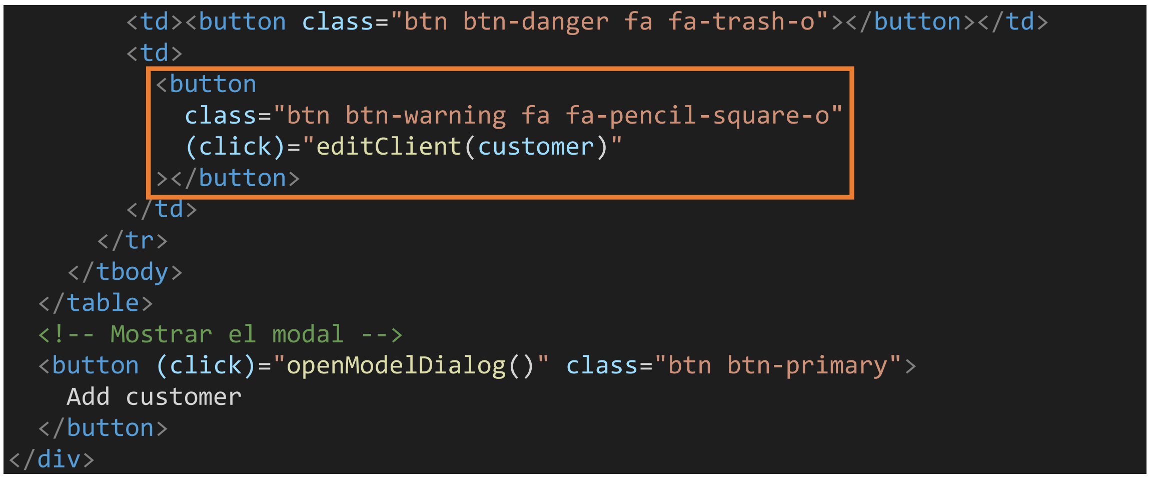
Editar usuario – EDIT

En el componente:

```
import { Component, OnInit } from "@angular/core";
import { Store } from "@ngrx/store";
import * as fromStore from "./store"; // referencia al archivo index.ts de "s
tore"
import { Customer } from "./models/customer.model";
import { NgForm } from "@angular/forms";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]
})
export class AppComponent implements OnInit {
  customers: Customer[];
  display: string = "none";
  isEditModeEnabled: boolean = false;
  person: Customer = {};

  constructor(private store: Store<fromStore.AppState>) {
    store.select(fromStore.getCustomers).subscribe(response => {
      this.customers = response;
    });

    store.select(fromStore.getCustomerById(2)).subscribe(response => {
      console.log(response);
    });
  }

  ngOnInit(): void {
    this.store.dispatch(new fromStore.LoadCustomer());
  }

  openModelDialog() {
    this.display = "block";
  }

  closeModal(myForm: NgForm) {
    this.display = "none";
  }
  editClient(customer: Customer) {
    this.isEditModeEnabled = true;
    this.person = {...customer}; // Copia del cliente seleccionado
    this.display = "block";
  }

  addCustomer(myForm: NgForm) {}

  updateCustomer(myForm: NgForm) {}
}
```

En el template:

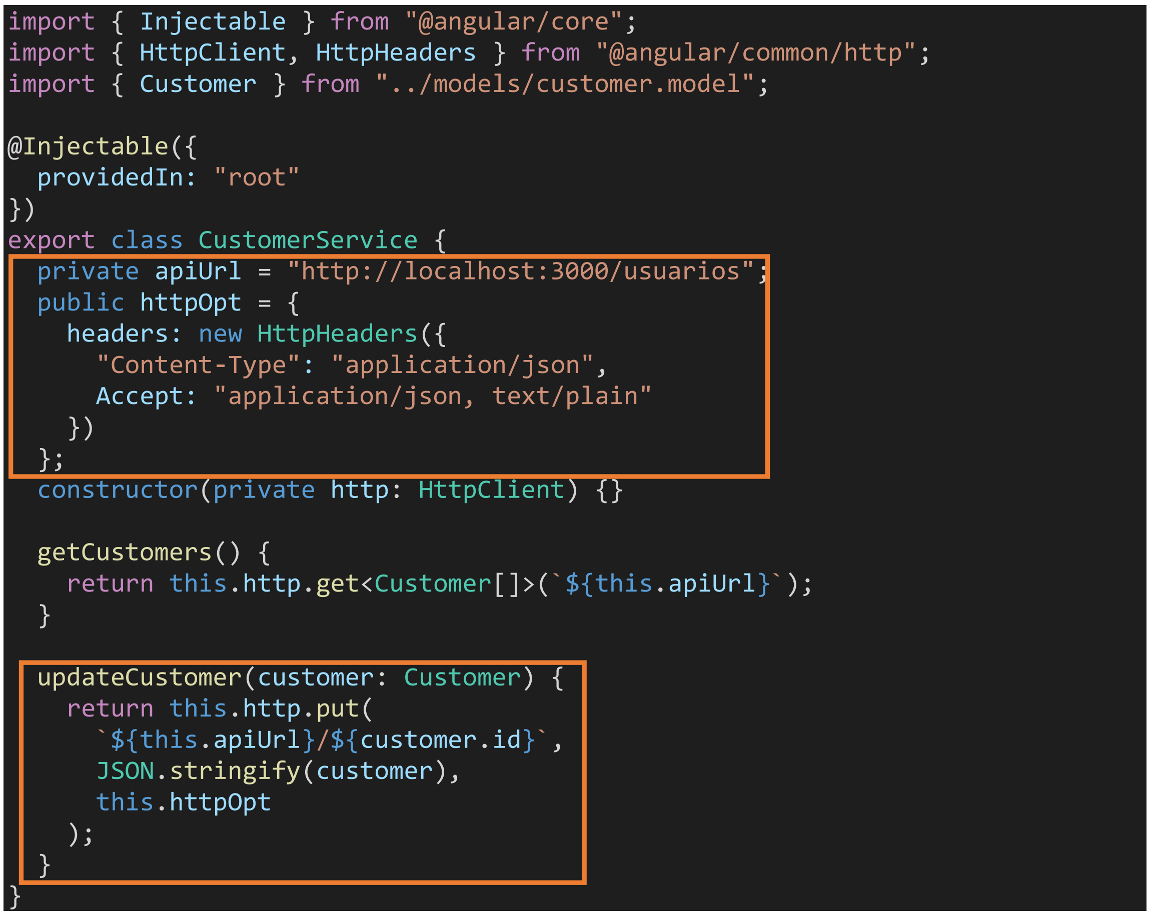
```
<div class="overlay" [ngStyle]="{ display: display }"></div>
<!-- Modal -->
<div class="modal" [ngStyle]="{ display: display }" tabindex="-
1" role="dialog">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">
          {{ isEditModeEnabled ? "Edit" : "Add" }} customer
        </h5>
        <button
          (click)="closeModal(myForm)"
          type="button"
          class="close"
          data-dismiss="modal"
          aria-label="Close"
        >
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        <form #myForm="ngForm">
          <div class="form-group">
            <input
              type="text"
              class="form-control"
              placeholder="Enter a name"
              name="name"
              [ngModel]="person.name"
              #name
            />
          </div>
          <div class="form-group">
            <input
              type="number"
              class="form-control"
              placeholder="Enter your age"
              name="age"
              [ngModel]="person.age"
              #age
            />
          </div>
          <div class="form-group">
            <input
              type="email"
              class="form-control"
              placeholder="Enter your email"
              name="email"
              [ngModel]="person.email"
              #email
            />
          </div>
          <div class="form-group">
            <input type="text" class="form-control" name="id" #id hidden />
          </div>
        </form>
      </div>
      <div class="modal-footer">
        <button
          class="btn btn-secondary"
          data-dismiss="modal"
          (click)="closeModal(myForm)"
        >
          Close
        </button>
        <input
          *ngIf="!isEditModeEnabled"
          type="button"
          class="btn btn-primary"
          (click)="addCustomer(myForm)"
          value="Add Client"
        />
        <input
          *ngIf="isEditModeEnabled"
          type="button"
          class="btn btn-success"
          (click)="updateCustomer(myForm)"
          value="Update"
        />
      </div>
    </div>
  </div>
</div>
<!-- Tabla -->
<div class="container">
  <form
    style="background:#428bca;padding-top:20px;margin-top:20px;padding-
bottom:2px;"
  >
    <div class="form-group">
      <div
        class="input-group"
        style="padding-left: 10px; width: 100% !important;"
      >
        <input type="text" id="filter" name="text" placeholder="Search" />
        <i class="fa fa-search" id="filterIcon"></i>
      </div>
    </div>
  </form>
  <table class="table table-striped">
    <thead>
      <tr>
        <th>Name</th>
        <th>Age</th>
        <th>Email</th>
        <th>Remove</th>
        <th>Edit</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let customer of customers">
        <td>{{ customer.name }}</td>
        <td>{{ customer.age }}</td>
        <td>{{ customer.email }}</td>
```



```
        <td><button class="btn btn-danger fa fa-trash-o"></button></td>
        <td>
          <button
            class="btn btn-warning fa fa-pencil-square-o"
            (click)="editClient(customer)"
          ></button>
        </td>
      </tr>
    </tbody>
  </table>
  <!-- Mostrar el modal -->
  <button (click)="openModelDialog()" class="btn btn-primary">
    Add customer
  </button>
</div>
```

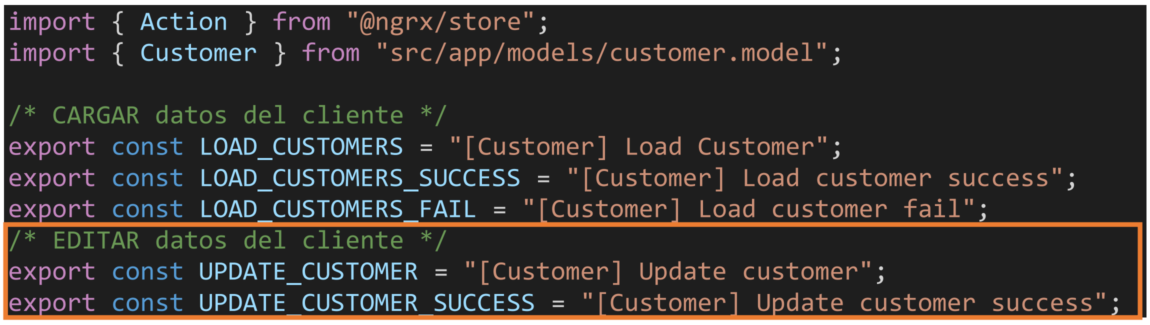
Actualizar customer – UPDATE

Creando el servicio:



```
import { Injectable } from "@angular/core";
import { HttpClient, HttpHeaders } from "@angular/common/http";
import { Customer } from "../models/customer.model";

@Injectable({
  providedIn: "root"
})
export class CustomerService {
  private apiUrl = "http://localhost:3000/usuarios";
  public httpOpt = {
    headers: new HttpHeaders({
      "Content-Type": "application/json",
      Accept: "application/json, text/plain"
    })
  };
  constructor(private http: HttpClient) {}

  getCustomers() {
    return this.http.get<Customer[]>(`${this.apiUrl}`);
  }

  updateCustomer(customer: Customer) {
    return this.http.put(
      `${this.apiUrl}/${customer.id}`,
      JSON.stringify(customer),
      this.httpOpt
    );
  }
}
```

Creando la acción:



```
import { Action } from "@ngrx/store";
import { Customer } from "src/app/models/customer.model";

/* CARGAR datos del cliente */
export const LOAD_CUSTOMERS = "[Customer] Load Customer";
export const LOAD_CUSTOMERS_SUCCESS = "[Customer] Load customer success";
export const LOAD_CUSTOMERS_FAIL = "[Customer] Load customer fail";
/* EDITAR datos del cliente */
export const UPDATE_CUSTOMER = "[Customer] Update customer";
export const UPDATE_CUSTOMER_SUCCESS = "[Customer] Update customer success";
export const UPDATE_CUSTOMER_FAIL = "[Customer] Update customer fail";

// Cagar data
export class LoadCustomer implements Action {
  readonly type = LOAD_CUSTOMERS;
}

export class LoadCustomerSuccess implements Action {
  readonly type = LOAD_CUSTOMERS_SUCCESS;
  constructor(public payload: Customer[]) {}
}

export class LoadCustomerFail implements Action {
  readonly type = LOAD_CUSTOMERS_FAIL;
  constructor(public payload: any) {}
}

// Editar data
export class UpdateCustomer implements Action {
  readonly type = UPDATE_CUSTOMER;
  constructor(public payload: Customer) {}
}

export class UpdateCustomerSuccess implements Action {
  readonly type = UPDATE_CUSTOMER_SUCCESS;
  constructor(public payload: any) {}
}

export class UpdateCustomerFail implements Action {
  readonly type = UPDATE_CUSTOMER_FAIL;
  constructor(public payload: any) {}
}

export type CustomerActions =
  | LoadCustomer
  | LoadCustomerSuccess
  | LoadCustomerFail
  | UpdateCustomer
  | UpdateCustomerSuccess
  | UpdateCustomerFail;
```

Creando el effect para la comunicación:

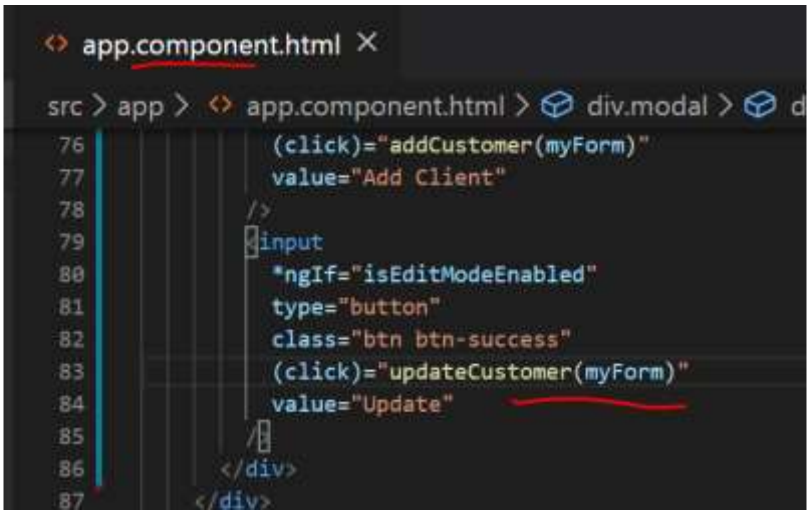
```
import { Injectable } from "@angular/core";
import { Actions, Effect, ofType } from "@ngrx/effects";

import { of, Observable } from "rxjs";
import { catchError, map, switchMap } from "rxjs/operators";

import * as fromCustomersActions from "../actions/customer.action";
import { Action } from "@ngrx/store";
import { CustomerService } from "src/app/services/customer.service";
//import all requried services or any dependencies

@Injectable()
export class CustomerEffects {
  constructor(
    private action$: Actions,
    private customerServie: CustomerService
  ) {}
  // Cargar
  @Effect()
  loadCustomers$: Observable<Action> = this.action$.pipe(
    ofType(fromCustomersActions.LOAD_CUSTOMERS),
    switchMap(() =>
      this.customerServie.getCustomers().pipe(
        map(response => {
          return new fromCustomersActions.LoadCustomerSuccess(response
);
        }),
        catchError(error =>
          of(new fromCustomersActions.LoadCustomerFail(error))
        )
      )
    )
  );

  // Editar
  @Effect()
  updateCustomers$: Observable<Action> = this.action$.pipe(
    ofType(fromCustomersActions.UPDATE_CUSTOMER),
    map((action: fromCustomersActions.UpdateCustomer) => action.payloa
d),
    switchMap(payload =>
      this.customerServie.updateCustomer(payload).pipe(
        map(
          response => new fromCustomersActions.UpdateCustomerSuccess(r
esponse)
        ),
        catchError(error =>
          of(new fromCustomersActions.UpdateCustomerFail(error))
        )
      )
    )
  );
}
```

Agregar el cambio en el reducer:

```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [],
  loaded: false,
  loading: false,
  error: ""
};

// Función reducer
export function reducer(
  state = initState,
  action: fromCustomerActions.CustomerActions
) {
  switch (action.type) {
    case fromCustomerActions.LOAD_CUSTOMERS:
      return { ...state, loading: true };
    case fromCustomerActions.LOAD_CUSTOMERS_SUCCESS:
      const data = action.payload; // Lista de clientes enviada por el api
      return { ...state, loading: false, loaded: true, data: data };
    case fromCustomerActions.LOAD_CUSTOMERS_FAIL:
      return { ...state, loading: false, loaded: false, error: action.payload
 };
    case fromCustomerActions.UPDATE_CUSTOMER_SUCCESS:
      let dataEdit = state.data.map(customer => {
        if (customer.id === action.payload.id) {
          return action.payload;
        } else {
          return customer;
        }
      });
      return {
        ...state,
        loading: false,
        loaded: true,
        data: dataEdit
      };

    default:
      return state;
  }
}

// Creando selector que retorne la propiedad "data"
export const getCustomers = (state: CustomerState) => state.data;
export const getCustomersLoaded = (state: CustomerState) => state.loaded;
export const getCustomersLoading = (state: CustomerState) => state.loading;
export const getCustomersError = (state: CustomerState) => state.error;
```

En el componente:

```
import { Component, OnInit } from "@angular/core";
import { Store } from "@ngrx/store";
import * as fromStore from "./store"; // referencia al archivo index.ts de "s
tore"
import { Customer } from "./models/customer.model";
import { NgForm } from "@angular/forms";

@Component({
  selector: "app-root",
  templateUrl: "./app.component.html",
  styleUrls: ["./app.component.scss"]
})
export class AppComponent implements OnInit {
  customers: Customer[];
  display: string = "none";
  isEditModeEnabled: boolean = false;
  person: Customer = {};

  constructor(private store: Store<fromStore.AppState>) {
    store.select(fromStore.getCustomers).subscribe(response => {
      this.customers = response;
    });

    store.select(fromStore.getCustomerById(2)).subscribe(response => {
      console.log(response);
    });
  }

  ngOnInit(): void {
    this.store.dispatch(new fromStore.LoadCustomer());
  }

  openModelDialog() {
    this.display = "block";
  }

  closeModal(myForm: NgForm) {
    this.display = "none";
  }

  editClient(customer: Customer) {
    this.isEditModeEnabled = true;
    this.person = {...customer}; // Copia del cliente seleccionado
    this.display = "block";
  }

  addCustomer(myForm: NgForm) {}

  updateCustomer(myForm: NgForm) {
    this.store.dispatch(new fromStore.UpdateCustomer(myForm.value));
    this.closeModal(myForm);
  }
}
```



```
          (click)="addCustomer(myForm)"
          value="Add Client"
        />
        <input
          *ngIf="isEditModeEnabled"
          type="button"
          class="btn btn-success"
          (click)="updateCustomer(myForm)"
          value="Update"
        />
      </div>
    </div>
```

Agregar customer – CREATE

Servicio:

```
addCustomer(customer: Customer) {
    return this.http.post(
      `${this.apiUrl}`,
      JSON.stringify(customer),
      this.httpOpt
    );
  }
```

Las acciones:

```
import { Action } from "@ngrx/store";
import { Customer } from "src/app/models/customer.model";

/* CARGAR datos del cliente */
export const LOAD_CUSTOMERS = "[Customer] Load Customer";
export const LOAD_CUSTOMERS_SUCCESS = "[Customer] Load customer success";
export const LOAD_CUSTOMERS_FAIL = "[Customer] Load customer fail";
/* EDITAR datos del cliente */
export const UPDATE_CUSTOMER = "[Customer] Update customer";
export const UPDATE_CUSTOMER_SUCCESS = "[Customer] Update customer success";
export const UPDATE_CUSTOMER_FAIL = "[Customer] Update customer fail";
/* Agregar cliente */
export const ADD_CUSTOMER = "[Customer] Add customer";
export const ADD_CUSTOMER_SUCCESS = "[Customer] Add customer success";
export const ADD_CUSTOMER_FAIL = "[Customer] Add customer fail";

// Cagar data
export class LoadCustomer implements Action {
  readonly type = LOAD_CUSTOMERS;
}

export class LoadCustomerSuccess implements Action {
  readonly type = LOAD_CUSTOMERS_SUCCESS;
  constructor(public payload: Customer[]) {}
}

export class LoadCustomerFail implements Action {
  readonly type = LOAD_CUSTOMERS_FAIL;
  constructor(public payload: any) {}
}

// Editar data
export class UpdateCustomer implements Action {
  readonly type = UPDATE_CUSTOMER;
  constructor(public payload: Customer) {}
}

export class UpdateCustomerSuccess implements Action {
  readonly type = UPDATE_CUSTOMER_SUCCESS;
  constructor(public payload: any) {}
}

export class UpdateCustomerFail implements Action {
  readonly type = UPDATE_CUSTOMER_FAIL;
  constructor(public payload: any) {}
}

// Agregar nuevo cliente
export class AddCustomer implements Action {
  readonly type = ADD_CUSTOMER;
  constructor(public payload: Customer) {}
}

export class AddCustomerSuccess implements Action {
  readonly type = ADD_CUSTOMER_SUCCESS;
  constructor(public payload: any) {}
}

export class AddCustomerFail implements Action {
  readonly type = ADD_CUSTOMER_FAIL;
  constructor(public payload: any) {}
}
```

```
export type CustomerActions =
  | LoadCustomer
  | LoadCustomerSuccess
  | LoadCustomerFail
  | UpdateCustomer
  | UpdateCustomerSuccess
  | UpdateCustomerFail
  | AddCustomer
  | AddCustomerSuccess
  | AddCustomerFail;
```

Effect:

```
 // Agregar
  @Effect()
  addCustomers$: Observable<Action> = this.action$.pipe(
    ofType(fromCustomersActions.ADD_CUSTOMER),
    map((action: fromCustomersActions.AddCustomer) => action.payload),
    switchMap(payload =>
      this.customerServie.addCustomer(payload).pipe(
        map(response => new fromCustomersActions.AddCustomerSuccess(response)
),
        catchError(error => of(new fromCustomersActions.AddCustomerFail(error
)))
      )
    )
  );
```

Agregando el cambio en el reducer:

```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [],
  loaded: false,
  loading: false,
  error: ""
};

// Función reducer
export function reducer(
  state = initState,
  action: fromCustomerActions.CustomerActions
) {
  switch (action.type) {
    case fromCustomerActions.LOAD_CUSTOMERS:
      return { ...state, loading: true };

    case fromCustomerActions.LOAD_CUSTOMERS_SUCCESS:
      const data = action.payload; // Lista de clientes enviada por el api
      return { ...state, loading: false, loaded: true, data: data };
    case fromCustomerActions.LOAD_CUSTOMERS_FAIL:
      return { ...state, loading: false, loaded: false, error: action.payload
 };

    case fromCustomerActions.UPDATE_CUSTOMER_SUCCESS:
      let dataEdit = state.data.map(customer => {
        if (customer.id === action.payload.id) {
          return action.payload;
        } else {
          return customer;
        }
      });
      return {
        ...state,
        loading: false,
        loaded: true,
        data: dataEdit
      };

    case fromCustomerActions.UPDATE_CUSTOMER_FAIL:
      return { ...state, loading: false, loaded: false, error: action.payload
 };

    case fromCustomerActions.ADD_CUSTOMER_SUCCESS:
      return {
        ...state,
        data: [...state.data, action.payload]
      };

    case fromCustomerActions.ADD_CUSTOMER_FAIL:
      return { ...state, error: action.payload };

    default:
      return state;
  }
}

// Creando selector que retorne la propiedad "data"
export const getCustomers = (state: CustomerState) => state.data;
export const getCustomersLoaded = (state: CustomerState) => state.loaded;
export const getCustomersLoading = (state: CustomerState) => state.loading;
export const getCustomersError = (state: CustomerState) => state.error;
```

Función que dispara la acción en el componente:

```
addCustomer(myForm: NgForm) {
    // Generando id del cliente
    let userId = new Date().getTime(); // tiempo en milisegundos
    let newCustomer = myForm.value;
    newCustomer["id"] = userId;
    if (newCustomer.name !== null && newCustomer !== undefined) {
      this.store.dispatch(new fromStore.AddCustomer(newCustomer));
      this.closeModal(myForm);
    }
  }
```

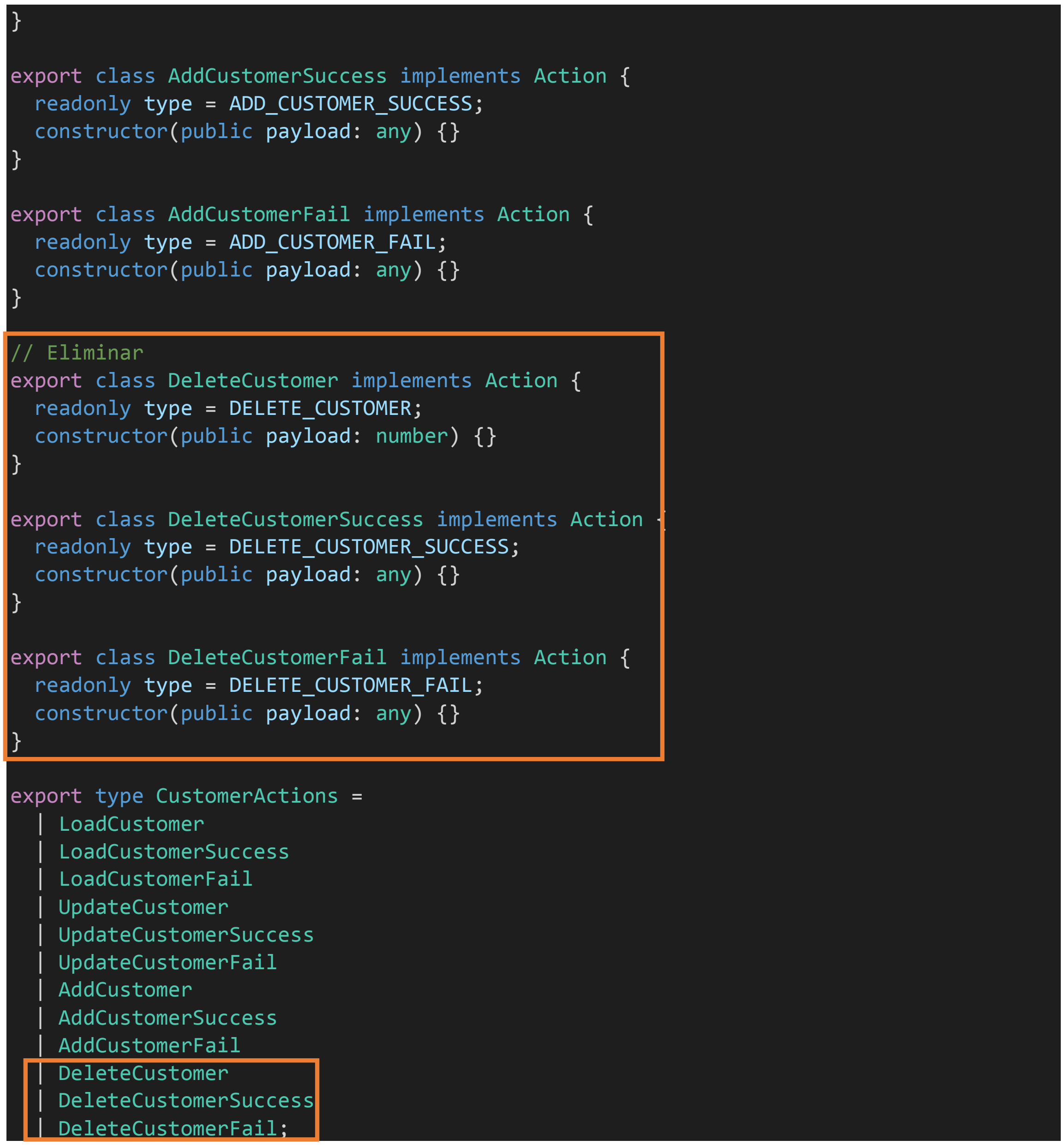
Borrar customer – DELETE

Servicio:

```
deleteCustomer(id: number) {
    return this.http.delete(`${this.apiUrl}/${id}`);
}
```

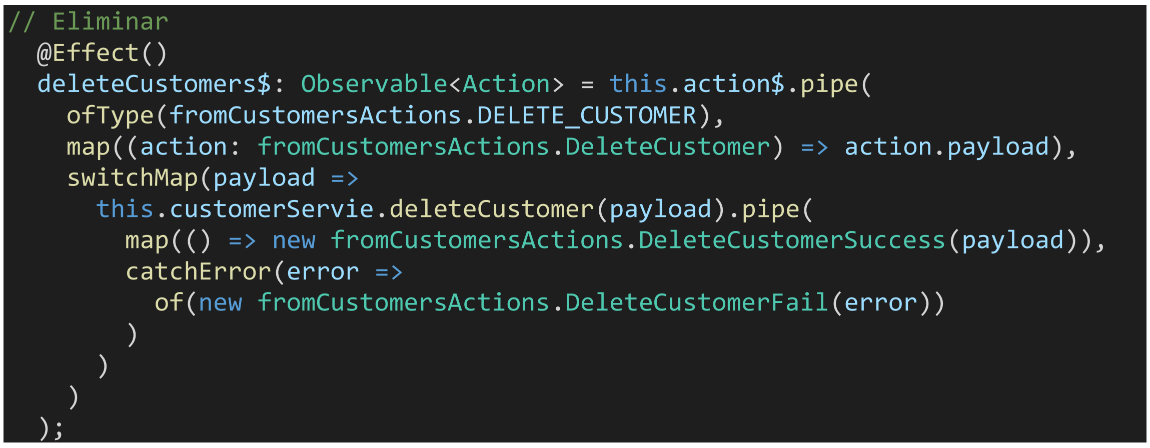
Acciones:

```
import { Action } from "@ngrx/store";
import { Customer } from "src/app/models/customer.model";

/* CARGAR datos del cliente */
export const LOAD_CUSTOMERS = "[Customer] Load Customer";
export const LOAD_CUSTOMERS_SUCCESS = "[Customer] Load customer success";
export const LOAD_CUSTOMERS_FAIL = "[Customer] Load customer fail";
/* EDITAR datos del cliente */
export const UPDATE_CUSTOMER = "[Customer] Update customer";
export const UPDATE_CUSTOMER_SUCCESS = "[Customer] Update customer success";
export const UPDATE_CUSTOMER_FAIL = "[Customer] Update customer fail";
/* Agregar cliente */
export const ADD_CUSTOMER = "[Customer] Add customer";
export const ADD_CUSTOMER_SUCCESS = "[Customer] Add customer success";
export const ADD_CUSTOMER_FAIL = "[Customer] Add customer fail";
/* Delete Customer */
export const DELETE_CUSTOMER = "[Customer] Delete customer";
export const DELETE_CUSTOMER_SUCCESS = "[Customer] Delete customer success";
export const DELETE_CUSTOMER_FAIL = "[Customer] Delete customer fail";

// Cagar data
export class LoadCustomer implements Action {
  readonly type = LOAD_CUSTOMERS;
}

export class LoadCustomerSuccess implements Action {
  readonly type = LOAD_CUSTOMERS_SUCCESS;
  constructor(public payload: Customer[]) {}
}

export class LoadCustomerFail implements Action {
  readonly type = LOAD_CUSTOMERS_FAIL;
  constructor(public payload: any) {}
}

// Editar data
export class UpdateCustomer implements Action {
  readonly type = UPDATE_CUSTOMER;
  constructor(public payload: Customer) {}
}

export class UpdateCustomerSuccess implements Action {
  readonly type = UPDATE_CUSTOMER_SUCCESS;
  constructor(public payload: any) {}
}

export class UpdateCustomerFail implements Action {
  readonly type = UPDATE_CUSTOMER_FAIL;
  constructor(public payload: any) {}
}

// Agregar nuevo cliente
export class AddCustomer implements Action {
  readonly type = ADD_CUSTOMER;
  constructor(public payload: Customer) {}
```



```
}

export class AddCustomerSuccess implements Action {
  readonly type = ADD_CUSTOMER_SUCCESS;
  constructor(public payload: any) {}
}

export class AddCustomerFail implements Action {
  readonly type = ADD_CUSTOMER_FAIL;
  constructor(public payload: any) {}
}

// Eliminar
export class DeleteCustomer implements Action {
  readonly type = DELETE_CUSTOMER;
  constructor(public payload: number) {}
}

export class DeleteCustomerSuccess implements Action {
  readonly type = DELETE_CUSTOMER_SUCCESS;
  constructor(public payload: any) {}
}

export class DeleteCustomerFail implements Action {
  readonly type = DELETE_CUSTOMER_FAIL;
  constructor(public payload: any) {}
}

export type CustomerActions =
  | LoadCustomer
  | LoadCustomerSuccess
  | LoadCustomerFail
  | UpdateCustomer
  | UpdateCustomerSuccess
  | UpdateCustomerFail
  | AddCustomer
  | AddCustomerSuccess
  | AddCustomerFail
  | DeleteCustomer
  | DeleteCustomerSuccess
  | DeleteCustomerFail;
```

Effect:



```
// Eliminar
  @Effect()
  deleteCustomers$: Observable<Action> = this.action$.pipe(
    ofType(fromCustomersActions.DELETE_CUSTOMER),
    map((action: fromCustomersActions.DeleteCustomer) => action.payload),
    switchMap(payload =>
      this.customerServie.deleteCustomer(payload).pipe(
        map(() => new fromCustomersActions.DeleteCustomerSuccess(payload)),
        catchError(error =>
          of(new fromCustomersActions.DeleteCustomerFail(error))
        )
      )
    )
  );
```

Notificar al reducer:

```
import * as fromCustomerActions from "../actions/customer.action";
import { Customer } from "src/app/models/customer.model";

// Modelo para los states
export interface CustomerState {
  data: Customer[]; // Lista de usuarios
  loaded: boolean;
  loading: boolean;
  error: string;
}

export const initState: CustomerState = {
  data: [],
  loaded: false,
  loading: false,
  error: ""
};

// Función reducer
export function reducer(
  state = initState,
  action: fromCustomerActions.CustomerActions
) {
  switch (action.type) {
    case fromCustomerActions.LOAD_CUSTOMERS:
      return { ...state, loading: true };

    case fromCustomerActions.LOAD_CUSTOMERS_SUCCESS:
      const data = action.payload; // Lista de clientes enviada por el api
      return { ...state, loading: false, loaded: true, data: data };

    case fromCustomerActions.LOAD_CUSTOMERS_FAIL:
      return { ...state, loading: false, loaded: false, error: action.payload
 };

    case fromCustomerActions.UPDATE_CUSTOMER_SUCCESS:
      let dataEdit = state.data.map(customer => {
        if (customer.id === action.payload.id) {
          return action.payload;
        } else {
          return customer;
        }
      });
      return {
        ...state,
        loading: false,
        loaded: true,
        data: dataEdit
      };

    case fromCustomerActions.UPDATE_CUSTOMER_FAIL:
      return { ...state, loading: false, loaded: false, error: action.payload
 };

    case fromCustomerActions.ADD_CUSTOMER_SUCCESS:
      return {
        ...state,
        data: [...state.data, action.payload]
      };

    case fromCustomerActions.ADD_CUSTOMER_FAIL:
      return { ...state, error: action.payload };
    case fromCustomerActions.DELETE_CUSTOMER_SUCCESS:
      const userId = action.payload;
      return {
        ...state,
        data: [
          ...state.data.filter(user => {
            user.id !== userId;
          })
        ]
      };

    case fromCustomerActions.DELETE_CUSTOMER_FAIL:
      return { ...state, error: action.payload };

    default:
      return state;
  }
}

// Creando selector que retorne la propiedad "data"
export const getCustomers = (state: CustomerState) => state.data;
export const getCustomersLoaded = (state: CustomerState) => state.loaded;
export const getCustomersLoading = (state: CustomerState) => state.loading;
export const getCustomersError = (state: CustomerState) => state.error;
```

En el componente:

```
deleteClient(customerId) {
    if (customerId !== undefined) {
      if (confirm("¿Estás segur@ de borrar este usuatio?")) {
        this.store.dispatch(new fromStore.DeleteCustomer(customerId));
      }
    }
  }
```

Reset Formulario

En el componente:

```
closeModal(myForm: NgForm) {
    myForm.reset();
    this.display = "none";
  }
```