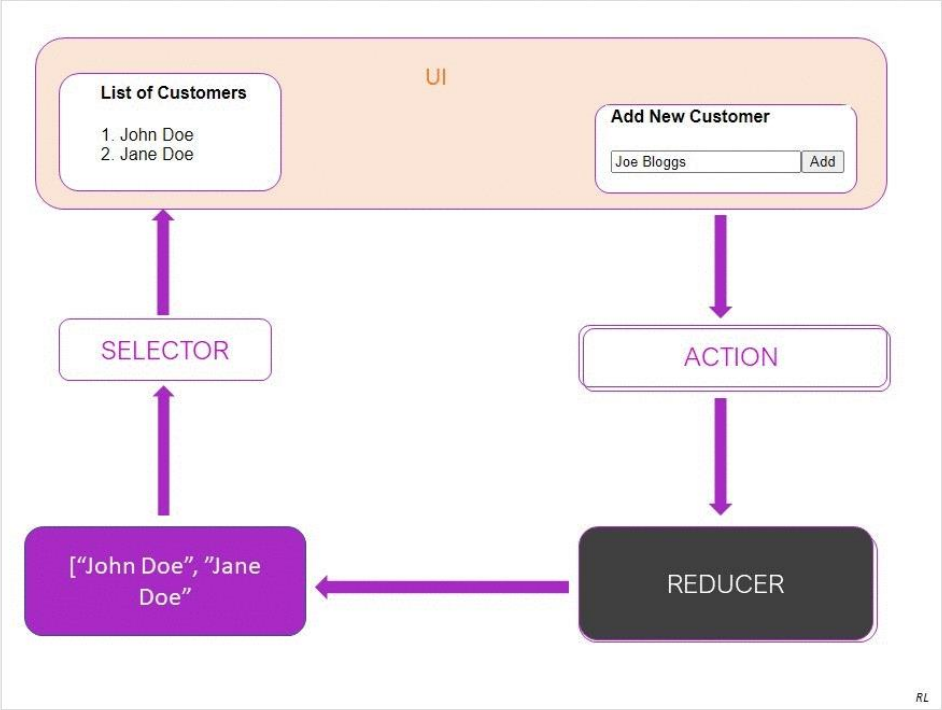
LABORATÓRIO 4

CRIANDO UMA APLICAÇÃO QUE UTILIZA O NGRX

Nesse laboratório, vamos construir uma aplicação que utiliza o controle de Estados para fazer modificações. Como é apenas um laboratório, não vamos fazer a parte em que os Effects se comunicam com a camada de serviço. Após o laboratório, analisaremos o ambiente de desenvolvimento para identificar todos os passos criados nele.

Esse é o esboço do projeto:



Vamos realizar a criação de uma aplicação, acessar a sua pasta e iniciar a sua execução.

$ ng new lab4 --style=scss --routing=false

$ cd lab4

$ ng serve

Vamos instalar o NgRx e as ferramentas:

$ ng add @ngrx/schematics@latest

$ ng config cli.defaultCollection @ngrx/schematics

$ npm install @ngrx/store --save

$ npm install @ngrx/effects --save

$ npm install @ngrx/entity --save

```
$ npm install @ngrx/store-devtools –save
```

O package.json foi atualizado.

Vamos adicionar a store NgRx no nosso aplicativo

```
$ ng generate @ngrx/schematics:store State --root --module app.module.ts
```

Vamos criar um sub módulo com o nome costumer:

```
$ ng generate module Customer
```

Vamos criar uma nova classe, dentro da pasta models, chamada customer:

```
$ ng generate class models/customer
```

Nesse caso, vamos adicionar a propriedade nome nessa classe:

```
$ export class Customer {
    name = '';
  }
```

Vamos adicionar ações. Nesse caso, uma ação para adicionar um customer. Continuamos utilizando o angular CLI:

```
$ ng generate action customer/store/action/Customer
```

Não vamos gerar ações de falha por enquanto. Por isso, vamos selecionar não para essa opção.

Vamos modificar o typescript criado com o seguinte código

```
$ import { createAction, props } from '@ngrx/store';
    import {Customer} from "../../../models/customer";

    export const addCustomers = createAction(
      '[Customer] Add Customers',
      (customer: Customer) => ({customer})
    );
```

Agora, vamos adicionar um reducer. Todas as mudanças de estado acontecem no reducer baseadas na ação selecionada. Se o estado está mudando, então o reducer vai criar um novo customer ao invés de mudar a lista atual de customer. No caso de uma mudança de estado, o reducer sempre vai retornar uma nova lista de objetos customer.

```
$ ng generate reducer customer/store/reducer/Customer
```

Novamente, não vamos adicionar um reducer de sucesso e de falha, por isso a resposta para essa pergunta vai ser não.

Vamos adicionar o seguinte código para o reducer:

```
$ import {Action, createReducer, on} from '@ngrx/store';
    import { Customer } from 'src/app/models/customer';
    import * as CustomerActions from '../action/customer.actions';

    export const customerFeatureKey = 'customer';

    export interface CustomerState {
     customers: Customer[];
    }

    export const initialState: CustomerState = {
     customers: []
    };

    export const customerReducer = createReducer(
     initialState,
     on(CustomerActions.addCustomers,
       (state: CustomerState, {customer}) =>
           ({...state,
             customers: [...state.customers, customer]
           }))
    );

    export function reducer(state: CustomerState | undefined, action: Action): any {
     return customerReducer(state, action);
```

```
}
```

Vamos adicionar o selector.

```
$ ng generate selector customer/store/selector/Customer
```

Modifique o código para o seguinte:

```
$ import {createFeatureSelector, createSelector} from '@ngrx/store';
    import * as fromCustomer from '../reducer/customer.reducer';

    export const selectCustomerState =
    createFeatureSelector<fromCustomer.CustomerState>(
    fromCustomer.customerFeatureKey,
    );

    export const selectCustomers = createSelector(
    selectCustomerState,
     (state: fromCustomer.CustomerState) => state.customers
    );
```

Perfeito. Construímos, com isso, a nossa estrutura de Store.

Agora, vamos adicionar um componente para a visualização do customer.

```
    $ ng generate component customer/CustomerView
```

Observem que essa nova visualização foi criada dentro do sub módulo customer.

Adicionando códigos nessa visualização:

```
$ import { Component } from '@angular/core';
    import {CustomerState} from "../store/reducer/customer.reducer";
    import {Store, select} from "@ngrx/store";
    import {selectCustomers} from "../store/selector/customer.selectors";
    import {Customer} from "../../models/customer";
    import {Observable} from "rxjs";

    @Component({
```

```
  selector: 'app-customer-view',
  templateUrl: './customer-view.component.html',
  styleUrls: ['./customer-view.component.scss']
})
export class CustomerViewComponent {

  customers$: Observable<Customer[]>;

  constructor(private store: Store<CustomerState>) {
    this.customers$ = this.store.pipe(select(selectCustomers))
  }

}
```

Também vamos adicionar códigos no template correspondente:

```
$       <h4>List of Customers</h4>
        <div *ngFor="let customer of customers$ | async; let i = index">
          <span>{{i+1}}</span> {{customer.name}}
        </div>
```

Agora, vamos gerar controles para adicionar novos customers.

```
$       ng generate component customer/CustomerAdd
```

O componente deve ficar com esses códigos:

```
$       export class CustomerAddComponent {

          constructor(private store: Store<CustomerState>) {
          }

          addCustomer(customerName: string): void {
            const customer = new Customer();
            customer.name = customerName;
```

```
    this.store.dispatch(addCustomers(customer));
  }


 }
```

E o template com os seguintes códigos:

```
$       <h4>Add New Customer</h4>
        <input #box >
        <button (click)="addCustomer(box.value)">Add</button>
```

Adicionar o StoreModule no sub módulo do Customer

```
$       import { NgModule } from '@angular/core';
        import { CommonModule } from '@angular/common';
        import { CustomerViewComponent } from './customer-view/customer-
        view.component';
        import { CustomerAddComponent } from './customer-add/customer-add.component';
        import {customerFeatureKey, reducer} from "./store/reducer/customer.reducer";
        import {StoreModule} from "@ngrx/store";


        @NgModule({
         declarations: [
           CustomerViewComponent,
           CustomerAddComponent
         ],
         imports: [
           CommonModule,
           StoreModule.forFeature(customerFeatureKey, reducer),
         ],
         exports: [CustomerViewComponent,CustomerAddComponent]
        })
        export class CustomerModule { }
```

Após isso, temos que importar esse CostumerModule também no AppModule

```
$       imports: [
                    BrowserModule,
                    StoreModule.forRoot(reducers, { metaReducers }),
                    StoreDevtoolsModule.instrument(),
                    CustomerModule
                  ],
```

Atualize o AppComponent para que ele mostre tanto a visualização dos customers e a ferramenta de edição:

```
$       <div style="text-align:center">
                  <h1>
                    Welcome to {{ title }}!
                  </h1>
                </div>


                <app-customer-view></app-customer-view>
                <app-customer-add></app-customer-add>
```

Depois disso, basta atualizar o app. Podemos abrir o dev-tools do chrome e verificar os estados da aplicação.