Intro to NgRx

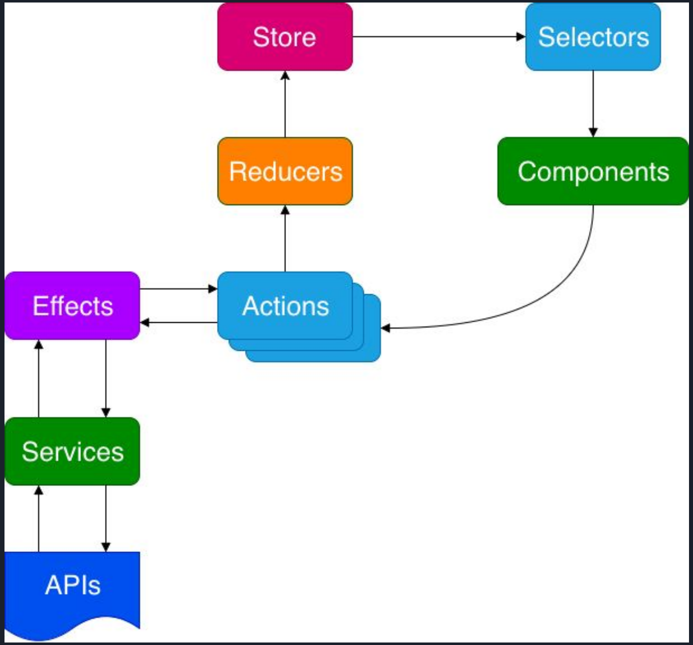
By: Andrew Evans

Why do we need NgRx?

- State = anything in your application
- Large applications require patterns
- State management can be difficult as applications scale
- Uniform method to handle state change

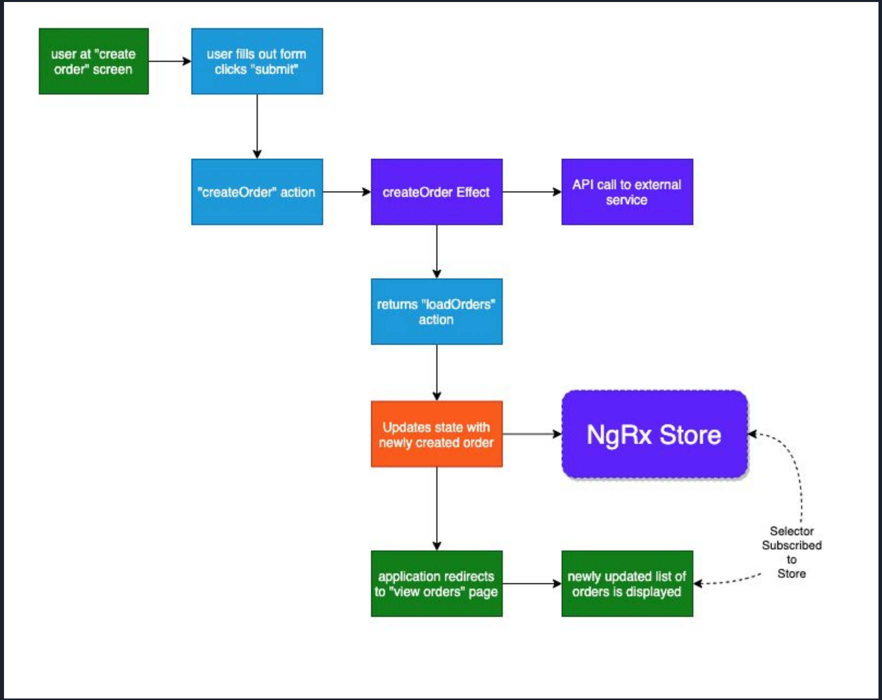
How does it work?

- State is Immutable (never changes only recreated)
- Store = holds state
- Actions = trigger events
- Reducers = tied to action, handle state change
- Effects = external API calls return new actions to generate new State
- Selectors = how you can retrieve slices of state from the Store



Example Flow

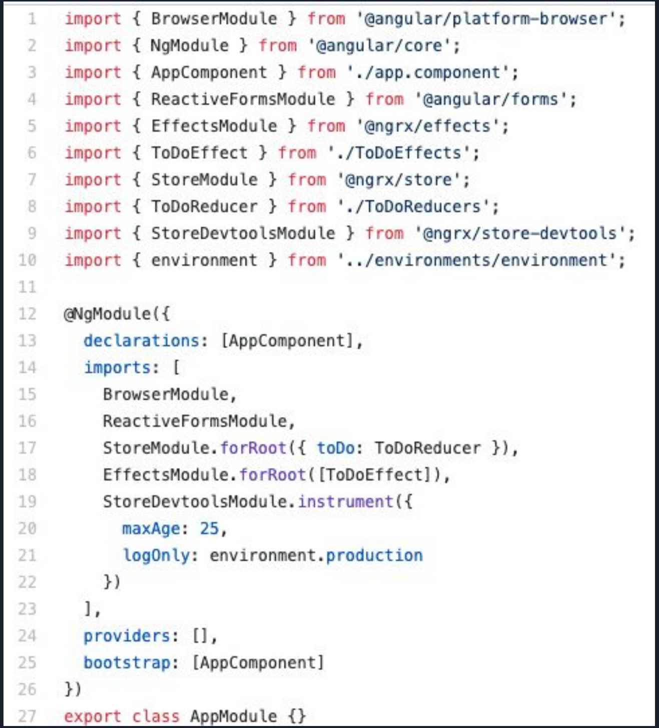
- User creates an order
- User clicks “submit”
- Angular Component dispatches “createOrder” action to store
- Action triggers effect which takes order and sends to the external API to formally “create an order”
- Triggered effect calls an action to loadOrders action
- loadOrders triggers effect to call effect to call API to get newly updated order list
- Component displaying orders is updated just by subscribing to the Orders Selector



user at "create order" screen
user fills out form clicks "submit"
"createOrder" action
createOrder Effect
API call to external service
returns "loadOrders" action
Updates state with newly created order
NgRx Store
Selector Subscribed to Store
application redirects to "view orders" page
newly updated list of orders is displayed

What does it look like in Angular?

- Import NgRx libraries
- App.module registers store and effects use
- Define actions, reducers, and effects in files alongside components
- Can implement “root state” or “feature state”

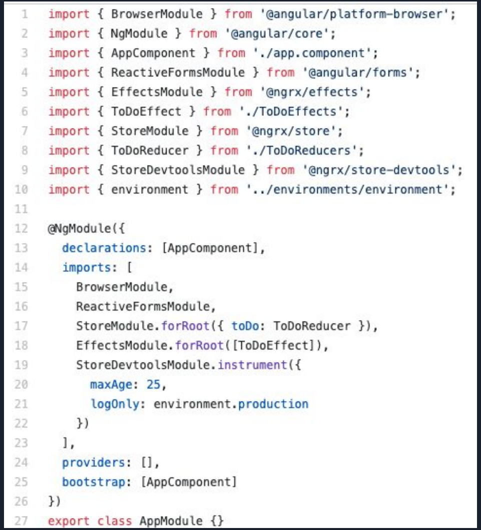


```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';
import { EffectsModule } from '@ngrx/effects';
import { ToDoEffect } from './ToDoEffects';
import { StoreModule } from '@ngrx/store';
import { ToDoReducer } from './ToDoReducers';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { environment } from '../environments/environment';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    StoreModule.forRoot({ toDo: ToDoReducer }),
    EffectsModule.forRoot([ToDoEffect]),
    StoreDevtoolsModule.instrument({
      maxAge: 25,
      logOnly: environment.production
    })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

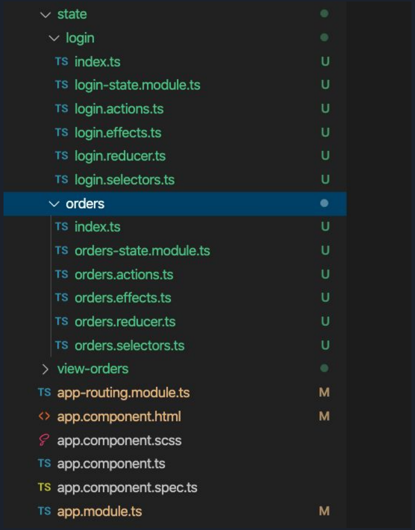
Root State vs. Feature State

- Root State = register everything in the app.module (project root)
- Feature State = create independent definitions of stores by feature



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { ReactiveFormsModule } from '@angular/forms';
import { EffectsModule } from '@ngrx/effects';
import { ToDoEffect } from './ToDoEffects';
import { StoreModule } from '@ngrx/store';
import { ToDoReducer } from './ToDoReducers';
import { StoreDevtoolsModule } from '@ngrx/store-devtools';
import { environment } from '../environments/environment';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    StoreModule.forRoot({ toDo: ToDoReducer }),
    EffectsModule.forRoot([ToDoEffect]),
    StoreDevtoolsModule.instrument({
      maxAge: 25,
      logOnly: environment.production
    })
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule {}
```



```
∨ state                              ●
  ∨ login                            ●
    TS index.ts                      U
    TS login-state.module.ts         U
    TS login.actions.ts              U
    TS login.effects.ts              U
    TS login.reducer.ts              U
    TS login.selectors.ts            U
  ∨ orders                           ●
    TS index.ts                      U
    TS orders-state.module.ts        U
    TS orders.actions.ts             U
    TS orders.effects.ts             U
    TS orders.reducer.ts             U
    TS orders.selectors.ts           U
  > view-orders                      ●
TS app-routing.module.ts             M
<> app.component.html                M
   app.component.scss
TS app.component.ts
TS app.component.spec.ts
TS app.module.ts                     M
```

LIVE CODING

Questions