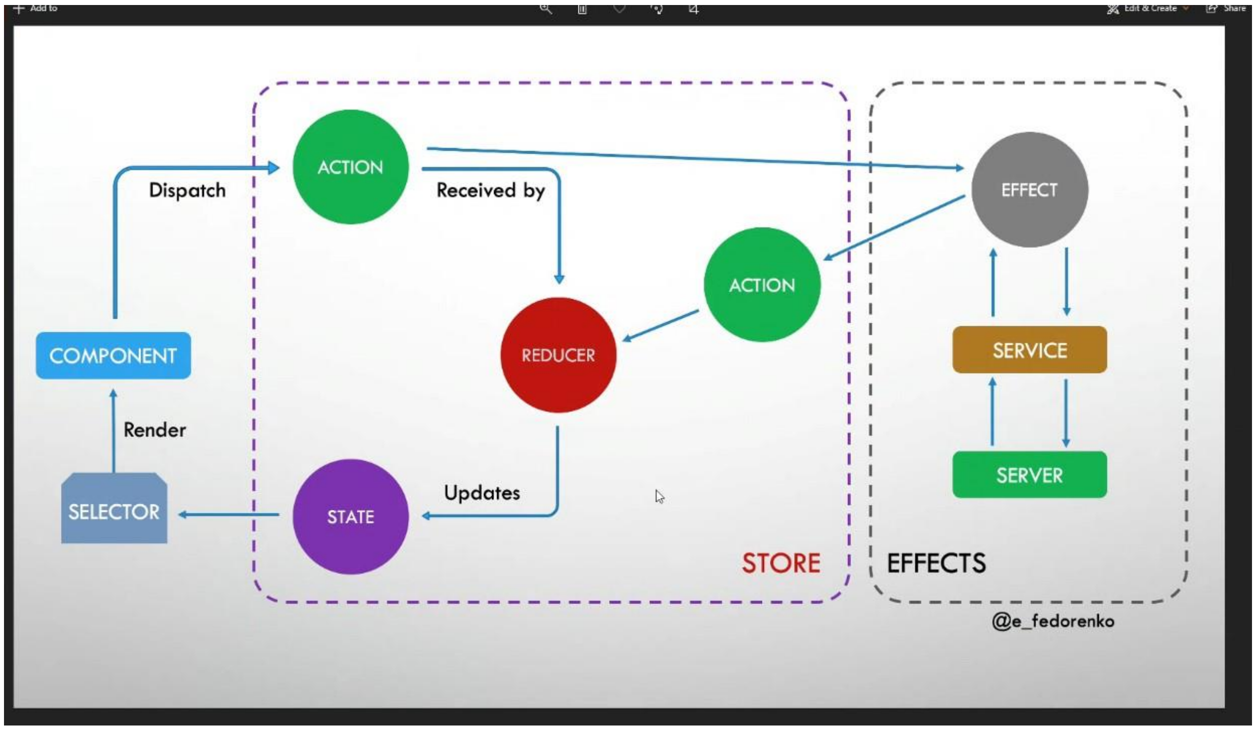
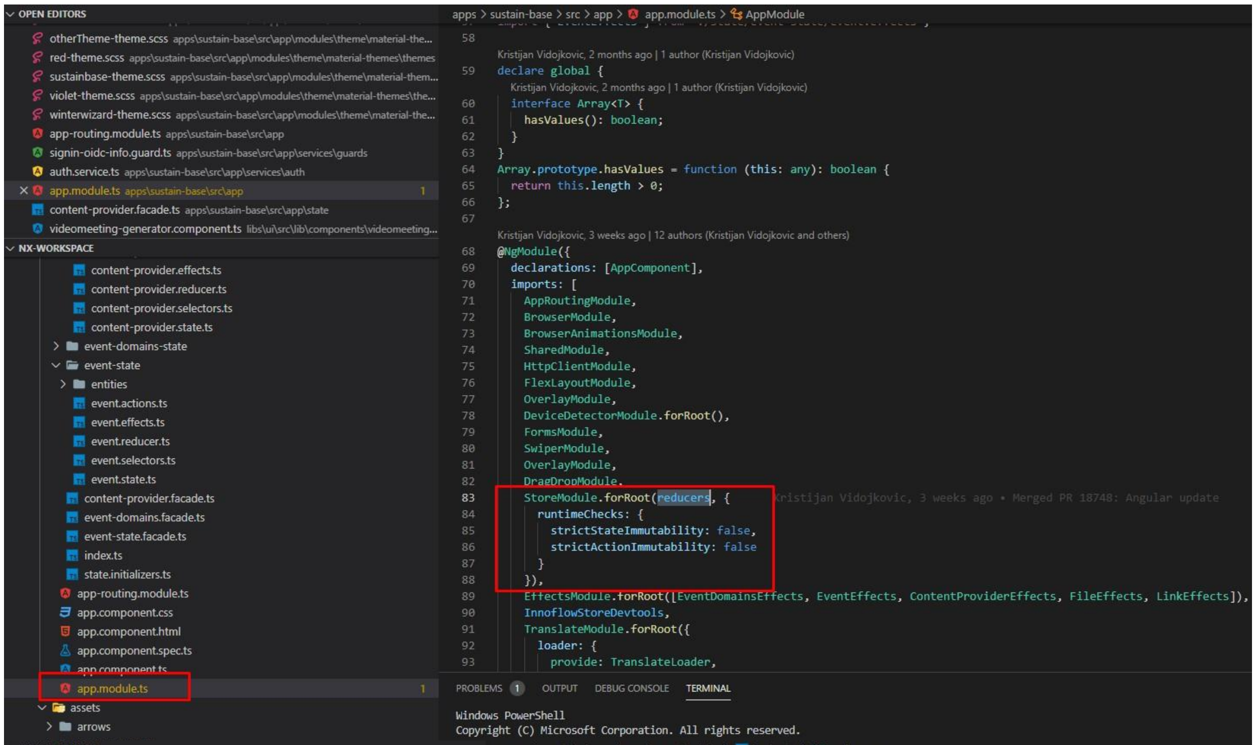

ACTION
Dispatch
Received by
EFFECT
ACTION
COMPONENT
REDUCER
SERVICE
Render
SERVER
SELECTOR
Updates
STATE
STORE
EFFECTS
@e_fedorenko



OPEN EDITORS

- otherTheme-theme.scss apps\sustain-base\src\app\modules\theme\material-the...
- red-theme.scss apps\sustain-base\src\app\modules\theme\material-themes\themes
- sustainbase-theme.scss apps\sustain-base\src\app\modules\theme\material-them...
- violet-theme.scss apps\sustain-base\src\app\modules\theme\material-themes\the...
- winterwizard-theme.scss apps\sustain-base\src\app\modules\theme\material-the...
- app-routing.module.ts apps\sustain-base\src\app
- signin-oidc-info.guard.ts apps\sustain-base\src\app\services\guards
- auth.service.ts apps\sustain-base\src\app\services\auth
- app.module.ts apps\sustain-base\src\app 1
- content-provider.facade.ts apps\sustain-base\src\app\state
- videomeeting-generator.component.ts libs\ui\src\lib\components\videomeeting...

NX-WORKSPACE

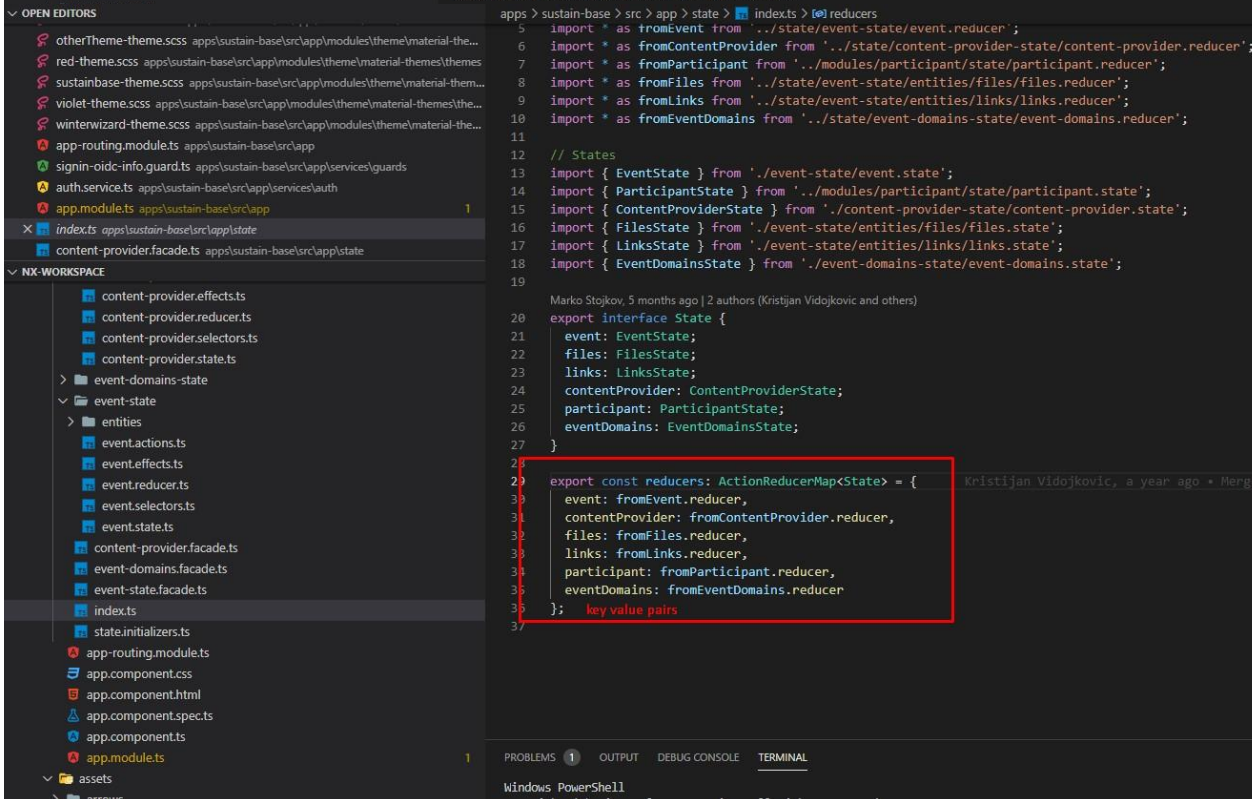
- content-provider.effects.ts
- content-provider.reducer.ts
- content-provider.selectors.ts
- content-provider.state.ts
- event-domains-state
- event-state
  - entities
  - event.actions.ts
  - event.effects.ts
  - event.reducer.ts
  - event.selectors.ts
  - event.state.ts
- content-provider.facade.ts
- event-domains.facade.ts
- event-state.facade.ts
- index.ts
- state.initializers.ts
- app-routing.module.ts
- app.component.css
- app.component.html
- app.component.spec.ts
- app.component.ts
- app.module.ts 1
- assets
  - arrows

apps > sustain-base > src > app > app.module.ts > AppModule

```
Kristijan Vidojkovic, 2 months ago | 1 author (Kristijan Vidojkovic)
declare global {
  Kristijan Vidojkovic, 2 months ago | 1 author (Kristijan Vidojkovic)
  interface Array<T> {
    hasValues(): boolean;
  }
}
Array.prototype.hasValues = function (this: any): boolean {
  return this.length > 0;
};

Kristijan Vidojkovic, 3 weeks ago | 12 authors (Kristijan Vidojkovic and others)
@NgModule({
  declarations: [AppComponent],
  imports: [
    AppRoutingModule,
    BrowserModule,
    BrowserAnimationsModule,
    SharedModule,
    HttpClientModule,
    FlexLayoutModule,
    OverlayModule,
    DeviceDetectorModule.forRoot(),
    FormsModule,
    SwiperModule,
    OverlayModule,
    DragDropModule,
    StoreModule.forRoot(reducers, {    Kristijan Vidojkovic, 3 weeks ago • Merged PR 18748: Angular update
      runtimeChecks: {
        strictStateImmutability: false,
        strictActionImmutability: false
      }
    }),
    EffectsModule.forRoot([EventDomainsEffects, EventEffects, ContentProviderEffects, FileEffects, LinkEffects]),
    InnoflowStoreDevtools,
    TranslateModule.forRoot({
      loader: {
        provide: TranslateLoader,
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```



OPEN EDITORS

- otherTheme-theme.scss apps\sustain-base\src\app\modules\theme\material-the...
- red-theme.scss apps\sustain-base\src\app\modules\theme\material-themes\themes
- sustainbase-theme.scss apps\sustain-base\src\app\modules\theme\material-them...
- violet-theme.scss apps\sustain-base\src\app\modules\theme\material-themes\the...
- winterwizard-theme.scss apps\sustain-base\src\app\modules\theme\material-the...
- app-routing.module.ts apps\sustain-base\src\app
- signin-oidc-info.guard.ts apps\sustain-base\src\app\services\guards
- auth.service.ts apps\sustain-base\src\app\services\auth
- app.module.ts apps\sustain-base\src\app 1
- index.ts apps\sustain-base\src\app\state
- content-provider.facade.ts apps\sustain-base\src\app\state

NX-WORKSPACE

- content-provider.effects.ts
- content-provider.reducer.ts
- content-provider.selectors.ts
- content-provider.state.ts
- event-domains-state
- event-state
  - entities
  - event.actions.ts
  - event.effects.ts
  - event.reducer.ts
  - event.selectors.ts
  - event.state.ts
- content-provider.facade.ts
- event-domains.facade.ts
- event-state.facade.ts
- index.ts
- state.initializers.ts
- app-routing.module.ts
- app.component.css
- app.component.html
- app.component.spec.ts
- app.component.ts
- app.module.ts 1
- assets

apps > sustain-base > src > app > state > index.ts > reducers

```
import * as fromEvent from '../state/event-state/event.reducer';
import * as fromContentProvider from '../state/content-provider-state/content-provider.reducer';
import * as fromParticipant from '../modules/participant/state/participant.reducer';
import * as fromFiles from '../state/event-state/entities/files/files.reducer';
import * as fromLinks from '../state/event-state/entities/links/links.reducer';
import * as fromEventDomains from '../state/event-domains-state/event-domains.reducer';

// States
import { EventState } from './event-state/event.state';
import { ParticipantState } from '../modules/participant/state/participant.state';
import { ContentProviderState } from './content-provider-state/content-provider.state';
import { FilesState } from './event-state/entities/files/files.state';
import { LinksState } from './event-state/entities/links/links.state';
import { EventDomainsState } from './event-domains-state/event-domains.state';

Marko Stojkov, 5 months ago | 2 authors (Kristijan Vidojkovic and others)
export interface State {
  event: EventState;
  files: FilesState;
  links: LinksState;
  contentProvider: ContentProviderState;
  participant: ParticipantState;
  eventDomains: EventDomainsState;
}

export const reducers: ActionReducerMap<State> = {    Kristijan Vidojkovic, a year ago • Merg
  event: fromEvent.reducer,
  contentProvider: fromContentProvider.reducer,
  files: fromFiles.reducer,
  links: fromLinks.reducer,
  participant: fromParticipant.reducer,
  eventDomains: fromEventDomains.reducer
};  key value pairs
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
```

Actions

Actions are events that happen as a result of user interaction with the application. Using our `podcastId` state example above, a user could select a specific podcast and the application would filter based on this specific podcast.
The `Action` class always includes a `type` property representing the action being dispatched.

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

export enum ActionTypes {
  SELECT_PODCAST = '[alsoa.ui.podcast.component] SELECT_PODCAST',
  REQUEST_FAILURE = '[alsoa.ui.podcast.component] REQUEST_FAILURE'
}

export class SelectPodcastAction implements Action {
  public readonly type = ActionTypes.SELECT_PODCAST;
  constructor(public payload: string) { }
}

export class RequestFailureAction implements Action {
  public readonly type = ActionTypes.REQUEST_FAILURE;
}

export type Actions = SelectPodcastAction
  | RequestFailureAction;
```

The class `SelectPodcastAction` includes a type property of `[alsoa.ui.podcast.component] SELECT_PODCAST'`, along with `payload` property of type string. The payload represents the action data associated with the action necessary to complete the action.

Reducer

The reducer generates a new state based on the action dispatched and any payload information contained within the action. These files contain a `switch` statement for any action that changes and returns the new state. Get familiar with the spread syntax as a mechanism to preserve immutability.

```
import { initialState, State } from './state';
import { Actions, ActionTypes } from './actions';

const {
  SELECT_PODCAST,
  REQUEST_FAILURE
} = ActionTypes;

export function featureReducer(state: State = initialState, action: Actions) {
  switch (action.type) {
    case SELECT_PODCAST:
      return {
        ...state,
        podcastId: action.payload
      };
    case REQUEST_FAILURE:
    default:
      return state;
  }
};
```

Declaring an action creator

Without additional metadata:

```
export const increment = createAction('[Counter] Increment');
```

With additional metadata:

```
export const loginSuccess = createAction(
  '[Auth/API] Login Success',
  props<{ user: User }>()
);
```

With a function:

```
export const loginSuccess = createAction(
  '[Auth/API] Login Success',
  (response: Response) => response.user
);
```

Dispatching an action

Without additional metadata:

```
store.dispatch(increment());
```

With additional metadata:

```
store.dispatch(loginSuccess({ user: newUser }));
```

Referencing an action in a reducer

Using a switch statement:

```
switch (action.type) {
  // ...
  case AuthApiActions.loginSuccess.type: {
    return {
      ...state,
      user: action.user
    };
  }
}
```

Using a reducer creator:

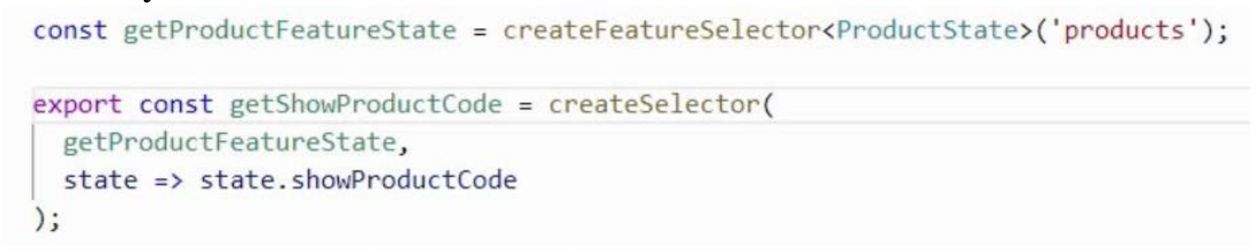
```
on(AuthApiActions.loginSuccess, (state, { user }) => ({ ...state,
user }))
```

Referencing an action in an effect

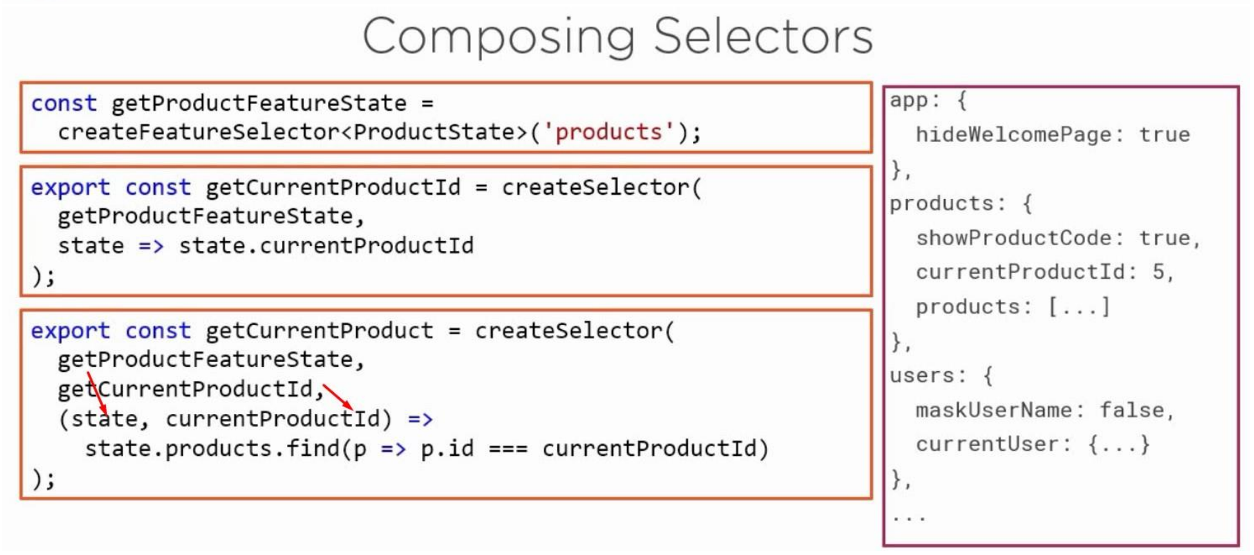
```
effectName$ = createEffect(
  () => this.actions$.pipe(
    ofType(AuthApiActions.loginSuccess),
    // ...
  ));
```

Selectors

Selectors provide a method to read slices of the state.
Combining selectors to retrieve slices of state and filtering through necessary data can also be achieved.



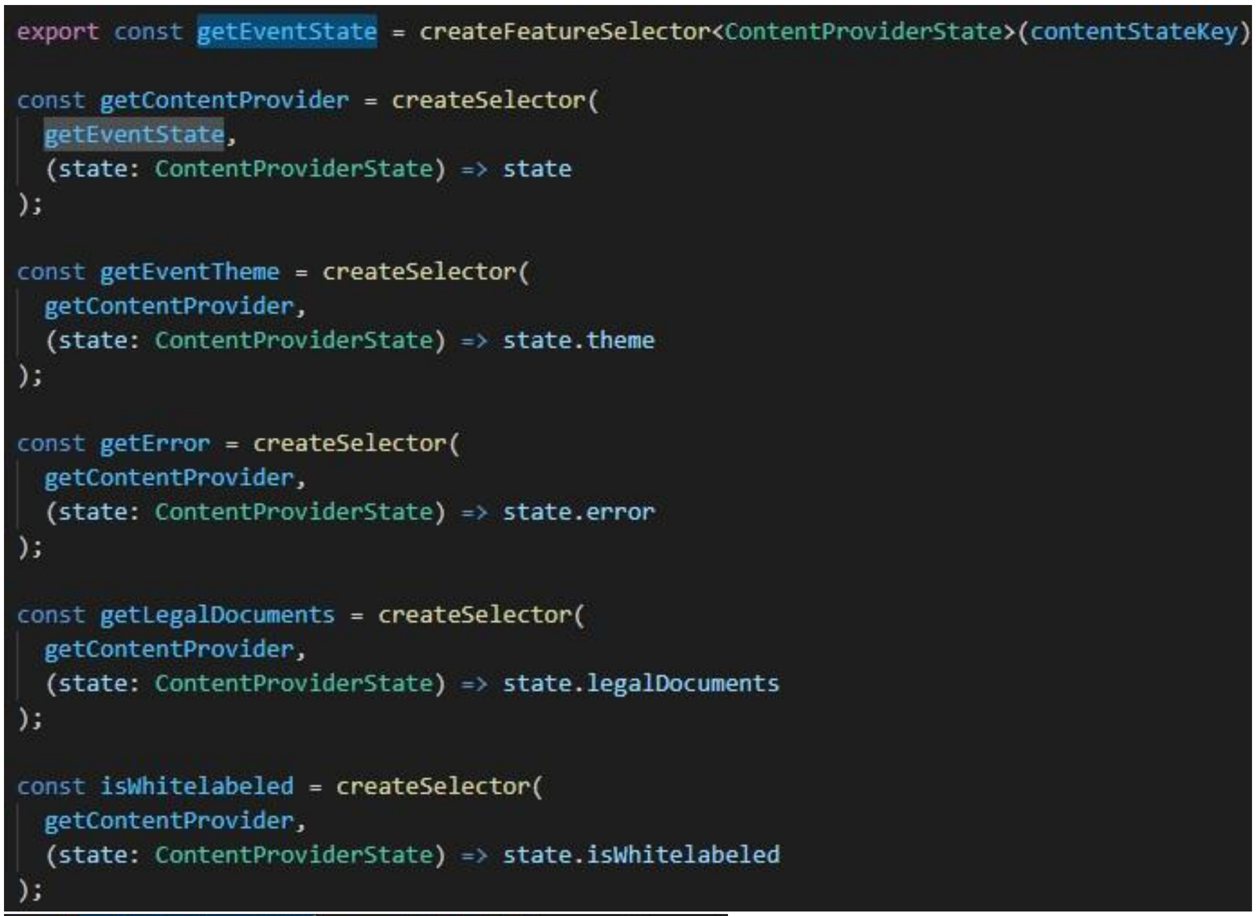
```
const getProductFeatureState = createFeatureSelector<ProductState>('products');

export const getShowProductCode = createSelector(
  getProductFeatureState,
  state => state.showProductCode
);
```



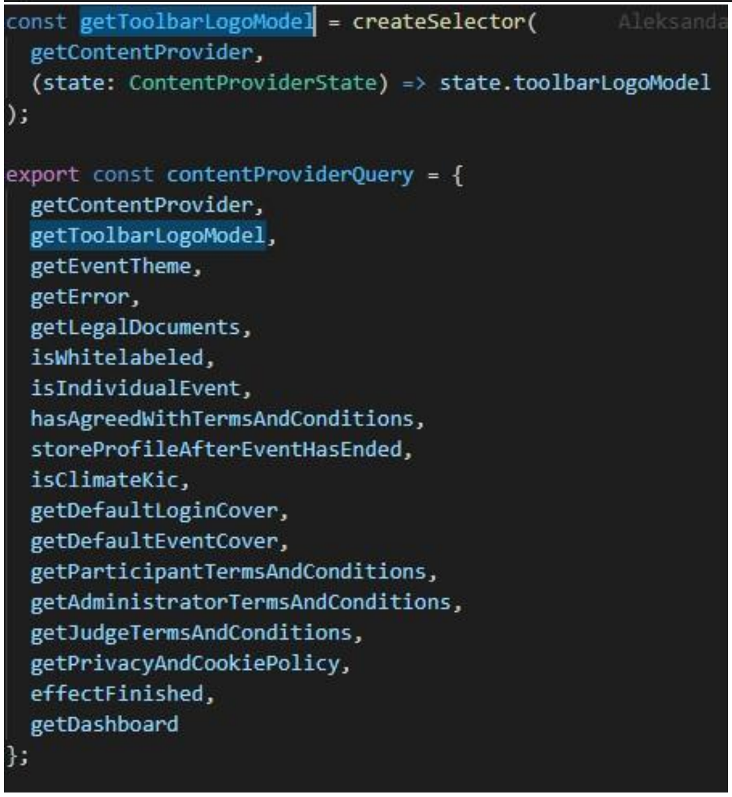
Composing Selectors

```
const getProductFeatureState =
  createFeatureSelector<ProductState>('products');

export const getCurrentProductId = createSelector(
  getProductFeatureState,
  state => state.currentProductId
);

export const getCurrentProduct = createSelector(
  getProductFeatureState,
  getCurrentProductId,
  (state, currentProductId) =>
    state.products.find(p => p.id === currentProductId)
);
```

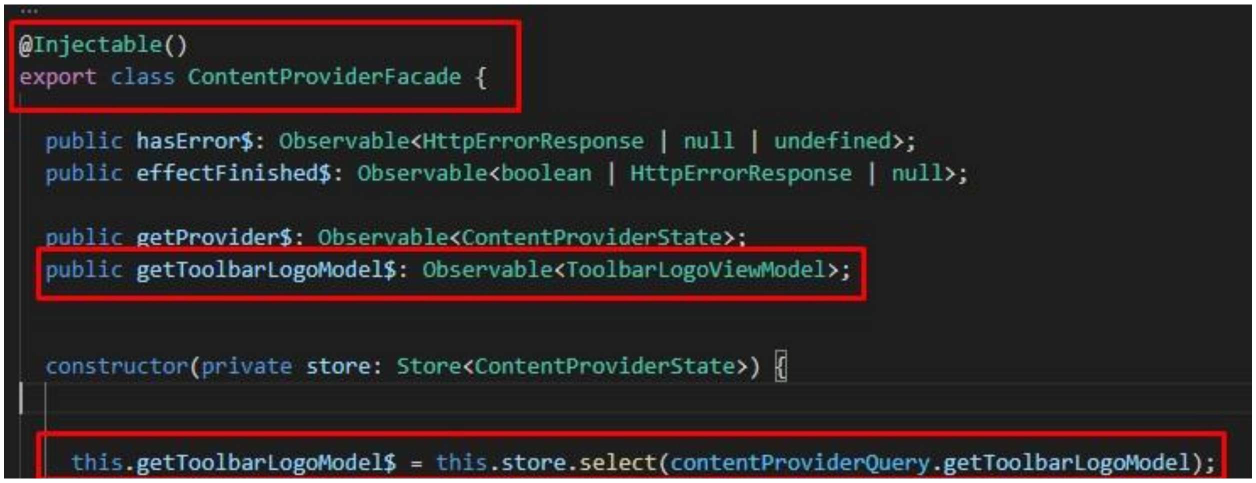
```
app: {
  hideWelcomePage: true
},
products: {
  showProductCode: true,
  currentProductId: 5,
  products: [...]
},
users: {
  maskUserName: false,
  currentUser: {...}
},
...
```



```
export const getEventState = createFeatureSelector<ContentProviderState>(contentStateKey)

const getContentProvider = createSelector(
  getEventState,
  (state: ContentProviderState) => state
);

const getEventTheme = createSelector(
  getContentProvider,
  (state: ContentProviderState) => state.theme
);

const getError = createSelector(
  getContentProvider,
  (state: ContentProviderState) => state.error
);

const getLegalDocuments = createSelector(
  getContentProvider,
  (state: ContentProviderState) => state.legalDocuments
);

const isWhitelabeled = createSelector(
  getContentProvider,
  (state: ContentProviderState) => state.isWhitelabeled
);
```



```
const getToolbarLogoModel = createSelector(        Aleksanda
  getContentProvider,
  (state: ContentProviderState) => state.toolbarLogoModel
);

export const contentProviderQuery = {
  getContentProvider,
  getToolbarLogoModel,
  getEventTheme,
  getError,
  getLegalDocuments,
  isWhitelabeled,
  isIndividualEvent,
  hasAgreedWithTermsAndConditions,
  storeProfileAfterEventHasEnded,
  isClimateKic,
  getDefaultLoginCover,
  getDefaultEventCover,
  getParticipantTermsAndConditions,
  getAdministratorTermsAndConditions,
  getJudgeTermsAndConditions,
  getPrivacyAndCookiePolicy,
  effectFinished,
  getDashboard
};
```



```
...
@Injectable()
export class ContentProviderFacade {

  public hasError$: Observable<HttpErrorResponse | null | undefined>;
  public effectFinished$: Observable<boolean | HttpErrorResponse | null>;

  public getProvider$: Observable<ContentProviderState>;
  public getToolbarLogoModel$: Observable<ToolbarLogoViewModel>;


  constructor(private store: Store<ContentProviderState>) {


    this.getToolbarLogoModel$ = this.store.select(contentProviderQuery.getToolbarLogoModel);
```

Effects

Effects exist to change or retrieve the state of an external system. For most of my use cases, effects communicate with a REST endpoint to query, insert, update, and delete different entities. Effects begin listening immediately for one or multiple actions.