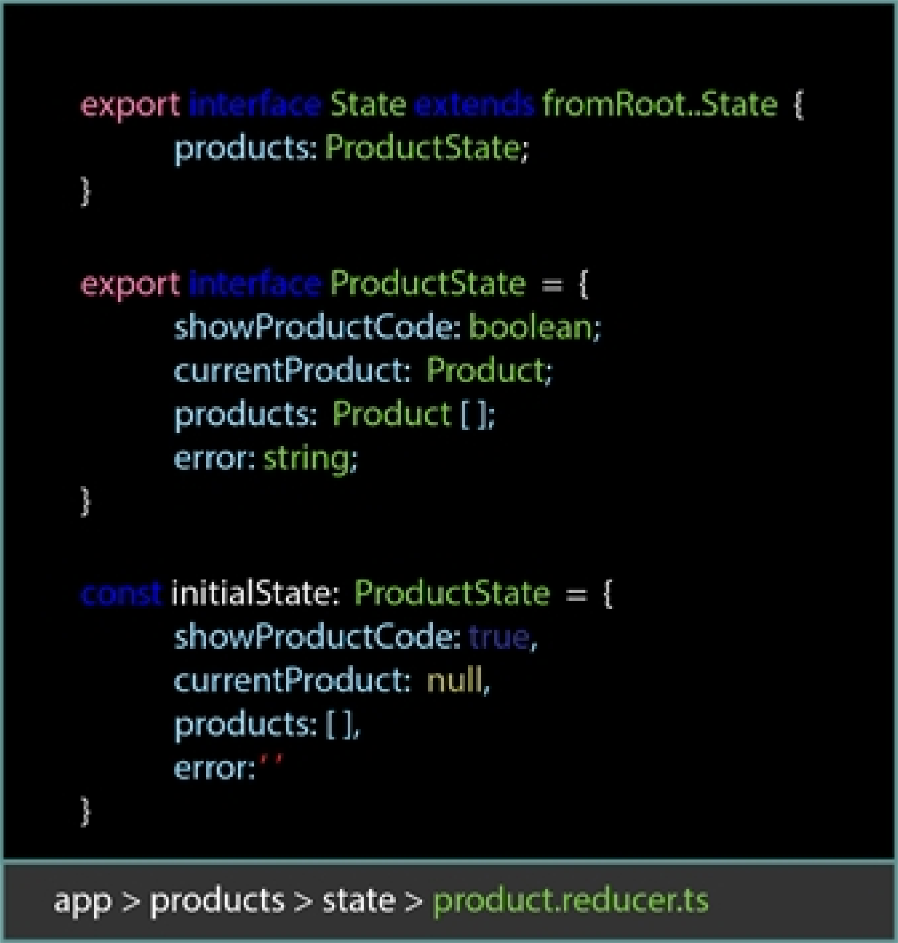
Strong Typing - Feature State Management

One of the great advantages of working with Typescript is it's ability to make your JavaScript Type safe. This adds an extra layer of protection as your data passed and saved to your client-side database (store) is checked for type compatibility. When you first define the state properties they will have the initial value of undefined, so we want to also provide the initial values.   This definition of your slice is located in your feature state's reducer.ts located in the state folder of your feature. Declaration of the slice includes the definition of the data type

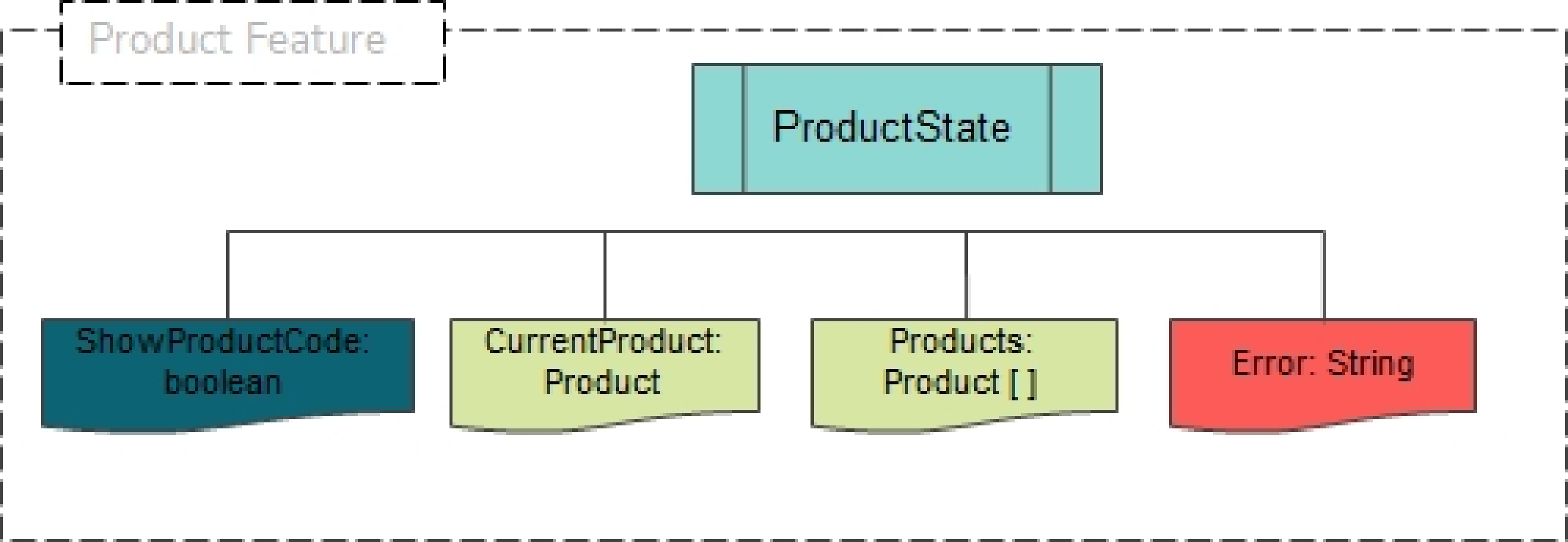


```
export interface State extends fromRoot..State {
        products: ProductState;
}

export interface ProductState = {
        showProductCode: boolean;
        currentProduct:  Product;
        products:  Product [ ];
        error: string;
}

const initialState:  ProductState = {
        showProductCode: true,
        currentProduct:  null,
        products: [ ],
        error:''
}
```

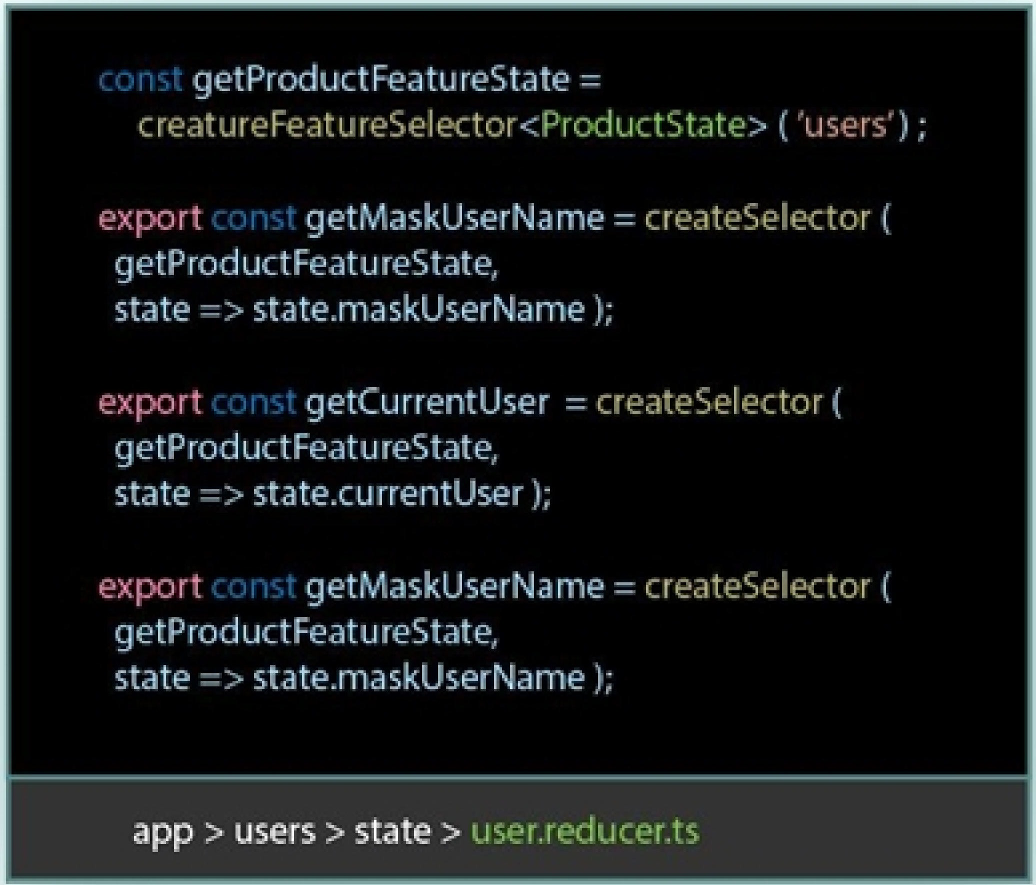
app > products > state > product.reducer.ts


Product Feature
ProductState
ShowProductCode: boolean
CurrentProduct: Product
Products: Product [ ]
Error: String

AppState operates similarly to lazy loading with the router. As you only load what state is critical for the app to function and then state slice corresponding to the feature is added and loaded when the feature is loaded.

Creating Selectors

Also declared in the feature reducer file are the selectors. The selectors are pure functions that are defined in the reducer.ts file and used by the components and services to listen to the state and convey any changes to all subscribed parties. In this example we are defining selectors that will access the state values for the username, currentUser, or whether the user is authenticated. This can be very helpful because you can monitor the user's authentication and control the user's access across the entire site through the immutable state.   In the code below a feature selector is created and all the selectors are attached, almost as if you are creating a domain in which the selectors can be found.



```
const getProductFeatureState =
   creatureFeatureSelector<ProductState> ( 'users' ) ;

export const getMaskUserName = createSelector (
 getProductFeatureState,
 state => state.maskUserName );

export const getCurrentUser  = createSelector (
 getProductFeatureState,
 state => state.currentUser );

export const getMaskUserName = createSelector (
 getProductFeatureState,
 state => state.maskUserName );
```

app > users > state > user.reducer.ts

If the status of the user's authentication changes at any time the store is updated via NgRx actions and immediately all subscribed parties are notified. The process for changing the store's data begins with NgRx actions.

Actions