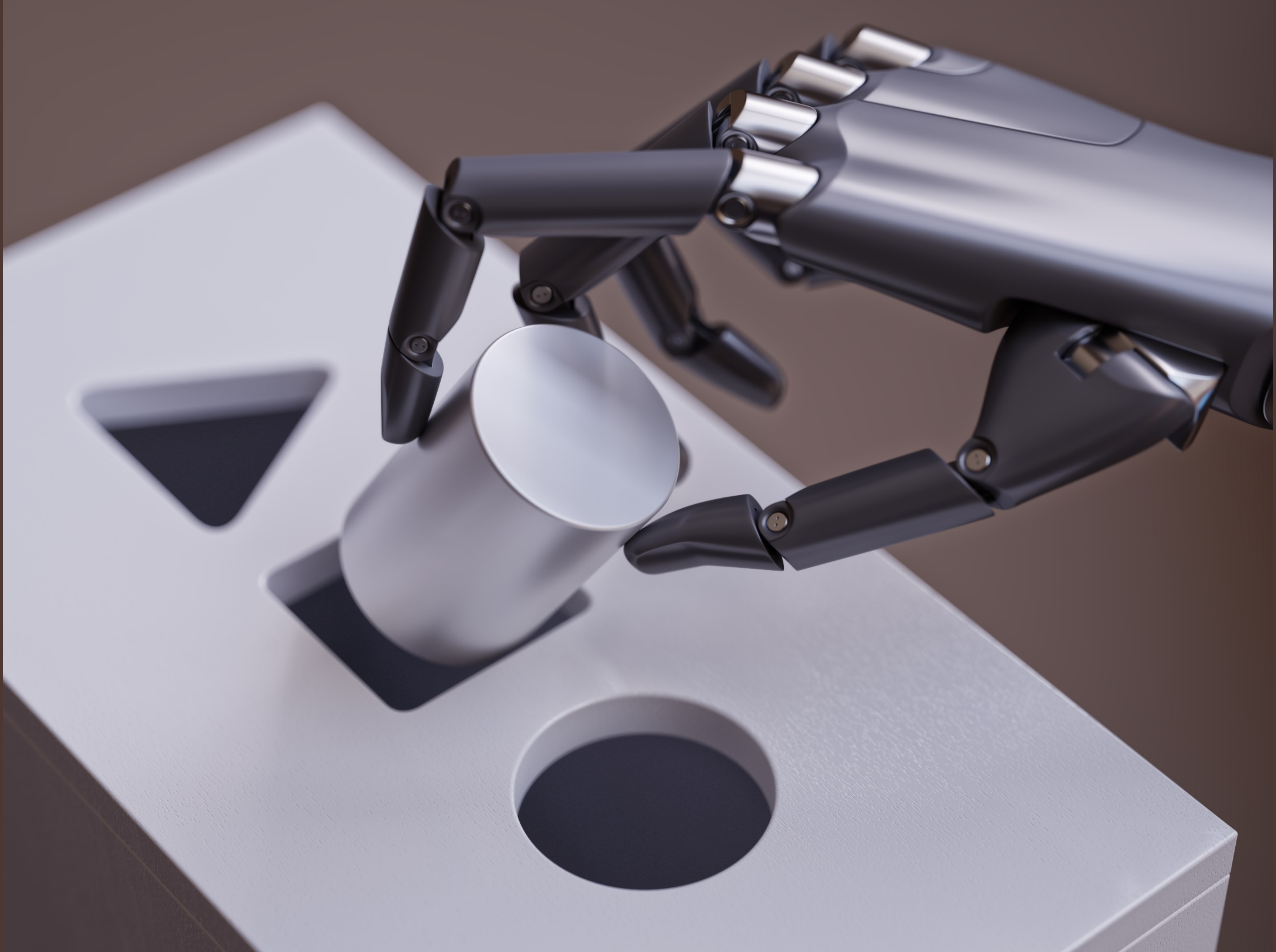
Strongly Typing the State

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE
@deborahkurata blogs.msmvps.com/deborahk/



Demo

The state of our state

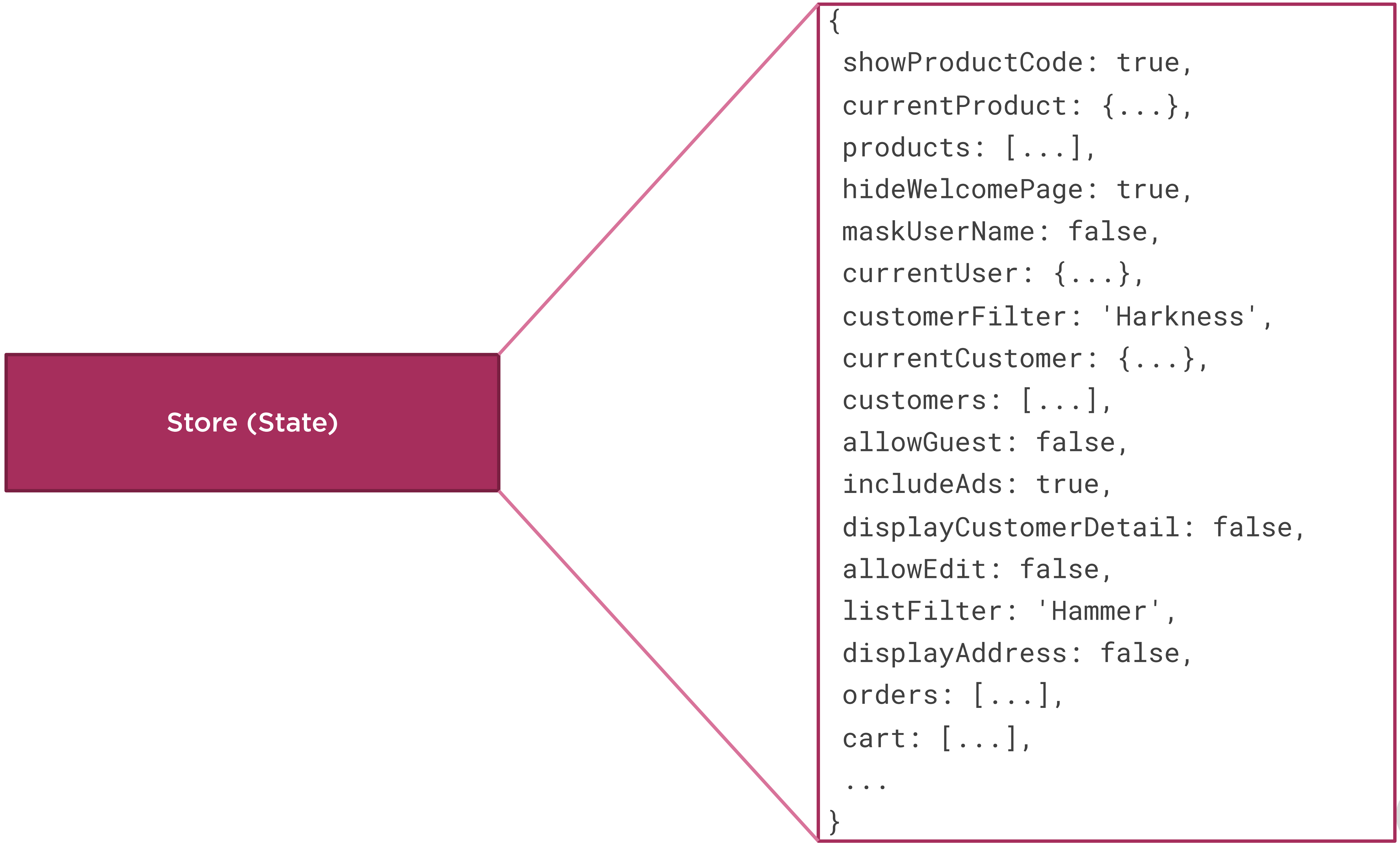
Module Overview

Define interfaces for slices of state

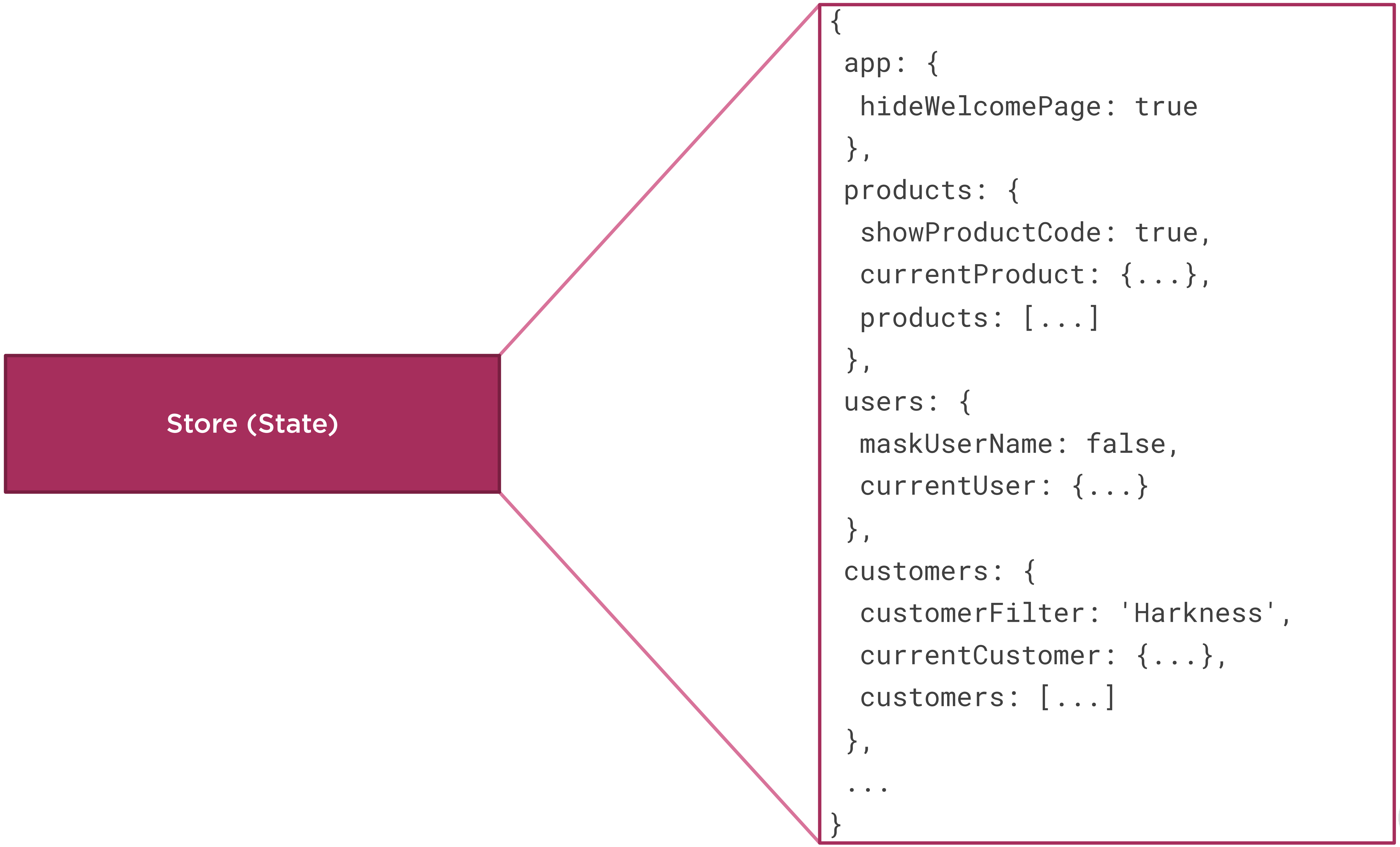
Use the interfaces for strong typing

Set initial state values

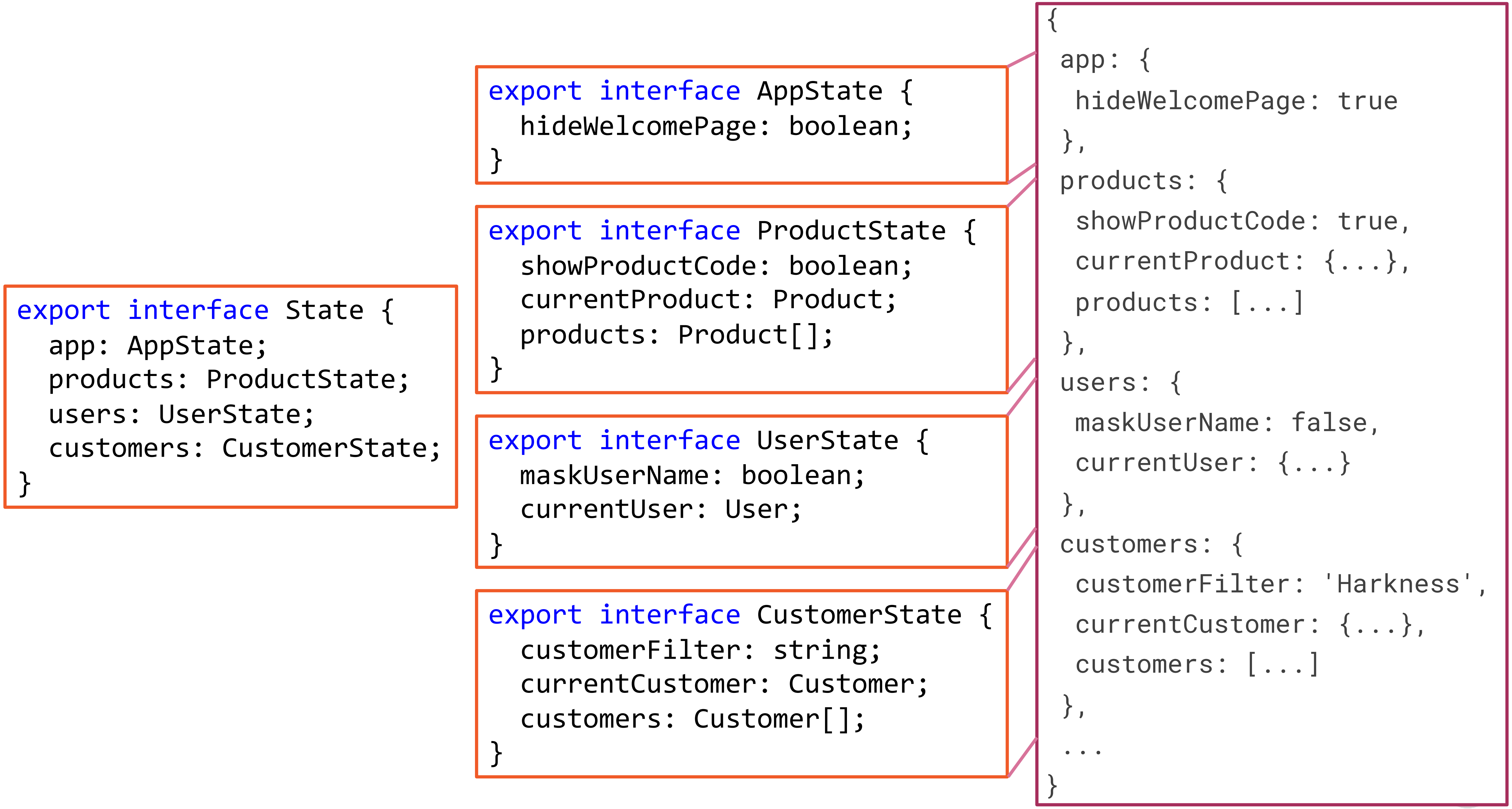
Build selectors



```
{
 showProductCode: true,
 currentProduct: {...},
 products: [...],
 hideWelcomePage: true,
 maskUserName: false,
 currentUser: {...},
 customerFilter: 'Harkness',
 currentCustomer: {...},
 customers: [...],
 allowGuest: false,
 includeAds: true,
 displayCustomerDetail: false,
 allowEdit: false,
 listFilter: 'Hammer',
 displayAddress: false,
 orders: [...],
 cart: [...],
 ...
}
```



```
{
 app: {
  hideWelcomePage: true
 },
 products: {
  showProductCode: true,
  currentProduct: {...},
  products: [...]
 },
 users: {
  maskUserName: false,
  currentUser: {...}
 },
 customers: {
  customerFilter: 'Harkness',
  currentCustomer: {...},
  customers: [...]
 },
 ...
}
```



```
export interface State {
  app: AppState;
  products: ProductState;
  users: UserState;
  customers: CustomerState;
}
```

```
export interface AppState {
  hideWelcomePage: boolean;
}
```

```
export interface ProductState {
  showProductCode: boolean;
  currentProduct: Product;
  products: Product[];
}
```

```
export interface UserState {
  maskUserName: boolean;
  currentUser: User;
}
```

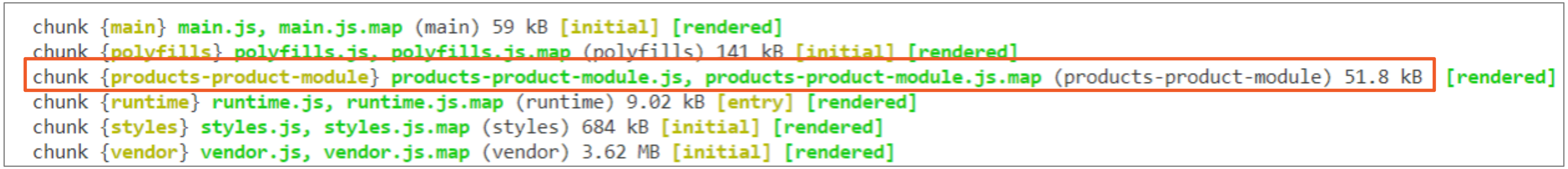
```
export interface CustomerState {
  customerFilter: string;
  currentCustomer: Customer;
  customers: Customer[];
}
```

```
{
 app: {
  hideWelcomePage: true
 },
 products: {
  showProductCode: true,
  currentProduct: {...},
  products: [...]
 },
 users: {
  maskUserName: false,
  currentUser: {...}
 },
 customers: {
  customerFilter: 'Harkness',
  currentCustomer: {...},
  customers: [...]
 },
 ...
}
```

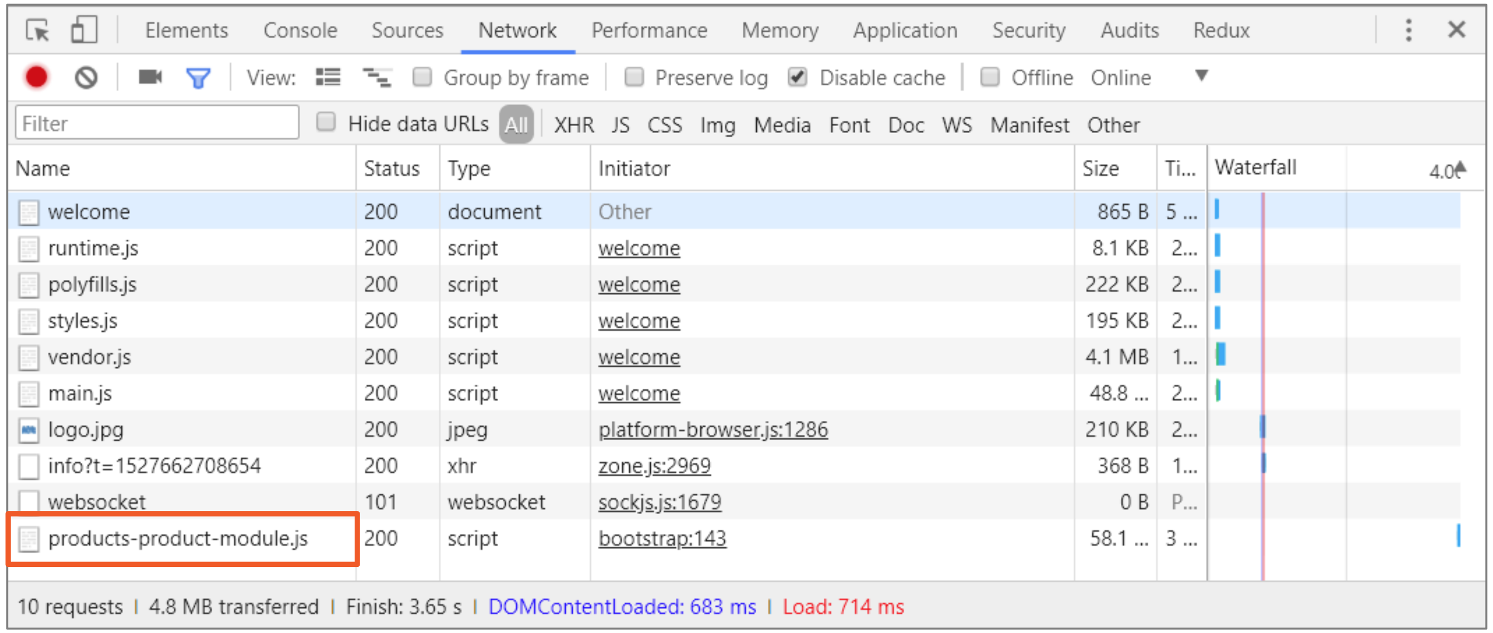
Demo

Defining an interface for a slice of state

Lazy Loading



```
chunk {main} main.js, main.js.map (main) 59 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 141 kB [initial] [rendered]
chunk {products-product-module} products-product-module.js, products-product-module.js.map (products-product-module) 51.8 kB [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 9.02 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 684 kB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 3.62 MB [initial] [rendered]
```



Group by frame ☐ · Preserve log ☐ · Disable cache ✓ · Offline ☐ · Online · Hide data URLs ☐ · All

Name	Status	Type	Initiator	Size	Ti...
welcome	200	document	Other	865 B	5 ...
runtime.js	200	script	welcome	8.1 KB	2...
polyfills.js	200	script	welcome	222 KB	2...
styles.js	200	script	welcome	195 KB	2...
vendor.js	200	script	welcome	4.1 MB	1...
main.js	200	script	welcome	48.8 ...	2...
logo.jpg	200	jpeg	platform-browser.js:1286	210 KB	2...
info?t=1527662708654	200	xhr	zone.js:2969	368 B	1...
websocket	101	websocket	sockjs.js:1679	0 B	P...
products-product-module.js	200	script	bootstrap:143	58.1 ...	3 ...

10 requests | 4.8 MB transferred | Finish: 3.65 s | DOMContentLoaded: 683 ms | Load: 714 ms

app

```
import { ProductState } from '../products/state/product.reducer';
import { UserState } from '../user/state/user.reducer';

export interface State {
 products: ProductState;
 users: UserState;
}
```

app

```
import { UserState } from '../user/state/user.reducer';

export interface State {
 users: UserState;
}
```

products

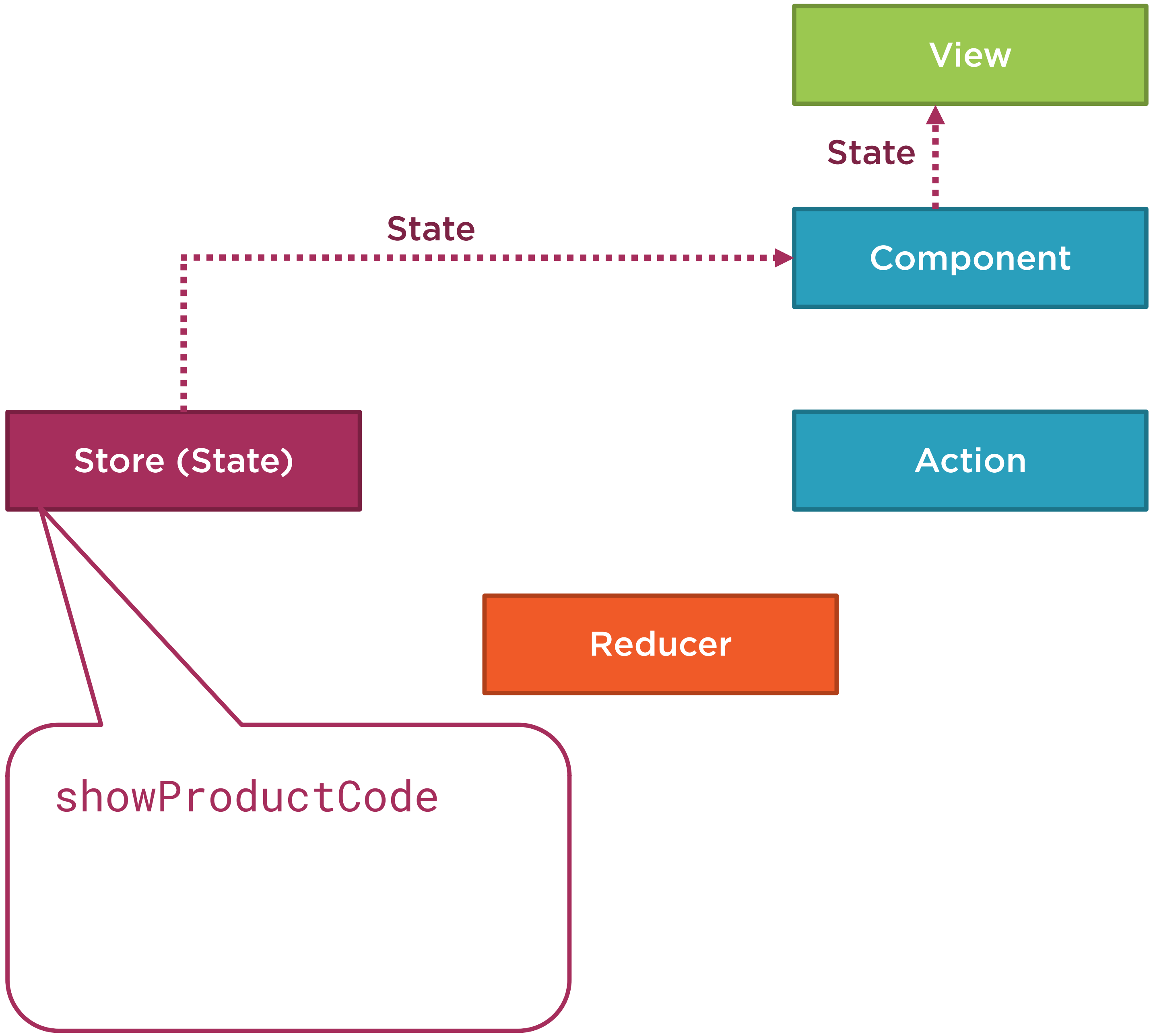
```
import * as AppState from '../../state/app.state';

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

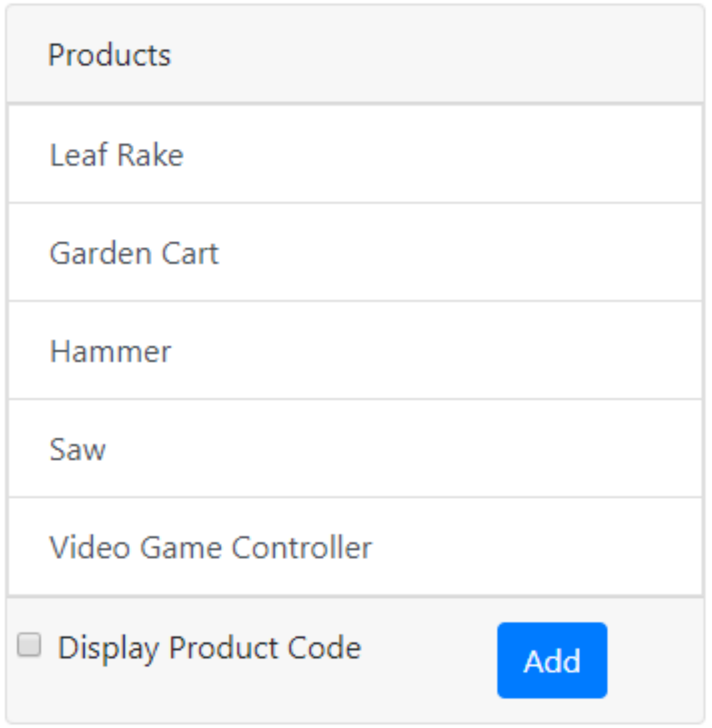
Demo

Extending the state interface for lazy loaded features

Demo

Strongly typing the state




Products

Leaf Rake

Garden Cart

Hammer

Saw

Video Game Controller

☐ Display Product Code

Set Initial Values

Product Reducer

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

Product Reducer

```
export const productReducer = createReducer<ProductState>(
  initialState,
  on(ProductActions.toggleProductCode, (state): ProductState =>
 {
    return {
      ...state,
      showProductCode: !state.showProductCode
    };
  })
);
```

Demo

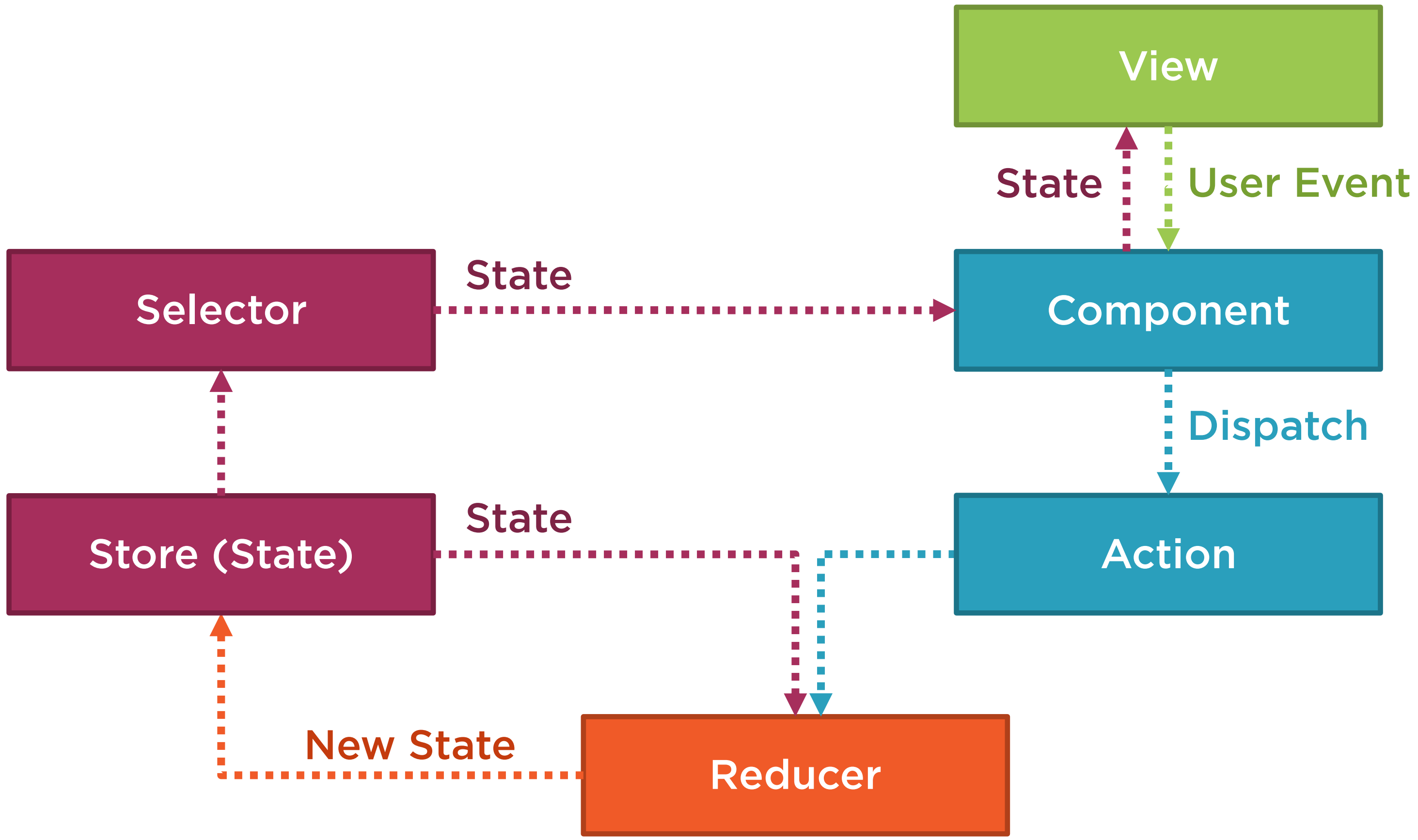
Setting initial state values

Store (State)

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

1. Hard-coded string
2. Knows the store structure
3. Watches for any changes

```
this.store.select('products').subscribe(
  products => this.displayCode = products.showProductCode
);
```


View
State
User Event
Selector
State
Component
Dispatch
Store (State)
State
Action
New State
Reducer

Benefits of Selectors

Provide a strongly typed API

Decouple the store from the components

Encapsulate complex data transformations

Reusable

Memoized (cached)

Selector

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

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

Selector

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

```
products: {
showProductCode: true
  currentProduct: {...},
  products: [...]
}
```

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

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

Using a Selector

Without a selector

```
this.store.select('products').subscribe(
   products => this.displayCode = products.showProductCode
);
```

With a selector

```
import { State, getShowProductCode } from '../state/product.reducer';

this.store.select(getShowProductCode).subscribe(
   showProductCode => this.displayCode = showProductCode
);
```

Demo

Building selectors

Demo

Using selectors

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)
);
```

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

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

Checklist: Strongly Typing State

Store (State)

Define an interface for each slice of state:

```
export interface ProductState {
  showProductCode: boolean;
  currentProduct: Product;
  products: Product[];
}
```

Compose them for the global application state

Use the interfaces to strongly type the state

```
import { State } from './state/product.reducer';
...
constructor(private store: Store<State>) { }
```

Checklist: Initializing State

Store (State)

Set initial values:

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

Initialize the state:

```
export const productReducer = createReducer<ProductState>(
  initialState,
  on(...)
);
```

Checklist: Building Selectors

Store (State)

Build selectors to define reusable state queries

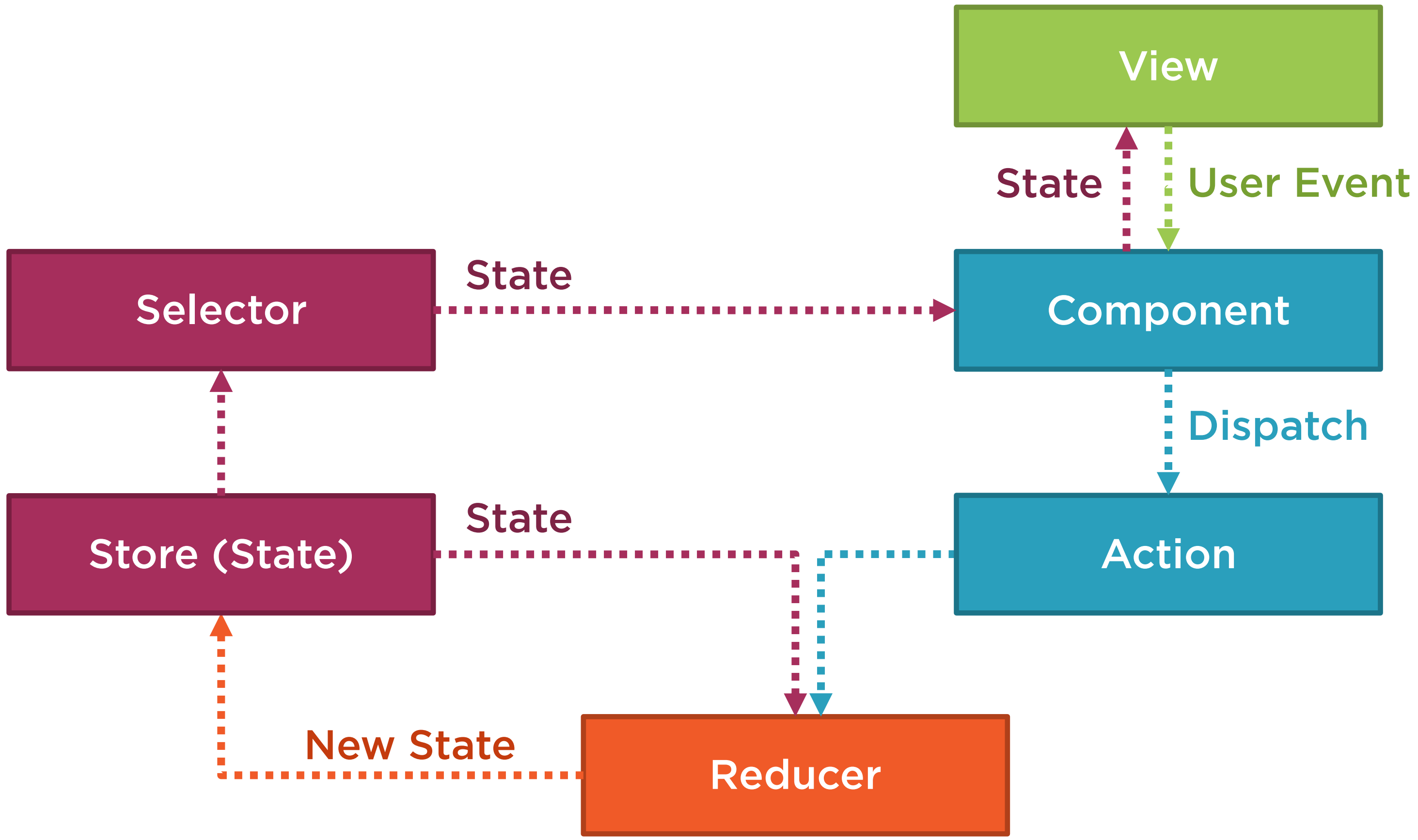
Conceptually similar to stored procedures

Feature selector:

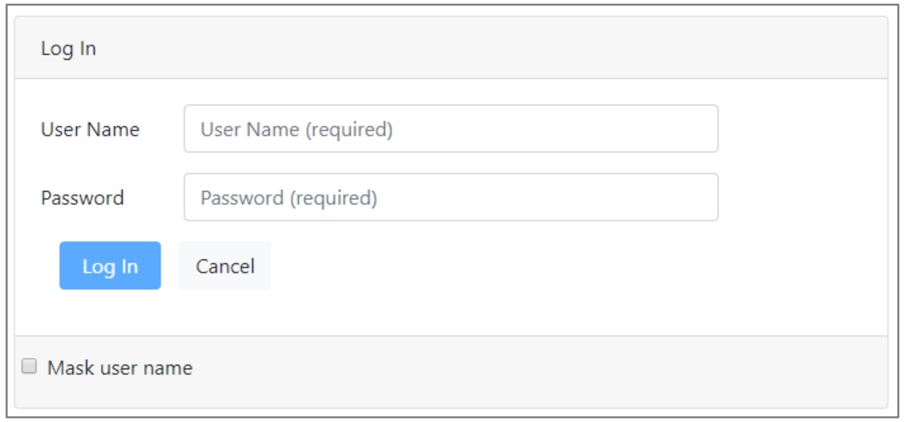
```
const getProductFeatureState =
  createFeatureSelector<ProductState>('products');
```

State selector:

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


View
State
User Event
Selector
State
Component
Dispatch
Store (State)
State
Action
New State
Reducer

Homework



Strongly type the user state

Build selectors for maskUserName and currentUser

Modify the reducer to use the strongly typed state

Modify the login component to use the strongly typed state and selector

https://github.com/DeborahK/Angular-NgRx-GettingStarted/tree/master/APM-Demo2