REDUX TOOLKIT in TS

Advanced Tips&Tricks

FABIO**BIONDI**.io

{codemotion}

Experts 2019
Web & Angular

MVP Microsoft® Most Valuable Professional

fabiobiondi.io

TS JS A

COUMMUNITY FOUNDER

A JS

26K users

twitch.tv/fabio_biondi

TYPED SAFE STORE

```
// Reducers
const reducers = combineReducers({
  theme: { value: number}
  items: itemsReducer,
  // other reducers
});


// Create the type of store based on rootReducer
export type RootState = ReturnType<typeof reducers>

export const store = configureStore({
  reducer: reducers,
  devTools: process.env.NODE_ENV !== 'production',
});
```

SELECTORS

```
import { RootState } from '../app';
export const getItems = (state: RootState) => state.items;
```

ACTIONS

```
import { createAction } from '@reduxjs/toolkit';
import { Item } from '../model/item';

export const loadItems = createAction('items/load')
export const addItem = createAction<Item>('items/add')
export const deleteItem = createAction<number>('items/delete')
```

REDUCERS

```
import { createReducer } from '@reduxjs/toolkit';
import { Item } from '../model/item';
import { addItem, deleteItem } from './items.actions';

export const initialState: Item[] = [];

// NOTE: action is any as type
export const itemsReducer = createReducer(initialState, {
  [addItem.type]: (state, action) => [...state, action.payload],
  [deleteItem.type]: (state, action) => state.filter(item => item.id !== action.payload)),
});
```

REDUCERS: TYPE SAFE ACTIONS

```
import { createReducer, PayloadAction } from '@reduxjs/toolkit';
import { Item } from '../model/item';
import { addItem, deleteItem } from './items.actions';

export const initialState: Item[] = [];

export const itemsReducer = createReducer(initialState, {
  [addItem.type]: (state: Item[], action: PayloadAction<Item>) => [...state, action.payl
  [deleteItem.type]: (state: Item[], action: PayloadAction<number>) => state.filter(item
})
```

TYPE SAFE REDUCER

```
import { createReducer } from '@reduxjs/toolkit';
import { Item } from '../model/item';
import { addItem, deleteItem } from './items.actions';

export const initialState: Item[] = [];

// NOTE: action is inferred correctly
export const itemsReducer = createReducer(initialState, builder =>
  builder
    .addCase(addItem, (state, action) => [...state, action.payload])
    .addCase(deleteItem, (state, action) => state.filter(item => item.id !== action.payload))
```

Slice & ImmerJS

```
import { createSlice, PayloadAction } from '@reduxjs/toolkit'
import { Item } from '../model/item';

export const itemsStore = createSlice({
  name: 'items',
  initialState: [] as Item[],
  reducers: {
    addItem(state, action: PayloadAction<Item>) {
      state.push(action.payload); // state mutation (thanks to ImmerJS)
    },
    deleteItem(state, action: PayloadAction<number>) {
      const index = state.findIndex(item => item.id === action.payload);
      state.splice(index, 1);     // state mutation (thanks to ImmerJS)
    },
  }
});

export const { addItem, deleteItem } = itemsStore.actions
```

Middleware: THUNK

```
export type AppThunk = ThunkAction<void, RootState, null, Action<string>>
```

```
export const deleteProduct = (id: number): AppThunk => async dispatch => {
  try {
    await Axios.delete(`${API}/items/${id}`);
    dispatch(deleteItemSuccess(id))
  } catch (err) {
    dispatch(deleteItemFail())
  }
};
```

Mlddleware: Redux Observable

```
npm install redux-observable
```

```
// define middleware
const epicMiddleware = createEpicMiddleware();

// root epics
const rootEpic = combineEpics(
  loadProductEpic,
  deleteProductEpic
);
epicMiddleware.run(rootEpic);

// Add Middleware
export const store = configureStore({
  reducer: rootReducer,
  devTools: process.env.NODE_ENV !== 'production',
  middleware: getDefaultMiddleware => getDefaultMiddleware().concat(epicMiddleware)
});
```

Redux Observable - Epic

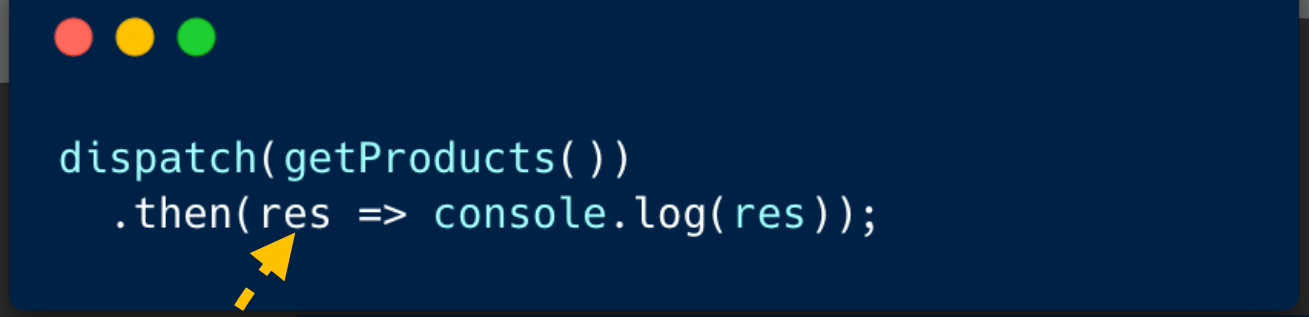
```
// TYPE SAFE
export const deleteProductEpic: Epic = (action$: Observable<PayloadAction<number>>): Observable<Action> => {
  return action$.pipe(
    ofType(deleteProduct.type),
    mergeMap(action => ajax.delete(`http://localhost:3001/products/${action.payload}`)),
    map(({response}) => ({ type: deleteProductSuccess.type })),
  );
}
```

Middleware: Redux Saga

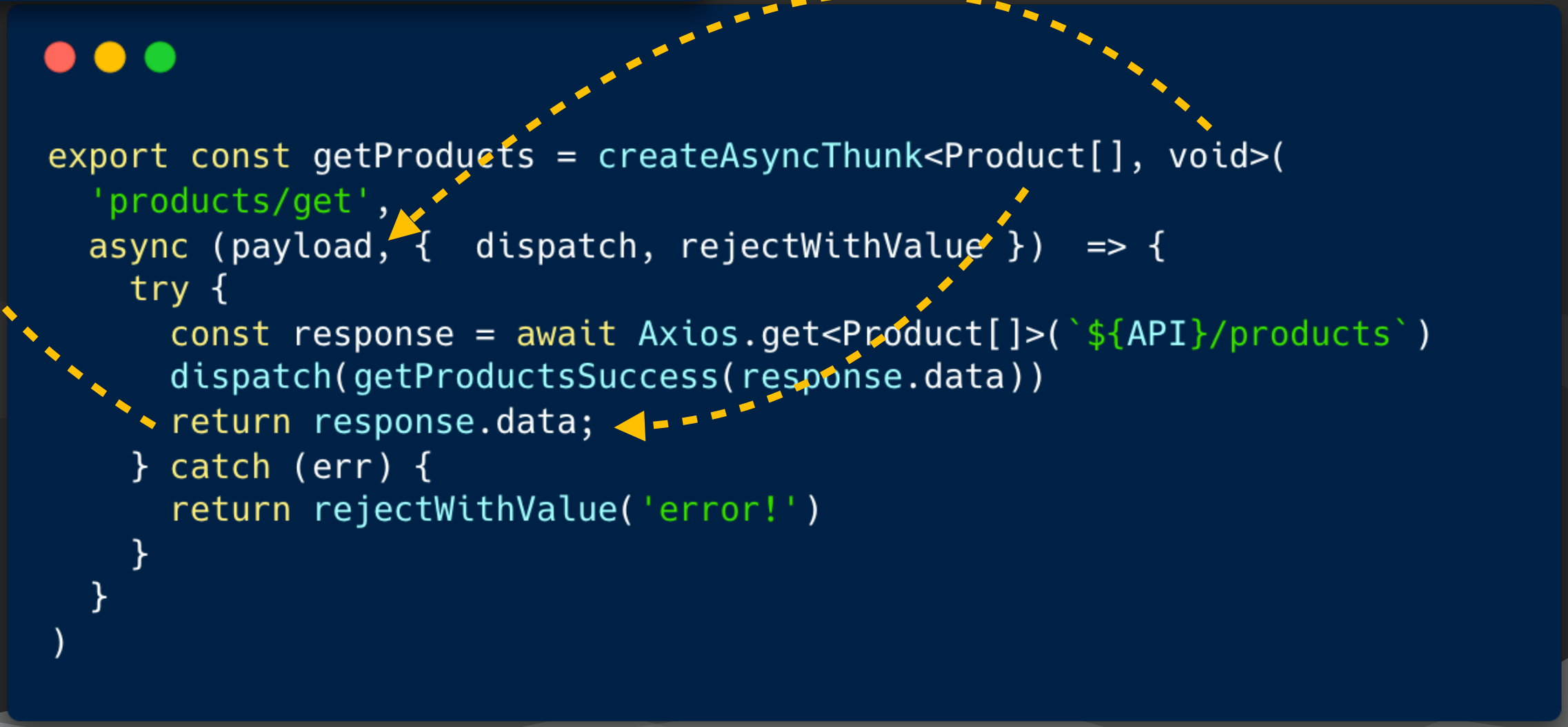
```
let callAPI = async ({ url, method, data }) => {
  return await Axios({ url, method, data });
};

export function* deleteItem({ id }) {
  try {
    const result = yield call(() =>
      callAPI({ url: `${API}/items/${id}`, method: 'DELETE' })
    );
    yield put(deleteItemSuccess(result.data));
  } catch (e) {
    yield put(deleteItemFail());
  }
}
```

createAsyncThunk

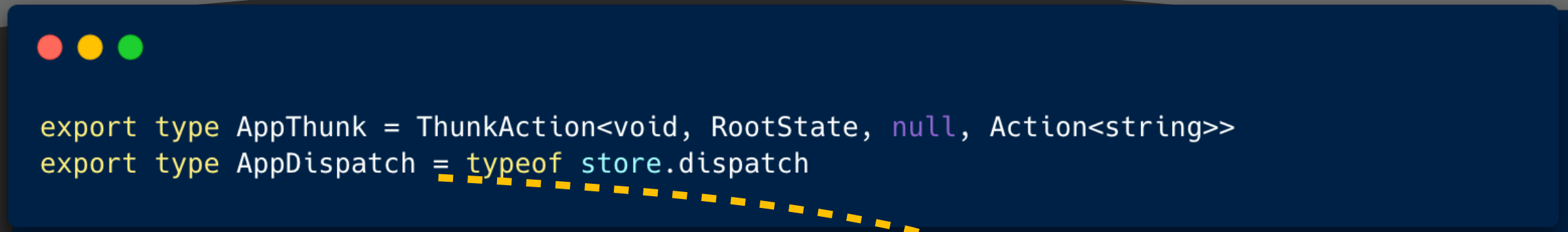
```
dispatch(getProducts())
  .then(res => console.log(res));
```



```
dispatch(getProducts())
  .then(res => console.log(res));
```



```
export const getProducts = createAsyncThunk<Product[], void>(
  'products/get',
  async (payload, {  dispatch, rejectWithValue })  => {
    try {
      const response = await Axios.get<Product[]>(`${API}/products`)
      dispatch(getProductsSuccess(response.data))
      return response.data;
    } catch (err) {
      return rejectWithValue('error!')
    }
  }
)
```



```ts
export type AppThunk = ThunkAction<void, RootState, null, Action<string>>
export type AppDispatch = typeof store.dispatch
```

```ts
export const CatalogPage: React.VFC = () => {

  const dispatch = useDispatch() as AppDispatch;

  useEffect(() => {
    dispatch(getProducts())
        .then(res => console.log(res));
  }, [dispatch]);

  const toggleHandler = (todo: Product) => {
    dispatch(toggleProduct(todo))
        .then(res => console.log(res));
  };

  // ...
}
```

EXTRA REDUCER (no typed)

```
export const httpStatusStore = createSlice({
  name: 'httpStatusStore',
  initialState: { error: false, pending: false } as ErrorState,
  reducers: {
    setError(state, action: PayloadAction<boolean>) {
      state.error = action.payload.value;  // state mutation by ImmerJS
    }
  },
  extraReducers: {
    ['products/get/fulfilled']  : (state, action) => {
      return { error: false, pending: false };
    },
    'products/get/rejected'   : (state, action) => {
      return { error: true, pending: false };
    },
    // ... other actions ...
    'items/add/pending'   : (state, action) => {
      return { error: false, pending: true };
    },
})

export const { setError } = httpStatusStore.actions;
```

TYPE SAFEFY with EXTRA REDUCERS

```
export const httpStatusStore = createSlice({
  name: 'httpStatusStore',
  initialState: { error: false, pending: false } as ErrorState,
  reducers: {
    // ...
  },
  extraReducers: builder => {
    builder.addCase(getProduct.fulfilled, (state,action) => {
      // both `state` and `action` are now correctly typed
      // ...
    }
    builder.addCase(getProduct.rejected, (state,action) => {
      // ...
    }
    builder.addCase(getProduct.pending, (state,action) => {
      // ...
    }
  }
})

export const { setError } = httpStatusStore.actions;
```

CUSTOM MIDDLEWARE

```
export const store = configureStore({
  reducer: rootReducer,
  middleware: getDefaultMiddleware => getDefaultMiddleware().concat(logger)
});
```

```
import { Middleware } from "@reduxjs/toolkit";
import { RootState } from '../../App';

export const logger: Middleware<{ }, RootState> = (store) => {
  return next => {
    return action => {
      try {
        console.log('action', action);
        console.log('current state', store.getState());
        const result = next(action);
        console.log('next state', store.getState())
        return result;
      } catch(e) {
        console.log(e)
      }

    }
```

ENTITIES

```
export const booksAdapter = createEntityAdapter<Book>({
  selectId: (book: Book) => book.id,
  sortComparer: (a, b) => a.title.localeCompare(b.title),
})
```

ENTITIES

```
const rootReducer = combineReducers({
  // …
  books: booksSlice.reducer,
})
```

```
export const booksSlice = createSlice({
  name: 'books',
  initialState: booksAdapter.getInitialState({
    max: 10
  }),
  reducers: {
    bookAdded: booksAdapter.addOne,
    bookDeleted: booksAdapter.removeOne,
    booksReceived(state, action: PayloadAction<Book[]>) {
      // add many items, if entities are < of MAX value
      if (state.ids.length < state.max) {
        booksAdapter.addMany(state, action.payload)
      }
    },
    bookUpdated: booksAdapter.updateOne
  }
})
```

ENTITIES USAGE

```
const booksSelectors = booksAdapter.getSelectors<RootState>(
  (state) => state.books
)

export const Books: React.FC = () => {
  const allBooks = useSelector(booksSelectors.selectAll)

  function add(book: Book) {
    store.dispatch(bookAdded({ ...book}));
  }

  function addMany(books: Book[]) {
    store.dispatch( booksReceived([...books]) )
  }

  function deleteBook(id: string) {
    store.dispatch(bookDeleted(id))
  }

  return <div>
    {/*...*/}
    {
      allBooks?.map(book => {
        return <li key={book.id}>
          {book.title}
          <button onClick={() => deleteBook(book.id)}>Delete</button>
        </li>
```

RTK
QUERY

RKT QUERY: SERVICE

```ts
import { createApi, fetchBaseQuery } from '@rtk-incubator/rtk-query';

// Define a service using a base URL and expected endpoints
export const pokemonApi = createApi({
  reducerPath: 'pokemonApi',
  baseQuery: fetchBaseQuery({ baseUrl: 'https://pokeapi.co/api/v2/' }),
  endpoints: (builder) => ({
    getPokemonByName: builder.query({
      query: (name: string) => `pokemon/${name}`,
    }),
  }),
});

export const { useGetPokemonByNameQuery } = pokemonApi;
```

type-safe hooks (TS 4.1+ only)

`use(Endpointname)(Query|Mutation)`

RKT QUERY: USAGE

```
// Parent Component
<button onClick={() => setPokeName('pikachu')}>pikachu</button>
<button onClick={() => setPokeName('bulbasaur')}>Bulbasaur</button>
<PostDetail pokeName={pokeName}/>


// Child Component
export const PostDetail = ({ pokeName }: { pokeName: string }) => {
  const { data, error, isLoading, isFetching } = useGetPokemonByNameQuery(pokeName);

  if (isLoading) return <div>Loading...</div>;
  if (error) return <div>Error!!!</div>;
  if (!data) return <div>Missing post!</div>;

  return (
    <div>
      {data.species.name} {isFetching ? '...refetching' : ''}
    </div>
  );
};
```

POLLING

```tsx
export const PollingDemo: React.FC = () => {
  const { data, error, isLoading } = useGetItemsByCatQuery(
    'fruits',
    { pollingInterval: 3000 }
  );

  return (
    <div className="App">
      {error ? (
        <>Oh no, there was an error</>
      ) : isLoading ? (
        <>Loading...</>
      ) : data ? (
        <ul>
          { data.items?.map(item => <li key={item.id}>{item.name}</li> }
        </ul>
      ) : null}
    </div>
  );
};
```

PREFETCH & PAGINATION

```
export const articlesApi = createApi({
  baseQuery: fetchBaseQuery({ baseUrl: "https://jsonplaceholder.typicode.com/" }),
  endpoints: (builder) => ({
    getArticles: builder.query<Post[], number | void>({
      query: (page = 1) => `posts?_page=${page}&_limit=3`
    })
  })
});

export const { useGetArticlesQuery, usePrefetch} = articlesApi;
```

```tsx
export const DemoPrefetch: React.FC = () => {
  const [page, setPage] = useState(1);
  const { data, isLoading, isFetching } = useGetArticlesQuery(page);

  const prefetchPage = usePrefetch('getArticles');

  const prefetchNext = useCallback(() => {
    prefetchPage(page + 1);
  }, [prefetchPage, page]);

  console.log(page)
  if (isLoading) {
    return <div>Loading</div>;
  }

  if (!data) {
    return <div>No posts</div>;
  }

  return <>
    {data?.map(item => <li key={item.id}>{item.title}</li>)}

    <button
      onClick={() => setPage(p => p + 1)}
      onMouseEnter={prefetchNext}
    >{isFetching ? 'Loading' : 'Next'}</button>
  </>
};
```

THAT's ALL

formazione.fabiobiondi.io

twitch.tv/fabio_biondi