Cheatsheets / **Redux**

codecademy

Refactoring with Redux Toolkit

Redux Toolkit

Redux Toolkit, also known as the `@reduxjs/redux-toolkit` package, contains packages and functions that are essential for building a Redux app. Redux Toolkit simplifies most Redux tasks like setting up the store, creating reducers and performing immutable updates.

Installing Redux Toolkit

The `@reduxjs/redux-toolkit` package is added to a project by first installing it with `npm`.
Some of the resources imported from `@reduxjs/redux-toolkit` are:

- `createSlice`
- `configureStore`

```
npm install @reduxjs/redux-toolkit
```

`createSlice()` Options Object

The `createSlice()` function is used to simplify and reduce the code needed when creating application slices. It takes an object of options as an argument. The options are:

- `name`: the slice name used as the prefix of the generated `action.type` strings
- `initialState`: the initial value for the state to be used by the reducer
- `reducers`: an object of action names and their corresponding case reducers

```
/*
The action.type strings created will be
'todos/clearTodos' and 'todos/addTodo'
*/
const options = {
  name: 'todos',
  initialState: [],
  reducers: {
    clearTodos: state => [],
    addTodo: (state, action)
      => [...state, action.payload]
  }
}
const todosSlice = createSlice(options);
```

“Mutable” Code with `createSlice()`

`createSlice()` lets you write immutable updates using “mutation-like” logic within the case reducers. This is because `createSlice()` uses the Immer library internally to turn mutating code into immutable updates. This helps to avoid accidentally mutating the state, which is the most commonly made mistake when using Redux.

```
/*
addTodo uses the mutating push() method
*/
const todosSlice = createSlice({
  name: 'todos',
  initialState: [],
  reducers: {
    clearTodos: state => [],
    addTodo: (state, action)
      => state.push(action.payload)
  }
});
```

Slices with `createSlice()`

`createSlice()` returns an object containing a slice reducer (`todosSlice.reducer`) and corresponding auto-generated action creators (`todosSlice.actions`).

> The slice reducer is generated from the case reducers provided by `options.reducers` .
>
> The action creators are automatically generated and named for each case reducer. The `action.type` values they return are a combination of the slice name (`'todos'`) and the action name (`'addTodo'`) separated by a forward slash (`todos/addTodo`).

When creating slices in separate files it is recommended to export the action creators as named exports and the reducer as a default export.

```
const todosSlice = createSlice({
  name: 'todos',
  initialState: [],
  reducers: {
    addTodo: (state, action)
      => state.push(action.payload)
  }
});

/*
todosSlice = {
  name: "todos",
  reducer: (state, action) => newState,
  actions: {
    addTodo: (payload) => ({type:
"todos/addTodo", payload})
  },
  caseReducers: {
    addTodo: (state, action) => newState
  }
}
*/

export const { addTodo } = todosSlice.actions;
export default todosSlice.reducer;
```

Create store with configureStore()

`configureStore()` accepts a single configuration object parameter. The input object should have a `reducer` property that is assigned a function to be used as the root reducer, or an object of slice reducers which will be combined to create a root reducer. When `reducer` is an object `configureStore()` will create a root reducer using Redux's `combineReducers()`.

codecademy

```
import todosReducer from
'.todos/todosSlice';
import filterReducer from
'.filter/filterSlice';

const store = configureStore({
  reducer: {
    todos: todosReducer,
    filter: filterReducer
  }
});
```