Lecture 10: Async Redux, Tools

Jordan Hayashi

Previous Lecture

- Scaling Complexity
- Flux
- Redux
- `simpleRedux/`
- Reducers
- Store
- Actions
- `react-redux`

Review: `react-redux`

- React bindings for redux
  - `<Provider />`
  - `connect()`
- `Provider` gives children access to our redux store
- `connect()` helps us subscribe to any subset of our store and bind our action creators

Async `simpleRedux/`

Supporting Async Requests

- Where do we want to add this support? How do we change our API?
    - Reducers
    - Store
    - Actions
    - Action creators
- We need to change more than just the action creators
- `Store.dispatch()` needs to accept other types
- Our addition is unideal, since we had to change our redux implementation

Redux Middleware

- This allows us to extend redux without having to touch the implementation
- Any function with this prototype can be middleware
  - `({getState, dispatch}) => next => action => void`
- We can reimplement our feature as middleware
- https://github.com/gaearon/redux-thunk
  - “A thunk is a function that wraps an expression to delay its evaluation”

Persisting State

- Our app can now be a pure function of the redux store
- If we can persist the store, we can reload the app into the current state
- React Native provides `AsyncStorage`
  - “Use an abstraction on top of `AsyncStorage` instead of using it directly for anything more than light usage since it operates globally.”

`redux-persist`

- Abstracts out the storage of the store into `AsyncStorage`
- Gives us `persistStore`, `persistReducer`, `PersistGate`
  - Automatically stores the state at every change
  - Automatically rehydrates the store when the app is re-opened
  - Will display loading screen while waiting for store to rehydrate
- https://github.com/rt2zz/redux-persist

Container vs Presentational Components

- As an application grows in size and complexity, not all components need to be aware of application state
- Container components are aware of redux state
- Presentational components are only aware of their props

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

Do I need Redux?

- Redux helps apps scale, but does add complexity
- Sometimes, the complexity overhead isn’t worth it
- Do as much as you can with local component state, then add redux if you hit pain points
    - Forgetting to pass a prop
    - Directly managing deeply nested state
    - Duplicated information in state
    - Not updating all dependent props
    - Components with large number of props
    - Uncertainty where a piece of data is managed

JavaScript Tools

- NPM
- Babel
- `@std/esm`
- Chrome devtools
- React/Redux devtools
- ESLint
- Prettier
- Flow/TypeScript

JavaScript Tools

- NPM
- Babel
- `@std/esm`
- Chrome devtools
- React/Redux devtools
- ESLint
- Prettier
- Flow/TypeScript

ESLint

- “A fully pluggable tool for identifying and reporting on patterns in JavaScript”
- Allows us to enforce code style rules and statically analyze our code to ensure it complies with the rules
  - Ensure style consistency across a codebase

https://github.com/eslint/eslint

ESLint: Setup

- Install
  - Per project: `` `npm install --save-dev eslint` ``
  - Globally: `` `npm install -g eslint` ``
- Create your own config
  - Per project: `` `./node_modules/.bin/eslint --init` ``
  - Globally: `` `eslint init` ``
- Or extend an existing config
  - https://github.com/airbnb/javascript
  - https://github.com/kensho/eslint-config-kensho

ESLint: Running

- Run on a file/directory
  - Per project: `` `./node_modules/.bin/eslint <path>` ``
  - Globally: `` `eslint <path>` ``
- Lint whole project by adding as an NPM script
- Most text editors have an integration

Prettier

- “Prettier is an opinionated code formatter”
- Prettier will rewrite your files to adhere to a specified code style
- It can integrate with ESLint
  - Specify an eslint config and pass `--fix` to `eslint` to have prettier auto-fix improper styling

https://github.com/prettier/prettier