# Project Setup

## Install Dependencies

We use [`yarn`](https://yarnpkg.com/en/) to manage dependencies as it provides more reliable dependency list that the traditional npm.

Once you have yarn installed, you can run `yarn install`.

## Build Voyager

With all dependencies installed, you can run `npm run build` to build the project.

You can also use `npm run watch` for continuous build.

## Running Current Voyager Prototype

Currently we use `npm run watch` and `python -m SimpleHTTPServer <port>` to serve the app.

In the future, we plan to migrate to use `webpack-dev-server` with hot module loading support. (See #250 -- help wanted!)

# Coding Convention

## General

- Don't use `export default` ([Why?](https://basarat.gitbooks.io/typescript/docs/tips/defaultIsBad.html)).

- We make `index.ts` inside `actions/`, `models/`, and `reducers/` export everything inside it so that we can just import from `actions` without worrying about file renaming later.

## Redux

### Actions

- Action names should start with the part of the model it is modifying, so it's easy to categorize. For example, all bookmark actions have the "Bookmark" prefix.

- Synchronous actions should implement `ReduxAction` interfaces in `src/actions/redux-action`, which follows pattern described in https://github.com/acdlite/redux-actions.

- Since we already have types, we do not make action creator functions.

- For asynchronous actions, since they involve multiple sub-actions, we still have action creator functions (e.g., `datasetLoad()` in `actions/dataset.ts`). These action creators follow `redux-thunk` patterns.

- In `src/action/index.ts`, we have `ACTION\_TYPE\_INDEX` that uses TS's mapped type to declare so we make sure we generate `ACTION\_TYPES` that contain all actions.

## Vega-Lite / CompassQL Directives

- In `src/models/shelf.ts`, we re-declare a number of Vega-Lite / CompassQL interfaces with "Shelf" prefix, allowing them to be `SHORT\_WILDCARD` or both `SHORT\_WILDCARD` and full `Wildcard`.

## React Components

- Since our props and states are immutable, we always use [`PureComponent`](https://facebook.github.io/react/docs/react-api.html#react.purecomponent) for all of our components and rely on its shallow prop and state comparison.

- Use lowercase-dash-separated-names for file names. (Using PascalCase as filename can be problematic for OSX in general as OSX is case-insensitive.)

- Following Redux's [Using with React](http://redux.js.org/docs/basics/UsageWithReact.html) guide, we distinguish between presentation and container components. However, instead of having event handlers (e.g., `onTodoClick` in the guide) for every single events, we pass in `handleAction` property to presentation components (which implements our `ActionHandler` interface). For more information, see `src/actions/redux-action.ts`.

- You can use [this plugin](https://marketplace.visualstudio.com/items?itemName=infeng.vscode-react-typescript) to generate snippets (but manually remove default exports).

- Use `render` as a prefix for class method that returns a React component and gets called by the component's `render` method (e.g., `renderAddFilterSpan` in `<Field/>`).

- For CSS, we use [react-css-modules](https://github.com/gajus/react-css-modules) and SASS, which makes it easy to modularize style in React using the `styleName` tag. (See [react-css-modules](https://github.com/gajus/react-css-modules)'s README for more details.)

- Since CSS-Modules use filename as a prefix, never name a scss file `index.scss`. Instead use the module name. For example, the scss file for `data-pane/index.tsx` is `data-pane/data-pane.scss`.

## Reducer

- We make our props and states immutable. (Using TypeScript's `Readonly` type wrapper in reducers is very helpful to enforce immutability.)

- We use `...` to make sure things are immutable and we don't cause side-effects.

- For array, we have utility methods in `src/reducers/util.ts`.

## Models

- `src/models` contains all interfaces and helper methods for our redux state.