Fullstack Junior-Phase Boilerplate

A collaborative "cheat sheet" of boilerplate code

WHAT IS THIS?

The boilerplate code required to set up the packages we've learned will someday be as ingrained in our minds as our birthdays and social security numbers. Until that day arrives, we wanted to create a concise, easy-to-read reference to get your projects up and running as quickly as possible.

This is NOT meant to be a "copy-paste and go" sort of reference, but rather a quick "reminder" sheet. You will still have to know what this code does and adapt it to your own needs.

HOW CAN I EDIT THIS?

You can hop into suggestion mode through the navigation in the top-right corner and start typing away! That'll let you leave comments and make temporary edits to this document, which one of the admins can turn into permanent edits once we log back in! If you're contributing via suggestions, you won't need to worry about formatting, we can pretty it up to match the rest of the document.

The code blocks are formatted using **Code Pretty**, which is pretty straightforward. To keep things consistent and to make formatting easy, please put your code blocks in single-cell-tables. (If you can't figure out Code Pretty, just put your suggestions in and we'll do our best to format it before approving it.)

Express

NPM Modules

```
npm install --save express body-parser {morgan OR volleyball}
```

Dependencies

```
const express = require('express');
const path = require('path'); // path formatting utility
const bodyParser = require('body-parser'); // parsing middleware
const morgan = require('morgan'); // logging middleware, can substitute with volleyball
```

Setup

```
// define express server
const app = express();

// use morgan logging middleware
app.use(morgan('dev'));

// use body-parser middleware
app.use(bodyParser.json()); // parse JSON requests
app.use(bodyParser.urlencoded({ extended: true })); // parse URL requests

// static routing for /public/ path
app.use(express.static(path.join(__dirname, '..', 'public')));

// send index.html
app.use('*', (req, res, next) =>
  res.sendFile(path.join(__dirname, '..', 'public/index.html'))
);

// start server and listen on port 3000 (usually done after a db.sync)
app.listen(3000, () => console.log(`server listening on port 3000`));

// error-handling, should come AFTER all other routes
```

```
app.use((err, req, res, next) =>
  res.status(err.status || 500).send(err.message || 'Internal server error.')
);
```

Express Router

```
// in 'index.js' or 'start' file
const apiRouter = require('./api'); // will depend on route and file structure
app.use('/api', apiRouter);

// in '/api/index.js'
const router = require('express').Router();
module.exports = router;

// write routes (i.e. router.get(), router.set() or sub-routes)
```

PostgreSQL

This one is pretty simple: you just need to run ‘createdb {server_name}’ on the command line. Most of the work we do with Postgres is via Sequelize.

If anyone else can think of something we need to remember here for setup, please suggest it!

Sequelize

NPM Packages

```
npm install --save sequelize pg {pg-native AND/OR pg-hstore}
```

Dependencies

```
const Sequelize = require('sequelize');
```

Setup

```
const db = new Sequelize('postgres://localhost:5432/DB-NAME-HERE', {
    logging: false,
    native: true // omit this line if using pg-hstore
  }
);
```

Socket.IO

NPM Packages

```
npm install socket.io --save
```

Create Server

```
var socketio = require('socket.io');

// **This part below app.listen so the express app has priority**
var io = socketio(server);
```

User Socket Server as Event-Emitter

```
io.on('connection', function (socket) {
/* This function receives the newly connected socket.
       This function will be called for EACH browser that connects to our server.
       i.e. If Ben and Matt both connect to the server, this will run once when Ben
       connects, and once when Matt connects */
    console.log('A new client has connected!');
    console.log(socket.id);
});
```

Creating Socket Event

```
// Never seen window.location before?
// This object describes the URL of the page we're on!
var socket = io(window.location.origin);

socket.on('connect', function () {
    console.log('I have made a persistent two-way connection to the server!');
});

// **Remember: socket refers to one individual socket
//             io refers to every socket
```

Quick reference for methods below:

```
socket.emit('message', "this is a test"); //sending to sender-client only

socket.broadcast.emit('message', "this is a test"); //sending to all clients except sender

socket.broadcast.to('game').emit('message', 'nice game'); //sending to all clients in 'game' room(channel)
except sender

socket.to('game').emit('message', 'enjoy the game'); //sending to sender client, only if they are in 'game'
room(channel)

socket.broadcast.to(socketid).emit('message', 'for your eyes only'); //sending to individual socketid

io.emit('message', "this is a test"); //sending to all clients, include sender

io.in('game').emit('message', 'cool game'); //sending to all clients in 'game' room(channel), include
sender

io.of('myNamespace').emit('message', 'gg'); //sending to all clients in namespace 'myNamespace', include
sender

socket.emit(); //send to all connected clients

socket.broadcast.emit(); //send to all connected clients except the one that sent the message

socket.on(); //event listener, can be called on client to execute on server

io.sockets.socket(); //for emitting to specific clients
io.sockets.emit(); //send to all connected clients (same as socket.emit)
io.sockets.on() ; //initial connection from a client.
```

React

Tom's Super Important Laws

1. **State must ALWAYS be initialized with the appropriate data type.**
2. **Dumb components should be as dumb as possible, they should only calculate the view and nothing more.**
3. **All asynchronous behavior (such as AJAX) and side effects should go into a thunk.**

NPM Packages

```
npm install --save react react-router-dom
```

Dependencies

```
import React from 'react';
import { HashRouter as Router, Route, Link } from 'react-router-dom';
```

Creating a Smart Component

```
export default class ViewPets extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      view: 'all',
      property: 'value'
    };
  }

  changeState(view) {
    this.setState({
      view: view,
      property: 'newValue'
    })
  }

  componentDidMount() {
    this.setState({ view: this.props.match.params.view })
  }

  render() {
    let animals = catsData.concat(dogsData);
    if(this.state.view === 'cats' {
      animals = catsData;
    }

    return (
      <div>
        <h1>Pets</h1>
        <AnimalList animals={animals} />
      </div>
    )
  }
}
```

Creating a Dumb Component

```
const AnimalList = ({ animals }) => {
  return (
```

```
    <div className="gallery">
      { animals.map(animal => {
        return <AnimalCard key={animal.id} animal={animal} />;
      })}
    </div>
  )
}
```

Component Lifecycle Methods

```
/* Mounting */
  constructor(props) {} // Called before a component is mounted.

  componentWillMount() {} // Called immediately before mounting occurs.

  render() {} // Renders using returned JSX

  componentDidMount() {} // Called immediately after mounting.

/* Updating */
  componentWillReceiveProps(nextProps) {} // Invoked before a mounted component gets new props.

  shouldComponentUpdate(nextProps, nextState) {} // Invoked before rendering when new props or state are
received.

  componentWillUpdate(nextProps, nextState) {} // Invoked immediately before rendering after new props are
received.

  componentDidUpdate(prevProps, prevState) {} // Invoked immediately after updating occurs, but not called
on initial render.

/* Unmounting */
  componentWillUnmount() {} // Invoked immediately before a component is unmounted or destroyed.

/* Error Handling */
  componentDidCatch(error, info) {} // Catches errors anywhere in child component tree.
```

Using React Router

```
// Inside a Component
render() {
    return (
      <Router>
        <div className="col-xs-10">
          <Route exact path="/" component={AllAlbums} />
          <Route path="/albums" component={AllAlbums} />
        </div>
      </Router>
    );
  }
```

Moar Code

```
```

Redux

NPM Packages

```
npm install --save ***
```

Dependencies

```
// In store.js
import { createStore } from 'redux';
```

Define the Initial State

```
const initialState = {
     counter: 0,
};
```

Define Action Types

```
const INCREMENT_COUNTER = 'INCREMENT_COUNTER';
```

Define Action Creator

```
export function incrementCounter(interval) {
  return {
    type: INCREMENT_COUNTER,
    interval
  }
}
```

Define Reducer

```
function reducer(prevState=initialState, action) {
  switch(action.type) {
    case INCREMENT_COUNTER:
      let newState = Object.assign({}, prevState);
      newState.counter += action.interval;
      return newState;
    default:
      return prevState;
  }
};
```

Define and Export Store

```
const store = createStore(reducer);
export default store;
```

React-Redux

NPM Packages

```
npm install --save react-redux
```

Dependencies (index.js)

```
import { Provider } from 'react-redux';
import React from 'react';
```

```
import ReactDOM from 'react-dom';
import { BrowserRouter as Router } from 'react-router-dom'; // choose router type
import { Main } from './components'; // will depend on where your Main.js is defined
import store from './store'; // will depend on where your store is defined
```

Setup (index.js)

```
ReactDOM.render(
 <Provider store={store}>
   <Router>
     <Main /> // render your main component
   </Router>
 </Provider>,
 document.getElementById('app') // second argument to render(), references root node in your HTML
);
```

Dependencies (connected component)

```
import { connect } from 'react-redux';
```

Setup (connected component)

```

```

Webpack

Webpack.config.js

```
'use strict';

// The exports is a configuration object that tells webpack what to do
module.exports = {

  // The entry field tells webpack where our application starts.
  // Webpack will start building this file and any subsequent file(s) that are imported by that file
  entry: './browser/react/index.js',

  // The output field specifies where webpack's output will go. In this case, we've specified
  // that it should put it into a file called bundle.js in our public directory
  output: {
    path: __dirname,
    filename: './public/bundle.js'
  },

  // The context field simply sets the context for relative pathnames
  context: __dirname,

  // This handy option tells webpack to create another, special file called "bundle.js.map".
  // This special file is called a "source-map".
  // If enabled, your browser will automatically request this file so that it can faithfully re-create your source code in your browser's dev tools.
  // This way, when you open the code for debugging in Chrome dev tools, instead of seeing the hard-to-read transpiled code that webpack creates, you'll
  // see your clean source code.
  // For more info: https://developers.google.com/web/tools/chrome-devtools/javascript/source-maps
```

```
  devtool: 'source-map',

  // Here is where we specify what kinds of special syntax webpack should look out for
  module: {
    // Loaders are special node modules that we've installed that know how to parse certain syntax.
    // There are loaders for all different kinds of syntax.
    loaders: [
      {
        // Here, we want to test and see if any files end with .js or .jsx.
        // Only files that match this criteria will be parsed by this loader.
        test: /jsx?$/,
        // We want webpack to ignore anything in a node_modules or bower_components directory.
        // This is very important - modules have a responsibility to build their own js files.
        // If we were to do this ourselves, building our bundle.js would take forever!
        exclude: /(node_modules|bower_components)/,
        // We're using the babel-loader module to read our files - it can handle both ES6 and JSX!
        // Babel will use our .babelrc to figure out how to compile our code
        loader: 'babel-loader',
        // Here, we telling webpack to look for any syntax that looks like ES6 and any syntax that looks
like JSX.
        // If it finds it, the babel-loader will transpile it for us!
        query: {
          presets: ['react', 'es2015']
        }
      }
    ]
  }
};
```

Thunk

NPM Packages

```
npm install --save redux-thunk
```

Additional Resources

Placeholder

ES6 JavaScript - What You Need To Know

Destructuring assignment

- `let {a, b} = o` assigns Object `o`'s `a` and `b` properties to variables `a`, `b`
- `let [a, b] = arr` assigns first/second items of Array `arr` to variables `a` and `b`
- Assign defaults with `=`, e.g. `let {max = 5} = options`
- Destructuring can be performed on function arguments.
  `function fn({options = {}, flag = true}) { ... }`

`for .. of` loops

- Works on Iterables, including `Array`, `Map`, `Set` and generators.
- Does not work with objects.
- Use with destructuring assignment and `let`
  `for (let [key, value] of map) { ... }`

`let` / `const`

- Make variables scoped by block, not function
- Use in place of `var`
- `const` prevents re-assignment, but does not make assigned objects immutable

=> arrow functions

- *`argument => returned expression`*
- `this` inside function is equal to `this` where it was defined
  `function() { ... }.bind(this)`
- *returned expression* can be a block
  `x => { console.log('doubling'); return x*2 }`
- Use parentheses for more than one argument
  `(min, x, max) => Math.max(min, Math.min(x, max))`
- Use parentheses when argument is being destructured
  `({x, y}) => Math.sqrt(x*x, y*y)`

Backtick (``) Template Strings

- Interpolate with `${expression}`
  `` `Token token=${identity.get('accessToken')}` ``
- Can be split over multiple lines

... (spread operators / rest parameters)

- In functions parameters, creates an array of remaining arguments
  `function classes(...args) { return args.join(' ') }`
- In function arguments, expands array to actual parameters
  `console.log(...args)`
- Similar to `Function.prototype.apply`, but doesn't modify `this`

New `Array` Methods

- `arr.find(callback[, thisArg])`
  return the first item which when passed to `callback`, produces a truthy value
- `arr.findIndex(callback[, thisArg])`
  return the index of the first item which when passed to `callback` produces a truthy value
- `arr.fill(value[, start = 0[, end = this.length]])`
  fills all the elements of an array from a `start` index to an `end` index
- `arr.copyWithin(target, start[, end = this.length])`
  copies the sequence of items within the array to the position starting with `target`, taken from the position starting with `start`

New Built-in Classes

- `Map` - Map keys to values. Unlike objects, keys don't have to be strings
- `Set` - Store a set, where each stored value is unique
- `Symbol` - Use to make private object/class properties
- `Promise` - Manage callbacks for an event which will occur in the future

JavaScript Promises - What You Need To Know

The four functions you need to know

1. **`new Promise(fn)`**
   - `fn` takes two arguments: `resolve` and `reject`
   - `resolve` and `reject` are both functions which can be called with one argument
   - Returned promise will be rejected if an exception is thrown in the passed in function
2. **`promise.then(onResolve, onReject)`**
   - Returns a promise
   - Returned promise resolves to value returned from handler
   - Chain by returning a promise from `onResolve` or `onReject`
   - Returned promise will be rejected if an exception is thrown in a handler
   - Use `Promise.reject` to return a rejected promise from `onReject`
   - Make sure to follow by `promise.catch`
3. **`promise.catch(onReject)`**
   - Returns a promise
   - Equivalent to `promise.then(null, onReject)`
4. **`Promise.all([promise1, promise2, ...])`**
   - Returns a promise
   - When all arguments resolve, returned promise resolves to an array of all results
   - When any arguments are rejected, returned promise is immediately rejected with the same value
   - Useful for managing doing multiple things concurrently

Packages

- es6-promise - Polyfill older browsers
- bluebird - Get extra promise methods
- promisify-node - Promisify callback-accepting functions (npm)

Extra Reading

- Are JavaScript Promises swallowing your errors?
- Promises at MDN
- Promise browser support at Can I Use

The two functions you should know

- **`Promise.resolve(value)`**
  - Returns a promise which resolves to `value`
  - If `value` is a promise, just returns `value`
- **`Promise.reject(value)`**
  - Returns a rejected promise with the value `value`
  - Useful while processing errors with `promise.catch`

Patterns

- **Promisify a callback-accepting function `fn`**

  Assume callback passed to `fn` takes two arguments: `callback(error, data)`, where `error` is `null` if successful, and `data` is null if unsuccessful.

```
new Promise(function(resolve, reject) {
  fn(function(error, data) {
    if (error) {
      reject(error);
    }
    else {
      resolve(data);
    }
  });
});
```

- **Catch exceptions thrown in `then` handlers**

```
promise
  .then(function() { ... })
  .catch(function(err) {
    console.log(err.stack);
  });
```

React

THE ESSENTIALS

1. **`React.createElement(type, props, children)`**

   Create a `ReactElement` with the given component class, `props` and `children`.

   ```
   var link = React.createElement('a', {href: '#'}, "Save")
   var nav = React.createElement(MyNav, {flat: true}, link)
   ```

2. **`React.cloneElement(element, props, children)`**

   Create a new `ReactElement`, merging in new `props` and `children`.

3. **`ReactDOM.render(element, domNode)`**

   Take a `ReactElement`, and render it to a DOM node. E.g.

   ```
   ReactDOM.render(
     React.createElement('div'),
     document.getElementById('container')
   )
   ```

4. **`ReactDOM.findDOMNode(element)`**

   Return the DOM node corresponding to the given element (after `render`).

SPECIAL PROPS

`children` is automatically added to `this.props` by `React.createElement`.

`className` corresponds to the HTML `class` attribute.

`htmlFor` corresponds to the HTML `for` attribute.

`key` uniquely identifies a `ReactElement`. Used with elements in arrays.

`ref` accepts a callback function which will be called:

1. with the component instance or DOM node on mount.
2. with `null` on unmount and when the passed in function changes.

`style` accepts an *object* of styles, instead of a string.

PROPTYPES

Available under `React.PropTypes`. Optionally append `.isRequired`.

`any`	`array`	`bool`	`element`	`func`
`node`	`number`	`object`	`string`	

`instanceOf(constructor)`

`oneOf(['News', 'Photos'])`

`oneOfType([propType, propType])`

CLASS COMPONENTS

```
var MyComponent = React.createClass({
  displayName: 'MyComponent',

  /* ... options and lifecycle methods ... */

  render: function() {
    return React.createElement( /* ... */ )
  },
})
```

Options

`propTypes`	`object` mapping prop names to types
`getDefaultProps`	`function()` returning `object`
`getInitialState`	`function()` returning `object`

Lifecycle Methods

`componentWillMount`	`function()`
`componentDidMount`	`function()`
`componentWillReceiveProps`	`function(nextProps)`
`shouldComponentUpdate`	`function(nextProps, nextState)` -> `bool`
`componentWillUpdate`	`function(nextProps, nextState)`
`componentDidUpdate`	`function(prevProps, prevState)`
`componentWillUnmount`	`function()`

COMPONENT INSTANCES

- Accessible as `this` within class components
- Stateless functional components do not have component instances.
- Serve as the object passed to `ref` callbacks
- One component instance may persist over multiple equivalent `ReactElement`s.

Properties

`props` contains any props passed to `React.createElement`

`state` contains state set by `setState` and `getInitialState`

Methods

1. `setState(changes)` applies the given changes to `this.state` and re-renders
2. `forceUpdate()` immediately re-renders the component to the DOM