# Full Stack React Notes

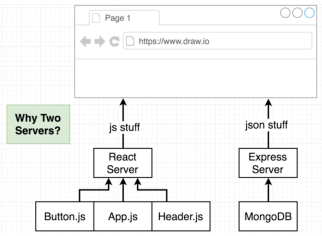
## Useful Resources / Libraries

Passport / Passport Strategy - useful with OAuth
- a strategy is a module that helps authentication with one specific provider

nodemon - refresh node server when any changes are made
mongoose.js - easier interaction with mongoDB (see: mongoosejs.com API docs)
mlab.com - mongodb deployment
cookie-session - middleware that helps manage cookies in express application
create-react-app - facebookincubator/create-react-app (see: github + documentation in readme)
concurrently - install package to run client and server side together with one command
http-proxy-middleware - package for proxy setup in react 2.0+
- create setupProxy.js and add proxy scripts export to file
- remove proxy scripts code from package.json
- will automatically be imported / read by rest of app

redux, react-redux, react-router-dom - goto react packages
materialize css - materializecss.com -
material ui - makes use of javascript based styling, so it might be challenging to change styling properties
axios - api package
redux-thunk
Stripe / Recurly - billing library api
npmjs.com - search npm package/api docs
body-parser - body parsing middleware before your handlers available under req.body property
sendgrid - identify users uniquely when sending emails
postman - REST client
redux-form - good documentation
sublime SideBarEnhancements package
email regex - email validation regular expressions
path-parser and url - routing help (see surveyRoutes.js)

## React




Running the Client and Server together: use concurrently
in scripts:

- “server": "nodemon index.js",
- “client”: ”npm run start --prefix client"
- “dev”: "concurrently \"npm run server\" \"npm run client\""

**Connecting react to back-end server:**

- use http-proxy-middleware for proxy help in dev
- in production the create-react-app server doesn’t “exist” so no need for proxy changes
- react project will build then will only reference relative route

**Reasons for architecture with API server:**

- browser assumes if uri is the same, cookies are included, if redirect uri is different cookies not included by default
- CORS (Cross Origin Resource Sharing) requests: if uri is the same, no CORS issues, if different, CORS issues

index.js - data layer control (redux)
App.js - rendering layer control (react router)

naming best practices
if given file exports a class, Capital leading character
if given file exports a function, lowercase leading character

**react-router:**
BrowserRouter can only have one child component (use div tag)
client side navigation library only, only once a user is inside an html document with js loaded do any of the react-router rules take effect

**Billing**

- Security
  - never accept raw credit card numbers
  - never store credit card info
  - always use an outside payment processor
- Billing is hard

- possible to avoid monthly payments/multiple plans
- fraud and chargebacks are a pain

make sure config from server side is completely independent of config from client side

<StripeCheckout
    amount={cents}
    token={token that you get back from stripe}
    stripeKey={publishable key from strip}
/>
token poorly named (should've probably been something like onToken)

**Environment Variables**
any custom env variable MUST begin with `REACT_APP_`
to define permanent env variables, create a file called `.env` in the root of your project:
    `REACT_APP_SECRET_CODE=abcdef`
.env.development / .env.production - environment specific keys
process.env.NODE_ENV - to get enviroment you're in
process.env.REACT_APP_SECRET_CODE - to get specific environemnt var defined

**Redux**
Redux Store
V
combineReducers
V
reducer1, reducer2, ...

**React Component** calls an **Action Creator** which returns an **Action** which is sent to **reducers** which update state in **redux store** which sends the state back to the components causing them to rerender.

Provider tag is a component that makes the store accessible to every component in the App

**Redux / Redux Form**
- Redux: allow us to put all of our data at the top without having to pass through our props all the way through the parent-child chain
- ReduxForm: don't have to wire up all the pieces from top to bottom. Not necessary for a simple one pages form.
    - formReducer is managed entirely by ReduxForm - records all values from

our form automatically
- only allows for one object to be passed in with property `form`
- `Field` component for each field in the form

Validation
- can pass a `validate` function to the object passed into reduxForm
- must return an errors object, if it's empty, validation is good to go
- order of errors in `validate` determines which error will show, the last thrown error will be the one displayed to user / in meta.errors

to reference a property of an object at runtime in JS, use [property] rather than object.property

Form Review - how to determine which component to show?
- Separate route? - would work well, but would have to put together extra code to make sure user is actually supposed to be there
- Redux? - update state in store with reducer, but would have to create a new action, action creator, reducer, etc.
- Component State? - inside higher level component, create boolean to determine which component to display, no need for separate route or updating global redux store
  - rule of thumb to determine whether to store something in redux store or component state: will any other feature / component expect to make use of this piece of state

in Event Handlers, only want to invoke a function call onEvent, so don't use parenthesis in JSX

destroyOnUnmount: false - doesn't remove values of form on "next"
- form: 'nameOfForm', can override between components

**Redux-Thunk**
gives us direct access to the Action Creator / dispatch function

Logging out from app:
- Full HTTP Request - will cause the browser page to refresh
- AJAX Request - no page refresh but we have to handle action creators, reducer, etc. and redirect to landing page

**Node / Express**
- Node - javascript runtime used to execute code outside of the browser
- Express - library that uses the Node runtime. has helpers to make dealing

with HTTP traffic easier

on server side we're using common js module so should use:
- const express = require('express')

on react side we're using ES2015 modules so can use:
- import express from 'express'

with express requests:
app.request( route, <middleware(s)>, function that processes requests and sends back response to user)
- can be any order as long as one sends response

**Express Route Handlers**

app - express App to register this route handler with
<request method> - watch for incoming request with this method
'<route>' - watch for incoming requests trying to access <route>
req / res - object representing the incoming request / outgoing response
res.send({hi:'there'}) - body of the arrow function - immediately send back some JSON back to whoever made this request

app.listen(<port>) - express tells node to listen to this port

**Project Structure**

config directory - API keys and settings
routes directory - route handlers, grouped by purpose
services directory - helper modules and business logic
index.js - helper modules and business logic

**HTTP is stateless** - By default information between requests is not shared
**Cookie based authentication** - use token to use in follow-up request to server

**Cookie-Session**

req.session
Cookie is a session (can directly access user_id) - easier to get started with
vs Express Session works by Cookie references a session (reference session_id)

Express middleware are wired up in the app.use() call

Whenever using passport, you can reference the user with req.user

to respond to a request use res.send(<data to communicate back to browser>)

## MongoDB / Mongoose

**Mongo** is "schema-less" unlike most other traditional dbs
model class —> an entire mongodb collection (one class —> one collection)
model instances —> each represents a different record in the collection
many collections, each collection has several records

with **Mongoose**, Schema needs to be defined
mongoose.model(<oneargument>) - fetch from mongoose
mongoose.model(<oneargument>, <twoargument>) - load to mongoose
.save() - store model instance in mongoDB
order of your require statements can result in errors
- need to require model schema before passport

model.id == model._id.$oid

each **record** inside of a **collection** is a **document**.
store a **subdocument collection** within a document
mongo size limit for a single record = 4mb
if a smaller amount of info needs to be stored on a type, can setup a

In Mongoose:
_relationshipField - '_' indicated relationship field

- use findOne, $elemMatch to find exact record you are looking for
- use updateOne - find record with given search criteria and use second object provided to update that record inside the mongo world, no need to pull it into express

Mongoose Tips:
- google search for what you're trying to do in query, lots of stack overflow questions on mongoose
- take everything from mongoose.connect() and above it in your root index.js, open node command line interface, past imports and required files into cmd line, can query mongoose in interface
  - Survey.find ({}).then(console.log)

query "projection" - whitelisting/backlisting fields

## Webhooks

when one server communicates with another server because of some event that happened on the first one

webhooks in production:

- sendgrid makes POST request to our server every 30 seconds or so
- POST to ourdomain.com/route/webhooks
- process list on API

webhooks in development:

- sendgrid makes POST request to our server every 30 seconds or so
- doesn't know where to POST to since we're local
- solve with: webhookhelper.localtunnel.com —> localtunnel server on our laptop —> POST

sendgrid.com - update Event Notifications under Settings —> Mail Settings
use preprocessing for requestHandler to avoid dirty data from WebHook

sendGrid info
"""
https://nkjchampemailysubdomain.localtunnel.me/api/surveys/webhooks

"""

**Deployment**

**Server Side Checklist:**

- **Dynamic Port Binding**: which port our app will use (listen to that port either process.env.PORT or use specific self-defined PORT locally)
- **Specify Node Environment**: tell heroku which version of node we are using
- **Specify Start Script**: what command to run to start our server
- **Create .gitignore file**: don't want to include dependencies

herok app link | heroku git repository
https://safe-earth-95313.herokuapp.com/ | https://git.heroku.com/safe-earth-95313.git
node_env - to determine the environment we're in

**Client Side**
3 Options:

1. build client project —> commit built project —> push to heroku (this method breaks convention / not encouraged)
2. push to heroku —> tell heroku to install *all* dependencies for client project —> heroku builds client project
3. push to CI server —> run tests —> CI builds and commits client —> CI pushes build to heroku