Jobify

Track Your Job Search

Project in Action - Jobify

Support

Find the App Useful? You can always buy me a coffee

Run The App Locally

```
npm run install-dependencies
```

- rename .env.temp to .env
- setup values for - MONGO_URL, JWT_SECRET, JWT_LIFETIME

```
npm start
```

- visit url http://localhost:3000/

Setup React App

- create **client** folder
- open terminal

```
cd client
```

```
npx create-react-app .
```

```
npm start
```

- set editor/browser side by side
- copy/paste assets from complete project

Spring Cleaning

- in src remove

- App.css
- App.test.js
- logo.svg
- reportWebVitals.js
- setupTests.js
- fix App.js and index.js

Title and Favicon

- change title in public/index.html
- replace favicon.ico in public
- resource Generate Favicons

Normalize.css and Global Styles

- CSS in JS (styled-components)
- saves times on the setup
- less lines of css
- speeds up the development
- normalize.css
- small CSS file that provides cross-browser consistency in the default styling of HTML elements.
- normalize docs

```
npm install normalize.css
```

- import 'normalize.css' in index.js
- SET BEFORE 'index.css'
- replace contents of index.css
- if any questions about normalize or specific styles
- Coding Addict - Default Starter Video
- Repo - Default Starter Repo

Landing Page

- zoom level 175%
- markdown preview extension
- get something on the screen
- react router and styled components right after
- create pages directory in the source
- for now Landing.js

- create component (snippets extension)
- setup basic return

```
<h4>Landing Page<h4>
```

- import logo.svg and main.svg
- import Landing in App.js and render

Styled Components

- CSS in JS
- Styled Components
- have logic and styles in component
- no name collisions
- apply javascript logic
- Styled Components Docs
- Styled Components Course

```
npm install styled-components
```

```
import styled from 'styled-components'

const El = styled.el`
  // styles go here
`
```

- no name collisions, since unique class
- vscode-styled-components extension
- colors and bugs
- style entire react component

```
const Wrapper = styled.el``

const Component = () => {
  return (
    <Wrapper>
      <h1> Component</h1>
    </Wrapper>
  )
}
```

- only responsible for styling

- wrappers folder in assets

Logo and Images

- logo built in Figma
- Cool Images

Logo

- create **components** folder in source
- create Logo.js
- move import and image logic
- export as default
- utilize index.js

React Router

- Version 6
- React Router Docs

```
npm install history@5 react-router-dom@6
```

- import four components

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom'
```

- Connect to browser's URL with BrowserRouter
- Routes instead of Switch

```
<BrowserRouter>
    <Routes>
      <Route path="/" element={<div>Dashboard</div>} />
      <Route path="/register" element={<div>Register</div>} />
      <Route path="/landing" element={<Landing />} />
      <Route path="*" element={<div>Error</div>}>
    </Routes>
</BrowserRouter>
```

```
<nav>
  <Link to='/'>Dashboard</Link>
  <Link to='/register'>Register</Link>
  <Link to='/landing'>Home</Link>
</nav>
```

- go to Landing.js

```
import { Link } from 'react-router-dom'

return (
  <Link to='/register' className='btn btn-hero'>
    Login / Register
  </Link>
)
```

Setup Pages

- create Error, Register, Dashboard pages
- basic return
- create index.js
- import all the pages
- export one by one
- basically the same, as in components
- import App.js
- add to element={}
- remove temp navbar

Error Page

```
import { Link } from 'react-router-dom'
import img from '../assets/images/not-found.svg'
import Wrapper from '../assets/wrappers/ErrorPage'

return (
  <Wrapper className='full-page'>
    <div>
      <img src={img} alt='not found' />
      <h3>text</h3>
      <p>text</p>
      <Link to='/'>back home</Link>
    </div>
  </Wrapper>
)
```

Auto Imports

- use while developing
- only sparingly while recording
- better picture
- messes with flow
- just my preference
- still use them, just not all the time

Register Page - Setup

- show preview in Browser and themes

```jsx
import { useState, useEffect } from 'react'
import { Logo } from '../components'
import Wrapper from '../assets/wrappers/RegisterPage'
// global context and useNavigate later

const initialState = {
  name: '',
  email: '',
  password: '',
  isMember: true,
}
// if possible prefer local state
// global state

function Register() {
  const [values, setValues] = useState(initialState)

  // global context and useNavigate later

  const handleChange = (e) => {
    console.log(e.target)
  }

  const onSubmit = (e) => {
    e.preventDefault()
    console.log(e.target)
  }
  return (
    <Wrapper className='full-page'>
      <form className='form' onSubmit={onSubmit}>
        <Logo />
        <h3>Login</h3>

        {/* name field */}
        <div className='form-row'>
          <label htmlFor='name' className='form-label'>
            name
          </label>

          <input
            type='text'
            value={values.name}
            name='name'
            onChange={handleChange}
            className='form-input'
          />
        </div>

        <button type='submit' className='btn btn-block'>
          submit
```

```
        </button>
      </form>
    </Wrapper>
  )
}
```

FormRow Component

- create FormRow.js in **components**
- setup import/export
- setup one for email and password
- hint "type,name,value"

```
const FormRow = ({ type, name, value, handleChange, labelText }) => {
  return (
    <div className='form-row'>
      <label htmlFor={name} className='form-label'>
        {labelText || name}
      </label>

      <input
        type={type}
        value={value}
        name={name}
        onChange={handleChange}
        className='form-input'
      />
    </div>
  )
}

export default FormRow
```

Alert Component

- right away setup as component
- create Alert.js in **components**

```
const Alert = () => {
  return <div className='alert alert-danger'>alert goes here</div>
}

export default Alert
```

- setup import/export
- alert-danger or alert-success

- eventually setup in global context
- showAlert in initialState (true || false)
- right after h3 login

```
values.showAlert && <Alert />
```

Toggle Member

```
const toggleMember = () => {
  setValues({ ...values, isMember: !values.isMember })
}

return (
  <Wrapper>
    {/* control h3 */}

    <h3>{values.isMember ? 'Login' : 'Register'}</h3>

    {/* toggle name */}

    {!values.isMember && (
      <FormRow
        type='text'
        name='name'
        value={values.name}
        handleChange={handleChange}
      />
    )}

    {/* right after submit btn */}
    {/* toggle button */}

    <p>
      {values.isMember ? 'Not a member yet?' : 'Already a member?'}

      <button type='button' onClick={toggleMember} className='member-btn'>
        {values.isMember ? 'Register' : 'Login'}
      </button>
    </p>
  </Wrapper>
)
```

Global Context

- in src create **context** directory
- actions.js
- reducer.js

- appContext.js

```
import React, { useState, useReducer, useContext } from 'react'

export const initialState = {
  isLoading: false,
  showAlert: false,
  alertText: '',
  alertType: '',
}
const AppContext = React.createContext()
const AppProvider = ({ children }) => {
  const [state, setState] = useState(initialState)

  return (
    <AppContext.Provider
      value={{
        ...state,
      }}
    >
      {children}
    </AppContext.Provider>
  )
}
// make sure use
export const useAppContext = () => {
  return useContext(AppContext)
}

export { AppProvider }
```

- index.js

```
import { AppProvider } from './context/appContext'

ReactDOM.render(
  <React.StrictMode>
    <AppProvider>
      <App />
    </AppProvider>
  </React.StrictMode>,
  document.getElementById('root')
)
```

- Register.js

```
import { useAppContext } from '../context/appContext'

const { isLoading, showAlert } = useAppContext()
```

- switch to global showAlert

useReducer

- React Tutorial
- useReducer vs Redux
- multiple reducers vs one

Wire Up Reducer

```
reducer.js

const reducer = (state, action) => {
  throw new Error(`no such action :${action.type}`)
}
export default reducer


appContext.js

import reducer from './reducer'

const [state, dispatch] = useReducer(reducer, initialState)
```

Display Alert

```
actions.js

export const DISPLAY_ALERT = 'SHOW_ALERT'
```

- setup imports (reducer and appContext)

```
appContext.js

const displayAlert() =>{
  dispatch({type:DISPLAY_ALERT})
}
```

reducer.js

```
if (action.type === DISPLAY_ALERT) {
  return {
    ...state,
    showAlert: true,
    alertType: 'danger',
    alertText: 'Please provide all values!',
  }
}
```

Alert.js in Components

```
import { useAppContext } from '../context/appContext'

const Alert = () => {
  const { alertType, alertText } = useAppContext()
  return <div className={`alert alert-${alertType}`}>{alertText}</div>
}
```

Display Alert

- JS Nuggets - Dynamic Object Keys

appContext.js

```
const handleChange = (e) => {
  setValues({ ...values, [e.target.name]: e.target.value })
}
```

- get displayAlert function

appContext.js

```
const onSubmit = (e) => {
  e.preventDefault()
  const { name, email, password, isMember } = values
  if (!email || !password || (!isMember && !name)) {
    displayAlert()
    return
  }
  console.log(values)
}
```

Clear Alert

- technically optional

actions.js

```
export const CLEAR_ALERT = 'CLEAR_ALERT'
```

- setup imports (reducer and appContext)

reducer.js

```
if (action.type === CLEAR_ALERT) {
  return {
    ...state,
    showAlert: false,
    alertType: '',
    alertText: '',
  }
}
```

appContext.js

```
const displayAlert = () => {
  dispatch({
    type: DISPLAY_ALERT,
  })
  clearAlert()
}

const clearAlert = () => {
  setTimeout(() => {
    dispatch({
      type: CLEAR_ALERT,
    })
  }, 3000)
}
```

Setup Server

- stop the dev server in client
- cd ..
- start setting up our server
- setup package.json

```
npm init -y
```

- create server.js
- console.log('server running...')

```
node server
```

ES6 vs CommonJS

```
CommonJS

const express = require('express')
const app = express()


ES6

import express from 'express'
const app = express()
```

- file extension .mjs

```
package.json

"type":"module"
```

Nodemon and Basic Express Server

```
npm install nodemon --save-dev


package.json

"start":"nodemon server"



npm install express
```

```
import express from 'express'
const app = express()

app.get('/', (req, res) => {
  res.send('Welcome!')
})

const port = process.env.PORT || 5000

app.listen(port, () => console.log(`Server is listening on port ${port}...`))
```

Jobify

Track Your Job Search

Project in Action - Jobify

Support

Find the App Useful? You can always buy me a coffee

Run The App Locally

```
npm run install-dependencies
```

- rename .env.temp to .env
- setup values for - MONGO_URL, JWT_SECRET, JWT_LIFETIME

```
npm start
```

- visit url http://localhost:3000/

Setup React App

- create **client** folder
- open terminal

```
cd client
```

```
npx create-react-app .
```

```
npm start
```

- set editor/browser side by side
- copy/paste assets from complete project

Spring Cleaning

- in src remove
- App.css
- App.test.js
- logo.svg
- reportWebVitals.js
- setupTests.js
- fix App.js and index.js

Title and Favicon

- change title in public/index.html
- replace favicon.ico in public
- resource Generate Favicons

Normalize.css and Global Styles

- CSS in JS (styled-components)
- saves times on the setup
- less lines of css
- speeds up the development
- normalize.css
- small CSS file that provides cross-browser consistency in the default styling of HTML elements.
- normalize docs

```
npm install normalize.css
```

- import 'normalize.css' in index.js
- SET BEFORE 'index.css'
- replace contents of index.css
- if any questions about normalize or specific styles
- Coding Addict - Default Starter Video
- Repo - Default Starter Repo

Landing Page

- zoom level 175%
- markdown preview extension
- get something on the screen
- react router and styled components right after
- create pages directory in the source
- for now Landing.js
- create component (snippets extension)
- setup basic return

```
<h4>Landing Page<h4>
```

- import logo.svg and main.svg
- import Landing in App.js and render

Styled Components

- CSS in JS
- Styled Components
- have logic and styles in component
- no name collisions
- apply javascript logic
- Styled Components Docs
- Styled Components Course

```
npm install styled-components
```

```
import styled from 'styled-components'

const El = styled.el`
  // styles go here
`
```

- no name collisions, since unique class
- vscode-styled-components extension
- colors and bugs
- style entire react component

```
const Wrapper = styled.el``

const Component = () => {
  return (
    <Wrapper>
      <h1> Component</h1>
    </Wrapper>
  )
}
```

- only responsible for styling
- wrappers folder in assets

Logo and Images

- logo built in Figma
- Cool Images

Logo

- create **components** folder in source
- create Logo.js
- move import and image logic
- export as default
- utilize index.js

React Router

- Version 6
- React Router Docs

```
npm install history@5 react-router-dom@6
```

- import four components

```
import { BrowserRouter, Routes, Route, Link } from 'react-router-dom'
```

- Connect to browser's URL with BrowserRouter
- Routes instead of Switch

```
<BrowserRouter>
    <Routes>
      <Route path="/" element={<div>Dashboard</div>} />
      <Route path="/register" element={<div>Register</div>} />
      <Route path="/landing" element={<Landing />} />
      <Route path="*" element={<div>Error</div>}>
    </Routes>
</BrowserRouter>
```

```
<nav>
  <Link to='/'>Dashboard</Link>
  <Link to='/register'>Register</Link>
  <Link to='/landing'>Home</Link>
</nav>
```

- go to Landing.js

```
import { Link } from 'react-router-dom'

return (
  <Link to='/register' className='btn btn-hero'>
    Login / Register
  </Link>
)
```

Setup Pages

- create Error, Register, Dashboard pages
- basic return
- create index.js
- import all the pages
- export one by one
- basically the same, as in components
- import App.js
- add to element={}
- remove temp navbar

Error Page

```
import { Link } from 'react-router-dom'
import img from '../assets/images/not-found.svg'
import Wrapper from '../assets/wrappers/ErrorPage'

return (
  <Wrapper className='full-page'>
    <div>
      <img src={img} alt='not found' />
      <h3>text</h3>
      <p>text</p>
      <Link to='/'>back home</Link>
    </div>
  </Wrapper>
)
```

Auto Imports

- use while developing
- only sparingly while recording
- better picture
- messes with flow
- just my preference
- still use them, just not all the time

Register Page - Setup

- show preview in Browser and themes

```jsx
import { useState, useEffect } from 'react'
import { Logo } from '../components'
import Wrapper from '../assets/wrappers/RegisterPage'
// global context and useNavigate later

const initialState = {
  name: '',
  email: '',
  password: '',
  isMember: true,
}
// if possible prefer local state
// global state

function Register() {
  const [values, setValues] = useState(initialState)

  // global context and useNavigate later

  const handleChange = (e) => {
    console.log(e.target)
  }

  const onSubmit = (e) => {
    e.preventDefault()
    console.log(e.target)
  }
  return (
    <Wrapper className='full-page'>
      <form className='form' onSubmit={onSubmit}>
        <Logo />
        <h3>Login</h3>

        {/* name field */}
        <div className='form-row'>
          <label htmlFor='name' className='form-label'>
            name
          </label>

          <input
            type='text'
            value={values.name}
            name='name'
            onChange={handleChange}
            className='form-input'
          />
        </div>

        <button type='submit' className='btn btn-block'>
          submit
```

```
        </button>
      </form>
    </Wrapper>
  )
}
```

FormRow Component

- create FormRow.js in **components**
- setup import/export
- setup one for email and password
- hint "type,name,value"

```
const FormRow = ({ type, name, value, handleChange, labelText }) => {
  return (
    <div className='form-row'>
      <label htmlFor={name} className='form-label'>
        {labelText || name}
      </label>

      <input
        type={type}
        value={value}
        name={name}
        onChange={handleChange}
        className='form-input'
      />
    </div>
  )
}

export default FormRow
```

Alert Component

- right away setup as component
- create Alert.js in **components**

```
const Alert = () => {
  return <div className='alert alert-danger'>alert goes here</div>
}

export default Alert
```

- setup import/export
- alert-danger or alert-success

- eventually setup in global context
- showAlert in initialState (true || false)
- right after h3 login

```
values.showAlert && <Alert />
```

Toggle Member

```
const toggleMember = () => {
  setValues({ ...values, isMember: !values.isMember })
}

return (
  <Wrapper>
    {/* control h3 */}

    <h3>{values.isMember ? 'Login' : 'Register'}</h3>

    {/* toggle name */}

    {!values.isMember && (
      <FormRow
        type='text'
        name='name'
        value={values.name}
        handleChange={handleChange}
      />
    )}

    {/* right after submit btn */}
    {/* toggle button */}

    <p>
      {values.isMember ? 'Not a member yet?' : 'Already a member?'}

      <button type='button' onClick={toggleMember} className='member-btn'>
        {values.isMember ? 'Register' : 'Login'}
      </button>
    </p>
  </Wrapper>
)
```

Global Context

- in src create **context** directory
- actions.js
- reducer.js

- appContext.js

```
import React, { useState, useReducer, useContext } from 'react'

export const initialState = {
  isLoading: false,
  showAlert: false,
  alertText: '',
  alertType: '',
}
const AppContext = React.createContext()
const AppProvider = ({ children }) => {
  const [state, setState] = useState(initialState)

  return (
    <AppContext.Provider
      value={{
        ...state,
      }}
    >
      {children}
    </AppContext.Provider>
  )
}
// make sure use
export const useAppContext = () => {
  return useContext(AppContext)
}

export { AppProvider }
```

- index.js

```
import { AppProvider } from './context/appContext'

ReactDOM.render(
  <React.StrictMode>
    <AppProvider>
      <App />
    </AppProvider>
  </React.StrictMode>,
  document.getElementById('root')
)
```

- Register.js

```
import { useAppContext } from '../context/appContext'

const { isLoading, showAlert } = useAppContext()
```

- switch to global showAlert

useReducer

- React Tutorial
- useReducer vs Redux
- multiple reducers vs one

Wire Up Reducer

```
reducer.js

const reducer = (state, action) => {
  throw new Error(`no such action :${action.type}`)
}
export default reducer


appContext.js

import reducer from './reducer'

const [state, dispatch] = useReducer(reducer, initialState)
```

Display Alert

```
actions.js

export const DISPLAY_ALERT = 'SHOW_ALERT'
```

- setup imports (reducer and appContext)

```
appContext.js

const displayAlert() =>{
  dispatch({type:DISPLAY_ALERT})
}
```

reducer.js

```
if (action.type === DISPLAY_ALERT) {
  return {
    ...state,
    showAlert: true,
    alertType: 'danger',
    alertText: 'Please provide all values!',
  }
}
```

Alert.js in Components

```
import { useAppContext } from '../context/appContext'

const Alert = () => {
  const { alertType, alertText } = useAppContext()
  return <div className={`alert alert-${alertType}`}>{alertText}</div>
}
```

Display Alert

- JS Nuggets - Dynamic Object Keys

appContext.js

```
const handleChange = (e) => {
  setValues({ ...values, [e.target.name]: e.target.value })
}
```

- get displayAlert function

appContext.js

```
const onSubmit = (e) => {
  e.preventDefault()
  const { name, email, password, isMember } = values
  if (!email || !password || (!isMember && !name)) {
    displayAlert()
    return
  }
  console.log(values)
}
```

Clear Alert

- technically optional

actions.js

```
export const CLEAR_ALERT = 'CLEAR_ALERT'
```

- setup imports (reducer and appContext)

reducer.js

```
if (action.type === CLEAR_ALERT) {
  return {
    ...state,
    showAlert: false,
    alertType: '',
    alertText: '',
  }
}
```

appContext.js

```
const displayAlert = () => {
  dispatch({
    type: DISPLAY_ALERT,
  })
  clearAlert()
}

const clearAlert = () => {
  setTimeout(() => {
    dispatch({
      type: CLEAR_ALERT,
    })
  }, 3000)
}
```

Setup Server

- stop the dev server in client
- cd ..
- start setting up our server
- setup package.json

```
npm init -y
```

- create server.js
- console.log('server running...')

```
node server
```

ES6 vs CommonJS

```
CommonJS

const express = require('express')
const app = express()


ES6

import express from 'express'
const app = express()
```

- file extension .mjs

```
package.json

"type":"module"
```

Nodemon and Basic Express Server

```
npm install nodemon --save-dev


package.json

"start":"nodemon server"



npm install express
```

```
import express from 'express'
const app = express()

app.get('/', (req, res) => {
  res.send('Welcome!')
})

const port = process.env.PORT || 5000

app.listen(port, () => console.log(`Server is listening on port ${port}...`))
```

Not Found Middleware

- in the root create **middleware** folder
- not-found.js
- setup function
- return 404 with message 'Route does not exist'
- import in server.js
- make sure to use .js extension
- place after home route

Error Middleware

- in the middleware create error-handler.js
- setup function
- accept 4 parameters, first one error
- log error
- return 500
- json({msg:'there was an error'})
- import in the server.js
- make sure to use .js extension
- place it last
- eventually handle Mongoose Errors, just like in the node-express
- showcase with async errors

ENV Variables

```
npm install dotenv
```

- import dotenv from 'dotenv'
- dotenv.config()

- create .env
- PORT=4000
- .gitignore
- /node_modules
- .env

Connect to MongoDB

- switched back to PORT=5000
- remove Error from '/'
- existing MongoDB Atlas Account

```
npm install mongoose
```

- create **db** folder
- create connect.js
- setup connectDB(url)
- in server.js create start() function
- get connection string
- setup as MONGO_URL in .env
- provide credentials and DB Name

Auth Controller and Route Structure

- create **controllers**
- authController.js
- create async functions

```
export { register, login, updateUser }
```

- return res.send('function name')
- create **routes** folder
- authRoutes.js
- setup express router
- import functions from authController.js

```
router.route('/register').post(register)
router.route('/login').post(login)
router.route('/updateUser').patch(updateUser)

export default router
```

- import authRouter in server.js

```
app.use('/api/v1/auth', authRouter)
```

Jobs Controller and Route Structure

- jobsController.js
- create async functions

```
export { createJob, deleteJob, getAllJobs, updateJob, showStats }
```

- return res.send('function name')
- jobsRoutes.js
- setup express router
- import functions from jobsController.js

```
router.route('/').post(createJob).get(getAllJobs)
// place before :id
router.route('/stats').get(showStats)
router.route('/:id').delete(deleteJob).patch(updateJob)

export default router
```

- in server.js jobsRouter

```
app.use('/api/v1/jobs', jobsRouter)
```

Postman

- URL global var
- JOBIFY Collection
- auth and jobs folders
- setup routes

User Model

- **models** folder
- User.js
- setup schema
- name, email, password, lastName, location
- all {type:String}

Validate Email

```
validate:{
  validator:(field)=> {return 2 > 1},
  message:'Please provide valid email'
  }
```

- Validator Package

```
npm install validator
```

- import in User.js
- validator.isEmail

Register User - Initial Setup

- authController
- import User model
- setup temporary try/catch
- await User.create(req.body)
- if success 201 with json({user}) (temp)
- if error 500 with json({msg:'there was an error'})

Pass Error to Error Handler

- next(error)

Express-Async-Errors Package

- remove try/catch
- Express-Async-Errors

```
npm install express-async-errors
```

- in server.js

- import 'express-async-errors'
- use throw Error('error') instead of next(error)

Http Status Codes

- constants for status codes
- personal preference
- provides consistency
- less bugs
- easier to read/manage
- Http Status Codes

```
npm install http-status-codes
```

- import/setup in authController and error-handler
- setup defaultError

Custom Errors

Refactor Errors

- create errors folder
- create custom-api, bad-request, not-found, index.js files
- add proper imports
- setup index.js just like in the front-end
- import {BadRequestError} in authController
- gotcha "errors/index.js"

Hash Passwords

- one way street, only compare hashed values
- bcrypt.js

```
npm install bcryptjs
```

- User Model
- import bcrypt from 'bcryptjs'
- await genSalt(10)
- await hash(password , salt)
- await compare(requestPassword , currentPassword)

- mongoose middleware
- UserSchema.pre('save',async function(){
  "this" points to instance created by UserSchema
  })

Mongoose - Custom Instance Methods

Custom Instance Methods

- UserSchema.methods.createJWT = function(){console.log(this)}
- register controller
- right after User.create()
- invoke user.createJWT()

JWT

- token
- jsonwebtoken

```
npm install jsonwebtoken
```

- User Model
- import jwt from 'jsonwebtoken'
- jwt.sign(payload,secret,options)
- createJWT

```
return jwt.sign({ userId: this._id }, 'jwtSecret', { expiresIn: '1d' })
```

```
return jwt.sign({ userId: this._id }, process.env.JWT_SECRET, {
  expiresIn: process.env.JWT_LIFETIME,
})
```

JWT_SECRET and JWT_LIFETIME

- Keys Generator
- RESTART SERVER!!!!

Complete Register

- password : {select:false}
- complete response

Concurrently

- front-end and backend (server)
- run separate terminals
- concurrently

```
npm install concurrently --save-dev
```

- package.json

```
// --kill-others switch, all commands are killed if one dies
// --prefix client - folder
// cd client && npm start
// escape quotes

"scripts": {
    "server": "nodemon server --ignore client",
    "client": "npm start --prefix client",
    "start": "concurrently --kill-others-on-fail \"npm run server\" \" npm run client\""
  },
```

Cors Error

Cors Error

- two fixes (cors package and proxy)

Cors Package

cors package

```
npm install cors
```

```
import cors from 'cors'

app.use(cors())
```

Proxy

- access from anywhere
- don't want to use full url

cra proxy

```
"proxy":"http://localhost:5000"
```

- my preference to remove trailing slash /
- restart app

Register User - Setup

```
appContext.js

const initialState = {
  user: null,
  token: null,
  userLocation: '',
}
```

- actions.js REGISTER_USER_BEGIN,SUCCESS,ERROR
- import reducer,appContext

```
appContext.js
const registerUser = async (currentUser) => {
  console.log(currentUser)
}
```

- import in Register.js

```
Register.js

const currentUser = { name, email, password }
if (isMember) {
  console.log('already a member')
} else {
  registerUser(currentUser)
}

return (
  <button type='submit' className='btn btn-block' disabled={isLoading}>
    submit
  </button>
)
```

Axios

- axios docs
- stop app

- cd client

```
npm install axios
```

- cd ..
- restart app

Register User - Complete

appContext.js

```
import axios from 'axios'

const registerUser = async (currentUser) => {
  dispatch({ type: REGISTER_USER_BEGIN })
  try {
    const response = await axios.post('/api/v1/auth/register', currentUser)
    console.log(response)
    const { user, token, location } = response.data
    dispatch({
      type: REGISTER_USER_SUCCESS,
      payload: {
        user,
        token,
        location,
      },
    })

    // will add later
    // addUserToLocalStorage({
    //   user,
    //   token,
    //   location,
    // })
  } catch (error) {
    console.log(error.response)
    dispatch({
      type: REGISTER_USER_ERROR,
      payload: { msg: error.response.data.msg },
    })
  }
  clearAlert()
}
```

```
reducer.js
if (action.type === REGISTER_USER_BEGIN) {
  return { ...state, isLoading: true }
}
if (action.type === REGISTER_USER_SUCCESS) {
  return {
    ...state,
    user: action.payload.user,
    token: action.payload.token,
    userLocation: action.payload.location,
    jobLocation: action.payload.location,
    isLoading: false,
    showAlert: true,
    alertType: 'success',
    alertText: 'User Created! Redirecting...',
  }
}
if (action.type === REGISTER_USER_ERROR) {
  return {
    ...state,
    isLoading: false,
    showAlert: true,
    alertType: 'danger',
    alertText: action.payload.msg,
  }
}
```

Navigate To Dashboard

```
Register.js
import { useEffect } from 'react'
import { useNavigate } from 'react-router-dom'

const Register = () => {
  const { user } = useAppContext()
  const navigate = useNavigate()

  useEffect(() => {
    if (user) {
      setTimeout(() => {
        navigate('/')
      }, 3000)
    }
  }, [user, navigate])
}
```

Local Storage

appContext.js

```
const addUserToLocalStorage = ({ user, token, location }) => {
  localStorage.setItem('user', JSON.stringify(user))
  localStorage.setItem('token', token)
  localStorage.setItem('location', location)
}

const removeUserFromLocalStorage = () => {
  localStorage.removeItem('token')
  localStorage.removeItem('user')
  localStorage.removeItem('location')
}

const registerUser = async (currentUser) => {
  // in try block
  addUserToLocalStorage({
    user,
    token,
    location,
  })
}

// set as default
const token = localStorage.getItem('token')
const user = localStorage.getItem('user')
const userLocation = localStorage.getItem('location')

const initialState = {
  user: user ? JSON.parse(user) : null,
  token: token,
  userLocation: userLocation || '',
  jobLocation: userLocation || '',
}
```

Morgan Package

- http logger middleware for node.js
- morgan docs

```
npm install morgan
```

```
import morgan from 'morgan'

if (process.env.NODE_ENV !== 'production') {
  app.use(morgan('dev'))
}
```

UnauthenticatedError

- unauthenticated.js in errors
- import/export

```
import { StatusCodes } from 'http-status-codes'
import CustomAPIError from './custom-api.js'

class UnauthenticatedError extends CustomAPIError {
  constructor(message) {
    super(message)
    this.statusCode = StatusCodes.UNAUTHORIZED
  }
}
```

Compare Password

```
User.js in models

UserSchema.methods.comparePassword = async function (candidatePassword) {
  const isMatch = await bcrypt.compare(candidatePassword, this.password)
  return isMatch
}


authController.js
const login = async (req, res) => {
  const { email, password } = req.body
  if (!email || !password) {
    throw new BadRequestError('Please provide all values')
  }
  const user = await User.findOne({ email }).select('+password')

  if (!user) {
    throw new UnauthenticatedError('Invalid Credentials')
  }
  const isPasswordCorrect = await user.comparePassword(password)
  if (!isPasswordCorrect) {
    throw new UnauthenticatedError('Invalid Credentials')
  }
  const token = user.createJWT()
  user.password = undefined
  res.status(StatusCodes.OK).json({ user, token, location: user.location })
}
```

- test in Postman

Login User - Setup

- actions.js LOGIN_USER_BEGIN,SUCCESS,ERROR
- import reducer,appContext

```
appContext.js
const loginUser = async (currentUser) => {
  console.log(currentUser)
}
```

- import in Register.js

```
Register.js

if (isMember) {
  loginUser(currentUser)
} else {
  registerUser(currentUser)
}
```

Login User - Complete

```
appContext.js
const loginUser = async (currentUser) => {
  dispatch({ type: LOGIN_USER_BEGIN })
  try {
    const { data } = await axios.post('/api/v1/auth/login', currentUser)
    const { user, token, location } = data

    dispatch({
      type: LOGIN_USER_SUCCESS,
      payload: { user, token, location },
    })

    addUserToLocalStorage({ user, token, location })
  } catch (error) {
    dispatch({
      type: LOGIN_USER_ERROR,
      payload: { msg: error.response.data.msg },
    })
  }
  clearAlert()
}
```

reducer.js

```
if (action.type === LOGIN_USER_BEGIN) {
  return {
    ...state,
    isLoading: true,
  }
}
if (action.type === LOGIN_USER_SUCCESS) {
  return {
    ...state,
    isLoading: false,
    user: action.payload.user,
    token: action.payload.token,
    userLocation: action.payload.location,
    jobLocation: action.payload.location,
    showAlert: true,
    alertType: 'success',
    alertText: 'Login Successful! Redirecting...',
  }
}
if (action.type === LOGIN_USER_ERROR) {
  return {
    ...state,
    isLoading: false,
    showAlert: true,
    alertType: 'danger',
    alertText: action.payload.msg,
  }
}
```

Refactor

actions.js

```
export const SETUP_USER_BEGIN = 'SETUP_USER_BEGIN'
export const SETUP_USER_SUCCESS = 'SETUP_USER_SUCCESS'
export const SETUP_USER_ERROR = 'SETUP_USER_ERROR'
```

appContext.js

```
const setupUser = async ({ currentUser, endPoint, alertText }) => {
  dispatch({ type: SETUP_USER_BEGIN })
  try {
    const { data } = await axios.post(`/api/v1/auth/${endPoint}`, currentUser)

    const { user, token, location } = data
    dispatch({
      type: SETUP_USER_SUCCESS,
      payload: { user, token, location, alertText },
    })
    addUserToLocalStorage({ user, token, location })
  } catch (error) {
    dispatch({
      type: SETUP_USER_ERROR,
      payload: { msg: error.response.data.msg },
    })
  }
  clearAlert()
}
```

reducer.js

```
if (action.type === SETUP_USER_BEGIN) {
  return { ...state, isLoading: true }
}
if (action.type === SETUP_USER_SUCCESS) {
  return {
    ...state,
    isLoading: false,
    token: action.payload.token,
    user: action.payload.user,
    userLocation: action.payload.location,
    jobLocation: action.payload.location,
    showAlert: true,
    alertType: 'success',
    alertText: action.payload.alertText,
  }
}
if (action.type === SETUP_USER_ERROR) {
  return {
    ...state,
    isLoading: false,
    showAlert: true,
    alertType: 'danger',
    alertText: action.payload.msg,
  }
}
```

- import/export

Register.js

```
const onSubmit = (e) => {
  e.preventDefault()
  const { name, email, password, isMember } = values
  if (!email || !password || (!isMember && !name)) {
    displayAlert()
    return
  }
  const currentUser = { name, email, password }
  if (isMember) {
    setupUser({
      currentUser,
      endPoint: 'login',
      alertText: 'Login Successful! Redirecting...',
    })
  } else {
    setupUser({
      currentUser,
      endPoint: 'register',
      alertText: 'User Created! Redirecting...',
    })
  }
}
```

Nested Pages in React Router 6

Dashboard pages

- delete Dashboard.js
- fix imports/exports
- replace in home route

```
<Route path='/' element={<div>dashboard</div>} />
```

- create **dashboard** directory in pages
- create AddJob,AllJobs,Profile,Stats,SharedLayout, index.js
- setup basic returns

```
return <h1>Add Job Page</h1>
```

- export all with index.js (just like components)
- import all pages in App.js

Nested Structure

App.js

```
<Route path='/' >
  <Route path="stats" element={<Stats />} />
  <Route path='all-jobs' element={<AllJobs />}></Route>
  <Route path='add-job' element={<AddJob />}></Route>
  <Route path='profile' element={<Profile />}></Route>
</Route>
```

Shared Layout

App.js

```
<Route path='/' element={<SharedLayout/>} >
```

SharedLayout.js

```
import { Outlet, Link } from 'react-router-dom'
import Wrapper from '../../assets/wrappers/SharedLayout'

const SharedLayout = () => {
  return (
    <Wrapper>
      <nav>
        <Link to='all-jobs'>all jobs</Link>
        <Link to='add-job'>all jobs</Link>
      </nav>
      <Outlet />
    </Wrapper>
  )
}

export default SharedLayout
```

App.js

```
<Route index element={<Stats/>} >
```

Protected Route

- create ProtectedRoute.js in pages
- import/export
- wrap SharedLayout in App.js

```
<Route
  path='/'
  element={
    <ProtectedRoute>
      <SharedLayout />
    </ProtectedRoute>
  }
/>
```

ProtectedRoute.js

```
import { Navigate } from 'react-router-dom'
import { useAppContext } from '../context/appContext'

const ProtectedRoute = ({ children }) => {
  const { user } = useAppContext()
  if (!user) {
    return <Navigate to='/landing' />
  }
  return children
}
```

Navbar, SmallSidebar, BigSidebar

- create Navbar, SmallSidebar, BigSidebar in components
- import Wrappers from assets/wrappers
- simple return
- import/export

SharedLayout.js

```
import { Outlet } from 'react-router-dom'
import { Navbar, SmallSidebar, BigSidebar } from '../../components'
import Wrapper from '../../assets/wrappers/SharedLayout'

const SharedLayout = () => {
  const { user } = useAppContext()
  return (
    <>
      <Wrapper>
        <main className='dashboard'>
          <SmallSidebar />
          <BigSidebar />
          <div>
            <Navbar />
            <div className='dashboard-page'>
              <Outlet />
            </div>
          </div>
        </main>
      </Wrapper>
    </>
  )
}

export default SharedLayout
```

React Icons

React Icons

```
npm install react-icons
```

Navbar.js

```
import Wrapper from '../assets/wrappers/Navbar'
import {FaHome} from 'react-icons/fa'
const Navbar = () => {
  return (
    <Wrapper>
      <h4>navbar</h4>
      <FaHome>
    </Wrapper>
  )
}

export default Navbar
```

Navbar Setup

Navbar.js

```
import { useState } from 'react'
import { FaAlignLeft, FaUserCircle, FaCaretDown } from 'react-icons/fa'
import { useAppContext } from '../context/appContext'
import Logo from './Logo'
import Wrapper from '../assets/wrappers/Navbar'
const Navbar = () => {
  return (
    <Wrapper>
      <div className='nav-center'>
        <button
          className='toggle-btn'
          onClick={() => console.log('toggle sidebar')}
        >
          <FaAlignLeft />
        </button>

        <div>
          <Logo />
          <h3 className='logo-text'>dashboard</h3>
        </div>

        <div className='btn-container'>
          <button className='btn' onClick={() => console.log('show logout')}>
            <FaUserCircle />
            john
            <FaCaretDown />
          </button>
          <div className='dropdown show-dropdown'>
            <button
              onClick={() => console.log('logout user')}
              className='dropdown-btn'
            >
              logout
            </button>
          </div>
        </div>
      </div>
    </Wrapper>
  )
}

export default Navbar
```

Toggle Sidebar

actions.js

```
export const TOGGLE_SIDEBAR = 'TOGGLE_SIDEBAR'
```

- import/export

appContext.js

```
const initialState = {
  showSidebar: false,
}

const toggleSidebar = () => {
  dispatch({ type: TOGGLE_SIDEBAR })
}
```

reducer.js

```
if (action.type === TOGGLE_SIDEBAR) {
  return { ...state, showSidebar: !state.showSidebar }
}
```

Navbar.js

```
const { toggleSidebar } = useAppContext()

return (
  <button className='toggle-btn' onClick={toggleSidebar}>
    <FaAlignLeft />
  </button>
)
```

Toggle Dropdown

Navbar.js

```
const [showLogout, setShowLogout] = useState(false)

<div className='btn-container'>
  <button className='btn' onClick={() => setShowLogout(!showLogout)}>
    <FaUserCircle />
      {user.name}
    <FaCaretDown />
  </button>
  <div className={showLogout ? 'dropdown show-dropdown' : 'dropdown'}>
    <button onClick={() => logoutUser()} className='dropdown-btn'>
      logout
    </button>
  </div>
</div>
```

Logout User

actions.js

```
export const LOGOUT_USER = 'LOGOUT_USER'
```

- import/export

appContext.js

```
const logoutUser = () => {
  dispatch({ type: LOGOUT_USER })
  removeUserFromLocalStorage()
}

value={{logoutUser}}
```

reducer.js

```
import { initialState } from './appContext'

if (action.type === LOGOUT_USER) {
  return {
    ...initialState,
    user: null,
    token: null,
    userLocation: '',
    jobLocation: '',
  }
}
```

Navbar.js

```
const { user, logoutUser, toggleSidebar } = useAppContext()

return (
  <div className='btn-container'>
    <button className='btn' onClick={() => setShowLogout(!showLogout)}>
      <FaUserCircle />
      {user.name}
      {user && user.name}
      {user?.name} // optional chaining
      <FaCaretDown />
    </button>
    <div className={showLogout ? 'dropdown show-dropdown' : 'dropdown'}>
      <button onClick={logoutUser} className='dropdown-btn'>
        logout
      </button>
    </div>
  </div>
)
```

Setup Links

- create **utils**in the **src**
- setup links.js

```
import { IoBarChartSharp } from 'react-icons/io5'
import { MdQueryStats } from 'react-icons/md'
import { FaWpforms } from 'react-icons/fa'
import { ImProfile } from 'react-icons/im'

const links = [
  {
    id: 1,
    text: 'stats',
    path: '/',
    icon: <IoBarChartSharp />,
  },
  {
    id: 2,
    text: 'all jobs',
    path: 'all-jobs',
    icon: <MdQueryStats />,
  },
  {
    id: 3,
    text: 'add job',
    path: 'add-job',
    icon: <FaWpforms />,
  },
  {
    id: 4,
    text: 'profile',
    path: 'profile',
    icon: <ImProfile />,
  },
]

export default links
```

Small Sidebar - Setup

SmallSidebar.js

```
import Wrapper from '../assets/wrappers/SmallSidebar'
import { FaTimes } from 'react-icons/fa'
import { useAppContext } from '../context/appContext'
import links from '../utils/links'
import { NavLink } from 'react-router-dom'
import Logo from './Logo'

export const SmallSidebar = () => {
  return (
    <Wrapper>
      <div className='sidebar-container show-sidebar'>
        <div className='content'>
          <button className='close-btn' onClick={() => console.log('toggle')}>
            <FaTimes />
          </button>
          <header>
            <Logo />
          </header>
          <div className='nav-links'>nav links</div>
        </div>
      </div>
    </Wrapper>
  )
}

export default SmallSidebar
```

Small Sidebar - Toggle

SmallSidebar.js

```
const { showSidebar, toggleSidebar } = useAppContext()
```

SmallSidebar.js

```
return (
  <div
    className={
      showSidebar ? 'sidebar-container show-sidebar' : 'sidebar-container'
    }
  ></div>
)
```

SmallSidebar.js

```
return (
  <button className='close-btn' onClick={toggleSidebar}>
    <FaTimes />
  </button>
)
```

Small Sidebar - Nav Links

SmallSidebar.js

```
import { NavLink } from 'react-router-dom'

return (
  <div className='nav-links'>
    {links.map((link) => {
      const { text, path, id, icon } = link

      return (
        <NavLink
          to={path}
          className={({ isActive }) =>
            isActive ? 'nav-link active' : 'nav-link'
          }
          key={id}
          onClick={toggleSidebar}
        >
          <span className='icon'>{icon}</span>
          {text}
        </NavLink>
      )
    })}
  </div>
)
```

Nav Links Component

- in **components** create NavLinks.js
- styles still set from Wrapper
- also can setup in links.js, preference

```
import { NavLink } from 'react-router-dom'
import links from '../utils/links'

const NavLinks = ({ toggleSidebar }) => {
  return (
    <div className='nav-links'>
      {links.map((link) => {
        const { text, path, id, icon } = link

        return (
          <NavLink
            to={path}
            key={id}
            onClick={toggleSidebar}
            className={({ isActive }) =>
              isActive ? 'nav-link active' : 'nav-link'
            }
          >
            <span className='icon'>{icon}</span>
            {text}
          </NavLink>
        )
      })}
    </div>
  )
}

export default NavLinks
```

SmallSidebar.js

```
import NavLinks from './NavLinks'

return <NavLinks toggleSidebar={toggleSidebar}>
```

Big Sidebar

```
import { useAppContext } from '../context/appContext'
import NavLinks from './NavLinks'
import Logo from '../components/Logo'
import Wrapper from '../assets/wrappers/BigSidebar'

const BigSidebar = () => {
  const { showSidebar } = useAppContext()
  return (
    <Wrapper>
      <div
        className={
          showSidebar ? 'sidebar-container ' : 'sidebar-container show-sidebar'
        }
      >
        <div className='content'>
          <header>
            <Logo />
          </header>
          <NavLinks />
        </div>
      </div>
    </Wrapper>
  )
}

export default BigSidebar
```

Authenticate User Setup

- create auth.js in **middleware**

```
const auth = async (req, res, next) => {
  console.log('authenticate user')
  next()
}

export default auth
```

authRoutes.js

```
import authenticateUser from '../middleware/auth.js'

router.route('/updateUser').patch(authenticateUser, updateUser)
```

- two options

server.js

```
import authenticateUser from './middleware/auth.js'
app.use('/api/v1/jobs', authenticateUser, jobsRouter)
```

jobsRoutes.js

```
import authenticateUser from './middleware/auth.js'

// all routes !!!!

router.route('/stats').get(authenticateUser, showStats)
```

Auth - Bearer Schema

Postman

Headers

```
Authorization: Bearer <token>
```

auth.js

```
const auth = async (req, res, next) => {
  const headers = req.headers
  const authHeader = req.headers.authorization
  console.log(headers)
  console.log(authHeader)
  next()
}
```

Postman - Set Token Programmatically

- register and login routes
- Tests

```
const jsonData = pm.response.json()
pm.globals.set('token', jsonData.token)

Type: Bearer

Token: {
  {
    token
  }
}
```

Unauthenticated Error

```
auth.js

import { UnAuthenticatedError } from '../errors/index.js'

const auth = async (req, res, next) => {
  const authHeader = req.headers.authorization

  if (!authHeader) {
    // why, well is it 400 or 404?
    // actually 401
    throw new UnAuthenticatedError('Authentication Invalid')
  }

  next()
}
```

Auth Middleware

```
import jwt from 'jsonwebtoken'
import { UnAuthenticatedError } from '../errors/index.js'

const auth = async (req, res, next) => {
  // check header
  const authHeader = req.headers.authorization
  if (!authHeader || !authHeader.startsWith('Bearer')) {
    throw new UnauthenticatedError('Authentication invalid')
  }
  const token = authHeader.split(' ')[1]

  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET)
    // console.log(payload)
    // attach the user request object
    // req.user = payload
    req.user = { userId: payload.userId }
    next()
  } catch (error) {
    throw new UnauthenticatedError('Authentication invalid')
  }
}

export default auth
```

Update User

```
const updateUser = async (req, res) => {
  const { email, name, lastName, location } = req.body
  if (!email || !name || !lastName || !location) {
    throw new BadRequestError('Please provide all values')
  }

  const user = await User.findOne({ _id: req.user.userId })

  user.email = email
  user.name = name
  user.lastName = lastName
  user.location = location

  await user.save()

  // various setups
  // in this case only id
  // if other properties included, must re-generate

  const token = user.createJWT()
  res.status(StatusCodes.OK).json({
    user,
    token,
    location: user.location,
  })
}
```

Modified Paths

- user.save() vs User.findOneAndUpdate

User.js

```
UserSchema.pre('save', async function () {
  console.log(this.modifiedPaths())
  console.log(this.isModified('name'))

  // if (!this.isModified('password')) return
  // const salt = await bcrypt.genSalt(10)
  // this.password = await bcrypt.hash(this.password, salt)
})
```

Profile Page

appContext.js

```
const updateUser = async (currentUser) => {
  console.log(currentUser)
}

value={{updateUser}}
```

Profile.js

```
import { useState } from 'react'
import { FormRow, Alert } from '../../components'
import { useAppContext } from '../../context/appContext'
import Wrapper from '../../assets/wrappers/DashboardFormPage'

const Profile = () => {
  const { user, showAlert, displayAlert, updateUser, isLoading } =
    useAppContext()
  const [name, setName] = useState(user?.name)
  const [email, setEmail] = useState(user?.email)
  const [lastName, setLastName] = useState(user?.lastName)
  const [location, setLocation] = useState(user?.location)

  const handleSubmit = (e) => {
    e.preventDefault()
    if (!name || !email || !lastName || !location) {
      // test and remove temporary
      displayAlert()
      return
    }

    updateUser({ name, email, lastName, location })
  }
  return (
    <Wrapper>
      <form className='form' onSubmit={handleSubmit}>
        <h3>profile </h3>
        {showAlert && <Alert />}

        {/* name */}
        <div className='form-center'>
          <FormRow
            type='text'
            name='name'
            value={name}
            handleChange={(e) => setName(e.target.value)}
          />
          <FormRow
            labelText='last name'
            type='text'
            name='lastName'
            value={lastName}
            handleChange={(e) => setLastName(e.target.value)}
          />
          <FormRow
            type='email'
            name='email'
            value={email}
```

```
            handleChange={(e) => setEmail(e.target.value)}
          />

          <FormRow
            type='text'
            name='location'
            value={location}
            handleChange={(e) => setLocation(e.target.value)}
          />
          <button className='btn btn-block' type='submit' disabled={isLoading}>
            {isLoading ? 'Please Wait...' : 'save changes'}
          </button>
        </div>
      </form>
    </Wrapper>
  )
}

export default Profile
```

Bearer Token - Manual Approach

appContext.js

```
const updaterUser = async (currentUser) => {
  try {
    const { data } = await axios.patch('/api/v1/auth/updateUser', currentUser, {
      headers: {
        Authorization: `Bearer ${state.token}`,
      },
    })
    console.log(data)
  } catch (error) {
    console.log(error.response)
  }
}
```

Axios - Global Setup

appContext.js

```
axios.defaults.headers.common['Authorization'] = `Bearer ${state.token}`
```

Axios - Setup Instance

AppContext.js

```
const authFetch = axios.create({
  baseURL: '/api/v1',
  headers: {
    Authorization: `Bearer ${state.token}`,
  },
})

const updaterUser = async (currentUser) => {
  try {
    const { data } = await authFetch.patch('/auth/updateUser', currentUser)
  } catch (error) {
    console.log(error.response)
  }
}
```

Axios - Interceptors

- will use instance, but can use axios instead

appContext.js

```
// response interceptor
authFetch.interceptors.request.use(
  (config) => {
    config.headers.common['Authorization'] = `Bearer ${state.token}`
    return config
  },
  (error) => {
    return Promise.reject(error)
  }
)
// response interceptor
authFetch.interceptors.response.use(
  (response) => {
    return response
  },
  (error) => {
    console.log(error.response)
    if (error.response.status === 401) {
      console.log('AUTH ERROR')
    }
    return Promise.reject(error)
  }
)
```

Update User

actions.js

```
export const UPDATE_USER_BEGIN = 'UPDATE_USER_BEGIN'
export const UPDATE_USER_SUCCESS = 'UPDATE_USER_SUCCESS'
export const UPDATE_USER_ERROR = 'UPDATE_USER_ERROR'
```

appContext.js

```
const updateUser = async (currentUser) => {
  dispatch({ type: UPDATE_USER_BEGIN })
  try {
    const { data } = await authFetch.patch('/auth/updateUser', currentUser)

    // no token
    const { user, location, token } = data

    dispatch({
      type: UPDATE_USER_SUCCESS,
      payload: { user, location, token },
    })

    addUserToLocalStorage({ user, location, token })
  } catch (error) {
    dispatch({
      type: UPDATE_USER_ERROR,
      payload: { msg: error.response.data.msg },
    })
  }
  clearAlert()
}
```

```
reducer.js
if (action.type === UPDATE_USER_BEGIN) {
  return { ...state, isLoading: true }
}

if (action.type === UPDATE_USER_SUCCESS) {
  return {
    ...state,
    isLoading: false,
    token:action.payload.token
    user: action.payload.user,
    userLocation: action.payload.location,
    jobLocation: action.payload.location,
    showAlert: true,
    alertType: 'success',
    alertText: 'User Profile Updated!',
  }
}
if (action.type === UPDATE_USER_ERROR) {
  return {
    ...state,
    isLoading: false,
    showAlert: true,
    alertType: 'danger',
    alertText: action.payload.msg,
  }
}
```

401 Error - Logout User

appContext.js

```
// response interceptor
authFetch.interceptors.response.use(
  (response) => {
    return response
  },
  (error) => {
    if (error.response.status === 401) {
      logoutUser()
    }
    return Promise.reject(error)
  }
)

const updateUser = async (currentUser) => {
  dispatch({ type: UPDATE_USER_BEGIN })
  try {
    const { data } = await authFetch.patch('/auth/updateUser', currentUser)

    // no token
    const { user, location } = data

    dispatch({
      type: UPDATE_USER_SUCCESS,
      payload: { user, location, token },
    })

    addUserToLocalStorage({ user, location, token: initialState.token })
  } catch (error) {
    if (error.response.status !== 401) {
      dispatch({
        type: UPDATE_USER_ERROR,
        payload: { msg: error.response.data.msg },
      })
    }
  }
  clearAlert()
}
```

Job Model

- Job Model

Job.js

```
import mongoose from 'mongoose'

const JobSchema = new mongoose.Schema(
  {
    company: {
      type: String,
      required: [true, 'Please provide company name'],
      maxlength: 50,
    },
    position: {
      type: String,
      required: [true, 'Please provide position'],
      maxlength: 100,
    },
    status: {
      type: String,
      enum: ['interview', 'declined', 'pending'],
      default: 'pending',
    },

    jobType: {
      type: String,
      enum: ['full-time', 'part-time', 'remote', 'internship'],
      default: 'full-time',
    },
    jobLocation: {
      type: String,
      default: 'my city',
      required: true,
    },
    createdBy: {
      type: mongoose.Types.ObjectId,
      ref: 'User',
      required: [true, 'Please provide user'],
    },
  },
  { timestamps: true }
)

export default mongoose.model('Job', JobSchema)
```

Create Job

jobsController.js

```
import Job from '../models/Job.js'
import { StatusCodes } from 'http-status-codes'
import { BadRequestError, NotFoundError } from '../errors/index.js'

const createJob = async (req, res) => {
  const { position, company } = req.body

  if (!position || !company) {
    throw new BadRequestError('Please Provide All Values')
  }

  req.body.createdBy = req.user.userId

  const job = await Job.create(req.body)
  res.status(StatusCodes.CREATED).json({ job })
}
```

Job State Values

appContext.js

```
const initialState = {
  isEditing: false,
  editJobId: '',
  position: '',
  company: '',
  // jobLocation
  jobTypeOptions: ['full-time', 'part-time', 'remote', 'internship'],
  jobType: 'full-time',
  statusOptions: ['pending', 'interview', 'declined'],
  status: 'pending',
}
```

AddJob Page - Setup

```
import { FormRow, Alert } from '../../components'
import { useAppContext } from '../../context/appContext'
import Wrapper from '../../assets/wrappers/DashboardFormPage'
const AddJob = () => {
  const {
    isEditing,
    showAlert,
    displayAlert,
    position,
    company,
    jobLocation,
    jobType,
    jobTypeOptions,
    status,
    statusOptions,
  } = useAppContext()

  const handleSubmit = (e) => {
    e.preventDefault()

    if (!position || !company || !jobLocation) {
      displayAlert()
      return
    }
    console.log('create job')
  }

  const handleJobInput = (e) => {
    const name = e.target.name
    const value = e.target.value
    console.log(`${name}:${value}`)
  }

  return (
    <Wrapper>
      <form className='form'>
        <h3>{isEditing ? 'edit job' : 'add job'} </h3>
        {showAlert && <Alert />}

        {/* position */}
        <div className='form-center'>
          <FormRow
            type='text'
            name='position'
            value={position}
            handleChange={handleJobInput}
          />
          {/* company */}
          <FormRow
            type='text'
```

```
            name='company'
            value={company}
            handleChange={handleJobInput}
          />
          {/* location */}
          <FormRow
            type='text'
            labelText='location'
            name='jobLocation'
            value={jobLocation}
            handleChange={handleJobInput}
          />
          {/* job type */}

          {/* job status */}

          <div className='btn-container'>
            <button
              className='btn btn-block submit-btn'
              type='submit'
              onClick={handleSubmit}
            >
              submit
            </button>
          </div>
        </div>
      </form>
    </Wrapper>
  )
}

export default AddJob
```

Select Input

```
return (
  // job type
  <div className='form-row'>
    <label htmlFor='jobType' className='form-label'>
      job type
    </label>

    <select
      name='jobType'
      value={jobType}
      onChange={handleJobInput}
      className='form-select'
    >
      {jobTypeOptions.map((itemValue, index) => {
        return (
          <option key={index} value={itemValue}>
            {itemValue}
          </option>
        )
      })}
    </select>
  </div>
)
```

FormRowSelect

- create FormRowSelect in components
- setup import/export

```jsx
const FormRowSelect = ({ labelText, name, value, handleChange, list }) => {
  return (
    <div className='form-row'>
      <label htmlFor={name} className='form-label'>
        {labelText || name}
      </label>

      <select
        name={name}
        value={value}
        onChange={handleChange}
        className='form-select'
      >
        {list.map((itemValue, index) => {
          return (
            <option key={index} value={itemValue}>
              {itemValue}
            </option>
          )
        })}
      </select>
    </div>
  )
}

export default FormRowSelect
```

AddJob.js

```
return (
  <>
    {/* job status */}

    <FormRowSelect
      name='status'
      value={status}
      handleChange={handleJobInput}
      list={statusOptions}
    />

    {/* job type */}
    <FormRowSelect
      labelText='type'
      name='jobType'
      value={jobType}
      handleChange={handleJobInput}
      list={jobTypeOptions}
    />
  </>
)
```

Change State Values With Handle Change

- JS Nuggets Dynamic Object Keys

actions.js

```
export const HANDLE_CHANGE = 'HANDLE_CHANGE'
```

appContext.js

```
const handleChange = ({ name, value }) => {
  dispatch({
    type: HANDLE_CHANGE,
    payload: { name, value },
  })
}

value={{handleChange}}
```

reducer.js

```
if (action.type === HANDLE_CHANGE) {
  return { ...state, [action.payload.name]: action.payload.value }
}
```

AddJob.js

```
const { handleChange } = useAppContext()

const handleJobInput = (e) => {
  handleChange({ name: e.target.name, value: e.target.value })
}
```

Clear Values

actions.js

```
export const CLEAR_VALUES = 'CLEAR_VALUES'
```

appContext.js

```
const clearValues = () => {
    dispatch({ type: CLEAR_VALUES })
  }

value={{clearValues}}
```

reducer.js

```
if (action.type === CLEAR_VALUES) {
  const initialState = {
    isEditing: false,
    editJobId: '',
    position: '',
    company: '',
    jobLocation: state.userLocation,
    jobType: 'full-time',
    status: 'pending',
  }
  return { ...state, ...initialState }
}
```

AddJob.js

```
const { clearValues } = useAppContext()

return (
  <div className='btn-container'>
    {/* submit button */}

    <button
      className='btn btn-block clear-btn'
      onClick={(e) => {
        e.preventDefault()
        clearValues()
      }}
    >
      clear
    </button>
  </div>
)
```

Create Job

actions.js

```
export const CREATE_JOB_BEGIN = 'CREATE_JOB_BEGIN'
export const CREATE_JOB_SUCCESS = 'CREATE_JOB_SUCCESS'
export const CREATE_JOB_ERROR = 'CREATE_JOB_ERROR'
```

appContext.js

```
const createJob = async () => {
  dispatch({ type: CREATE_JOB_BEGIN })
  try {
    const { position, company, jobLocation, jobType, status } = state

    await authFetch.post('/jobs', {
      company,
      position,
      jobLocation,
      jobType,
      status,
    })
    dispatch({
      type: CREATE_JOB_SUCCESS,
    })
    // call function instead clearValues()
    dispatch({ type: CLEAR_VALUES })
  } catch (error) {
    if (error.response.status === 401) return
    dispatch({
      type: CREATE_JOB_ERROR,
      payload: { msg: error.response.data.msg },
    })
  }
  clearAlert()
}
```

AddJob.js

```
const { createJob } = useAppContext()

const handleSubmit = (e) => {
  e.preventDefault()
  // while testing

  // if (!position || !company || !jobLocation) {
  //   displayAlert()
  //   return
  // }
  if (isEditing) {
    // eventually editJob()
    return
  }
  createJob()
}
```

reducer.js

```
if (action.type === CREATE_JOB_BEGIN) {
  return { ...state, isLoading: true }
}
if (action.type === CREATE_JOB_SUCCESS) {
  return {
    ...state,
    isLoading: false,
    showAlert: true,
    alertType: 'success',
    alertText: 'New Job Created!',
  }
}
if (action.type === CREATE_JOB_ERROR) {
  return {
    ...state,
    isLoading: false,
    showAlert: true,
    alertType: 'danger',
    alertText: action.payload.msg,
  }
}
```

Get All Jobs

jobsController.js

```
const getAllJobs = async (req, res) => {
  const jobs = await Job.find({ createdBy: req.user.userId })

  res
    .status(StatusCodes.OK)
    .json({ jobs, totalJobs: jobs.length, numOfPages: 1 })
}
```

Jobs State Values

appContext.js

```
const initialState = {
  jobs: [],
  totalJobs: 0,
  numOfPages: 1,
  page: 1,
}
```

Get All Jobs Request

```
actions.js
export const GET_JOBS_BEGIN = 'GET_JOBS_BEGIN'
export const GET_JOBS_SUCCESS = 'GET_JOBS_SUCCESS'


appContext.js

import React, { useReducer, useContext, useEffect } from 'react'

const getJobs = async () => {
  let url = `/jobs`

  dispatch({ type: GET_JOBS_BEGIN })
  try {
    const { data } = await authFetch(url)
    const { jobs, totalJobs, numOfPages } = data
    dispatch({
      type: GET_JOBS_SUCCESS,
      payload: {
        jobs,
        totalJobs,
        numOfPages,
      },
    })
  } catch (error) {
    console.log(error.response)
    logoutUser()
  }
  clearAlert()
}

useEffect(() => {
  getJobs()
}, [])

value={{getJobs}}
```

reducer.js

```
if (action.type === GET_JOBS_BEGIN) {
  return { ...state, isLoading: true, showAlert: false }
}
if (action.type === GET_JOBS_SUCCESS) {
  return {
    ...state,
    isLoading: false,
    jobs: action.payload.jobs,
    totalJobs: action.payload.totalJobs,
    numOfPages: action.payload.numOfPages,
  }
}
```

AllJobs Page Setup

- create
- SearchContainer export
- JobsContainer export
- Job
- JobInfo

AllJobs.js

```
import { JobsContainer, SearchContainer } from '../../components'
const AllJobs = () => {
  return (
    <>
      <SearchContainer />
      <JobsContainer />
    </>
  )
}

export default AllJobs
```

JobsContainer.js

```
import { useAppContext } from '../context/appContext'
import { useEffect } from 'react'
import Loading from './Loading'
import Job from './Job'
import Wrapper from '../assets/wrappers/JobsContainer'

const JobsContainer = () => {
  const { getJobs, jobs, isLoading, page, totalJobs } = useAppContext()
  useEffect(() => {
    getJobs()
  }, [])

  if (isLoading) {
    return <Loading center />
  }
  if (jobs.length === 0) {
    return (
      <Wrapper>
        <h2>No jobs to display...</h2>
      </Wrapper>
    )
  }
  return (
    <Wrapper>
      <h5>
        {totalJobs} job{jobs.length > 1 && 's'} found
      </h5>
      <div className='jobs'>
        {jobs.map((job) => {
          return <Job key={job._id} {...job} />
        })}
      </div>
    </Wrapper>
  )
}

export default JobsContainer
```

Job.js

```
import moment from 'moment'

const Job = ({ company }) => {
  return <h5>{company}</h5>
}

export default Job
```

Moment.js

- Format Dates
- moment.js
- stop server
- cd client

```
npm install moment
```

Job.js

```
import moment from 'moment'

const Job = ({ company, createdAt }) => {
  let date = moment(createdAt)
  date = date.format('MMM Do, YYYY')
  return (
    <div>
      <h5>{company}</h5>
      <h5>{date}</h5>
    </div>
  )
}

export default Job
```

Job Component - Setup

appContext.js

```
const setEditJob = (id) => {
  console.log(`set edit job : ${id}`)
}
const deleteJob = (id) =>{
  console.log(`delete : ${id}`)
}
value={{setEditJob,deleteJob}}
```

Job.js

```
import { FaLocationArrow, FaBriefcase, FaCalendarAlt } from 'react-icons/fa'
import { Link } from 'react-router-dom'
import { useAppContext } from '../context/appContext'
import Wrapper from '../assets/wrappers/Job'
import JobInfo from './JobInfo'

const Job = ({
  _id,
  position,
  company,
  jobLocation,
  jobType,
  createdAt,
  status,
}) => {
  const { setEditJob, deleteJob } = useAppContext()

  let date = moment(createdAt)
  date = date.format('MMM Do, YYYY')

  return (
    <Wrapper>
      <header>
        <div className='main-icon'>{company.charAt(0)}</div>
        <div className='info'>
          <h5>{position}</h5>
          <p>{company}</p>
        </div>
      </header>
      <div className='content'>
        {/* content center later */}
        <footer>
          <div className='actions'>
            <Link
              to='/add-job'
              onClick={() => setEditJob(_id)}
              className='btn edit-btn'
            >
              Edit
            </Link>
            <button
              type='button'
              className='btn delete-btn'
              onClick={() => deleteJob(_id)}
            >
              Delete
            </button>
          </div>
```

```
        </footer>
      </div>
    </Wrapper>
  )
}

export default Job
```

JobInfo

```
JobInfo.js

import Wrapper from '../assets/wrappers/JobInfo'

const JobInfo = ({ icon, text }) => {
  return (
    <Wrapper>
      <span className='icon'>{icon}</span>
      <span className='text'>{text}</span>
    </Wrapper>
  )
}

export default JobInfo


Job.js
return (
  <div className='content'>
    <div className='content-center'>
      <JobInfo icon={<FaLocationArrow />} text={jobLocation} />
      <JobInfo icon={<FaCalendarAlt />} text={date} />
      <JobInfo icon={<FaBriefcase />} text={jobType} />
      <div className={`status ${status}`}>{status}</div>
    </div>
    {/* footer content */}
  </div>
)
```

SetEditJob

```
actions.js
export const SET_EDIT_JOB = 'SET_EDIT_JOB'
```

appContext.js

```
const setEditJob = (id) => {
  dispatch({ type: SET_EDIT_JOB, payload: { id } })
}
const editJob = () => {
  console.log('edit job')
}
value={{editJob}}
```

reducer.js

```
if (action.type === SET_EDIT_JOB) {
  const job = state.jobs.find((job) => job._id === action.payload.id)
  const { _id, position, company, jobLocation, jobType, status } = job
  return {
    ...state,
    isEditing: true,
    editJobId: _id,
    position,
    company,
    jobLocation,
    jobType,
    status,
  }
}
```

AddJob.js

```
const { isEditing, editJob } = useAppContext()
const handleSubmit = (e) => {
  e.preventDefault()

  if (!position || !company || !jobLocation) {
    displayAlert()
    return
  }
  if (isEditing) {
    editJob()
    return
  }
  createJob()
}
```

Edit Job - Server

jobsController.js

```
const updateJob = async (req, res) => {
  const { id: jobId } = req.params

  const { company, position } = req.body

  if (!company || !position) {
    throw new BadRequestError('Please Provide All Values')
  }

  const job = await Job.findOne({ _id: jobId })

  if (!job) {
    throw new NotFoundError(`No job with id ${jobId}`)
  }

  // check permissions

  const updatedJob = await Job.findOneAndUpdate({ _id: jobId }, req.body, {
    new: true,
    runValidators: true,
  })

  res.status(StatusCodes.OK).json({ updatedJob })
}
```

Alternative Approach

- optional
- multiple approaches
- different setups
- course Q&A

jobsController.js

```
const updateJob = async (req, res) => {
  const { id: jobId } = req.params
  const { company, position, jobLocation } = req.body

  if (!position || !company) {
    throw new BadRequestError('Please provide all values')
  }
  const job = await Job.findOne({ _id: jobId })

  if (!job) {
    throw new NotFoundError(`No job with id :${jobId}`)
  }

  // check permissions

  // alternative approach

  job.position = position
  job.company = company
  job.jobLocation = jobLocation

  await job.save()
  res.status(StatusCodes.OK).json({ job })
}
```

Check Permissions

jobsController.js

```
const updateJob = async (req, res) => {
  const { id: jobId } = req.params
  const { company, position, status } = req.body

  if (!position || !company) {
    throw new BadRequestError('Please provide all values')
  }
  const job = await Job.findOne({ _id: jobId })

  if (!job) {
    throw new NotFoundError(`No job with id :${jobId}`)
  }

  // check permissions
  // req.user.userId (string) === job.createdBy(object)
  // throw new UnAuthorizedError('Not authorized to access this route')

  // console.log(typeof req.user.userId)
  // console.log(typeof job.createdBy)

  checkPermissions(req.user, job.createdBy)

  const updatedJob = await Job.findOneAndUpdate({ _id: jobId }, req.body, {
    new: true,
    runValidators: true,
  })

  res.status(StatusCodes.OK).json({ updatedJob })
}
```

- utils folder
- checkPermissions.js
- import in jobsController.js

checkPermissions.js

```
import { UnAuthorizedError } from '../errors/index.js'

const checkPermissions = (requestUser, resourceUserId) => {
  // if (requestUser.role === 'admin') return
  if (requestUser.userId === resourceUserId.toString()) return
  throw new CustomError.UnauthorizedError('Not authorized to access this route')
}

export default checkPermissions
```

Remove/Delete Job

jobsController.js

```
const deleteJob = async (req, res) => {
  const { id: jobId } = req.params

  const job = await Job.findOne({ _id: jobId })

  if (!job) {
    throw new CustomError.NotFoundError(`No job with id : ${jobId}`)
  }

  checkPermissions(req.user, job.createdBy)

  await job.remove()
  res.status(StatusCodes.OK).json({ msg: 'Success! Job removed' })
}
```

Delete Job - Front-End

actions.js

```
export const DELETE_JOB_BEGIN = 'DELETE_JOB_BEGIN'
```

appContext.js

```
const deleteJob = async (jobId) => {
  dispatch({ type: DELETE_JOB_BEGIN })
  try {
    await authFetch.delete(`/jobs/${jobId}`)
    getJobs()
  } catch (error) {
    logoutUser()
  }
}
```

reducer.js

```
if (action.type === DELETE_JOB_BEGIN) {
  return { ...state, isLoading: true }
}
```

Edit Job - Front-End

actions.js

```
export const EDIT_JOB_BEGIN = 'EDIT_JOB_BEGIN'
export const EDIT_JOB_SUCCESS = 'EDIT_JOB_SUCCESS'
export const EDIT_JOB_ERROR = 'EDIT_JOB_ERROR'
```

appContext.js

```
const editJob = async () => {
  dispatch({ type: EDIT_JOB_BEGIN })
  try {
    const { position, company, jobLocation, jobType, status } = state

    await authFetch.patch(`/jobs/${state.editJobId}`, {
      company,
      position,
      jobLocation,
      jobType,
      status,
    })
    dispatch({
      type: EDIT_JOB_SUCCESS,
    })
    dispatch({ type: CLEAR_VALUES })
  } catch (error) {
    if (error.response.status === 401) return
    dispatch({
      type: EDIT_JOB_ERROR,
      payload: { msg: error.response.data.msg },
    })
  }
  clearAlert()
}
```

reducer.js

```
if (action.type === EDIT_JOB_BEGIN) {
  return { ...state, isLoading: true }
}
if (action.type === EDIT_JOB_SUCCESS) {
  return {
    ...state,
    isLoading: false,
    showAlert: true,
    alertType: 'success',
    alertText: 'Job Updated!',
  }
}
if (action.type === EDIT_JOB_ERROR) {
  return {
    ...state,
    isLoading: false,
    showAlert: true,
    alertType: 'danger',
    alertText: action.payload.msg,
  }
}
```

Create More Jobs

- Mockaroo
- setup mock-data.json in the root

Populate Database

- create populate.js in the root

populate.js

```
import { readFile } from 'fs/promises'

import dotenv from 'dotenv'
dotenv.config()

import connectDB from './db/connect.js'
import Job from './models/Job.js'

const start = async () => {
  try {
    await connectDB(process.env.MONGO_URL)
    await Job.deleteMany()

    const jsonProducts = JSON.parse(
      await readFile(new URL('./mock-data.json', import.meta.url))
    )
    await Job.create(jsonProducts)
    console.log('Success!!!!')
    process.exit(0)
  } catch (error) {
    console.log(error)
    process.exit(1)
  }
}

start()
```

Show Stats - Structure

- aggregation pipeline
- step by step
- Aggregation Pipeline

jobsController.js

```
import mongoose from 'mongoose'

const showStats = async (req, res) => {
  let stats = await Job.aggregate([
    { $match: { createdBy: mongoose.Types.ObjectId(req.user.userId) } },
    { $group: { _id: '$status', count: { $sum: 1 } } },
  ])

  res.status(StatusCodes.OK).json({ stats })
}
```

Show Stats - Object Setup

- Reduce Basics
- Reduce Object Example

jobsController.js

```
const showStats = async (req, res) => {
  let stats = await Job.aggregate([
    { $match: { createdBy: mongoose.Types.ObjectId(req.user.userId) } },
    { $group: { _id: '$status', count: { $sum: 1 } } },
  ])

  stats = stats.reduce((acc, curr) => {
    const { _id: title, count } = curr
    acc[title] = count
    return acc
  }, {})

  res.status(StatusCodes.OK).json({ stats })
}
```

Show Stats - Default Stats

jobsController.js

```
const showStats = async (req, res) => {
  let stats = await Job.aggregate([
    { $match: { createdBy: mongoose.Types.ObjectId(req.user.userId) } },
    { $group: { _id: '$status', count: { $sum: 1 } } },
  ])
  stats = stats.reduce((acc, curr) => {
    const { _id: title, count } = curr
    acc[title] = count
    return acc
  }, {})

  const defaultStats = {
    pending: stats.pending || 0,
    interview: stats.interview || 0,
    declined: stats.declined || 0,
  }
  let monthlyApplications = []
  res.status(StatusCodes.OK).json({ defaultStats, monthlyApplications })
}
```

Show Stats - Function Setup

actions.js

```
export const SHOW_STATS_BEGIN = 'SHOW_STATS_BEGIN'
export const SHOW_STATS_SUCCESS = 'SHOW_STATS_SUCCESS'
```

appContext.js

```
const initialState = {
  stats: {},
  monthlyApplications: []

}

const showStats = async () => {
    dispatch({ type: SHOW_STATS_BEGIN })
    try {
      const { data } = await authFetch('/jobs/stats')
      dispatch({
        type: SHOW_STATS_SUCCESS,
        payload: {
          stats: data.defaultStats,
          monthlyApplications: data.monthlyApplications,
        },
      })
    } catch (error) {
console.log(error.response)
      // logoutUser()
    }

clearAlert()
  }
  value={{showStats}}
```

reducers.js

```
if (action.type === SHOW_STATS_BEGIN) {
  return { ...state, isLoading: true, showAlert: false }
}
if (action.type === SHOW_STATS_SUCCESS) {
  return {
    ...state,
    isLoading: false,
    stats: action.payload.stats,
    monthlyApplications: action.payload.monthlyApplications,
  }
}
```

Stats Page - Structure

- components
- StatsContainer.js
- ChartsContainer.js
- StatsItem.js
- simple return
- import/export index.js

Stats.js

```
import { useEffect } from 'react'
import { useAppContext } from '../../context/appContext'
import { StatsContainer, Loading, ChartsContainer } from '../../components'

const Stats = () => {
  const { showStats, isLoading, monthlyApplications } = useAppContext()
  useEffect(() => {
    showStats()
  }, [])

  if (isLoading) {
    return <Loading center />
  }

  return (
    <>
      <StatsContainer />
      {monthlyApplications.length > 0 && <ChartsContainer />}
    </>
  )
}

export default Stats
```

StatsContainer

StatsContainer.js

```
import { useAppContext } from '../context/appContext'
import StatItem from './StatItem'
import { FaSuitcaseRolling, FaCalendarCheck, FaBug } from 'react-icons/fa'
import Wrapper from '../assets/wrappers/StatsContainer'
const StatsContainer = () => {
  const { stats } = useAppContext()
  const defaultStats = [
    {
      title: 'pending applications',
      count: stats.pending || 0,
      icon: <FaSuitcaseRolling />,
      color: '#e9b949',
      bcg: '#fcefc7',
    },
    {
      title: 'interviews scheduled',
      count: stats.interview || 0,
      icon: <FaCalendarCheck />,
      color: '#647acb',
      bcg: '#e0e8f9',
    },
    {
      title: 'jobs declined',
      count: stats.declined || 0,
      icon: <FaBug />,
      color: '#d66a6a',
      bcg: '#ffeeee',
    },
  ]

  return (
    <Wrapper>
      {defaultStats.map((item, index) => {
        return <StatItem key={index} {...item} />
      })}
    </Wrapper>
  )
}

export default StatsContainer
```

StatItem

StatItem.js

```
import Wrapper from '../assets/wrappers/StatItem'

function StatItem({ count, title, icon, color, bcg }) {
  return (
    <Wrapper color={color} bcg={bcg}>
      <header>
        <span className='count'>{count}</span>
        <div className='icon'>{icon}</div>
      </header>
      <h5 className='title'>{title}</h5>
    </Wrapper>
  )
}

export default StatItem
```

Aggregate Jobs Based on Year and Month

jobsController.js

```
let monthlyApplications = await Job.aggregate([
  { $match: { createdBy: mongoose.Types.ObjectId(req.user.userId) } },
  {
    $group: {
      _id: {
        year: {
          $year: '$createdAt',
        },
        month: {
          $month: '$createdAt',
        },
      },
      count: { $sum: 1 },
    },
  },
  { $sort: { '_id.year': -1, '_id.month': -1 } },
  { $limit: 6 },
])
```

Refactor Data

- install moment.js on the SERVER

```
npm install moment
```

jobsController.js

```
import moment from 'moment'

monthlyApplications = monthlyApplications
  .map((item) => {
    const {
      _id: { year, month },
      count,
    } = item
    // accepts 0-11
    const date = moment()
      .month(month - 1)
      .year(year)
      .format('MMM Y')
    return { date, count }
  })
  .reverse()
```

Charts Container

- BarChart.js
- AreaChart.js

ChartsContainer.js

```
import React, { useState } from 'react'

import BarChart from './BarChart'
import AreaChart from './AreaChart'
import { useAppContext } from '../context/appContext'
import Wrapper from '../assets/wrappers/ChartsContainer'

export default function ChartsContainer() {
  const [barChart, setBarChart] = useState(true)
  const { monthlyApplications: data } = useAppContext()

  return (
    <Wrapper>
      <h4>Monthly Applications</h4>

      <button type='button' onClick={() => setBarChart(!barChart)}>
        {barChart ? 'AreaChart' : 'BarChart'}
      </button>
      {barChart ? <BarChart data={data} /> : <AreaChart data={data} />}
    </Wrapper>
  )
}
```

Recharts Library

- install in the Client!!!

Recharts

```
npm install recharts
```

Bar Chart

BarChart.js

```
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  ResponsiveContainer,
} from 'recharts'

const BarChartComponent = ({ data }) => {
  return (
    <ResponsiveContainer width='100%' height={300}>
      <BarChart
        data={data}
        margin={{
          top: 50,
        }}
      >
        <CartesianGrid strokeDasharray='3 3' />
        <XAxis dataKey='date' />
        <YAxis allowDecimals={false} />
        <Tooltip />
        <Bar dataKey='count' fill='#2cb1bc' barSize={75} />
      </BarChart>
    </ResponsiveContainer>
  )
}
```

Area Chart

```jsx
import {
  ResponsiveContainer,
  AreaChart,
  Area,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
} from 'recharts'

const AreaChartComponent = ({ data }) => {
  return (
    <ResponsiveContainer width='100%' height={300}>
      <AreaChart
        data={data}
        margin={{
          top: 50,
        }}
      >
        <CartesianGrid strokeDasharray='3 3' />
        <XAxis dataKey='date' />
        <YAxis allowDecimals={false} />
        <Tooltip />
        <Area type='monotone' dataKey='count' stroke='#2cb1bc' fill='#bef8fd' />
      </AreaChart>
    </ResponsiveContainer>
  )
}
```

Filter

Get All Jobs - Initial Setup

jobsController.js

```
const getAllJobs = async (req, res) => {
  const { search, status, jobType, sort } = req.query

  const queryObject = {
    createdBy: req.user.userId,
  }

  // NO AWAIT
  let result = Job.find(queryObject)

  // chain sort conditions

  const jobs = await result

  res
    .status(StatusCodes.OK)
    .json({ jobs, totalJobs: jobs.length, numOfPages: 1 })
}
```

Status

jobsController.js

```
const getAllJobs = async (req, res) => {
  const { search, status, jobType, sort } = req.query

  const queryObject = {
    createdBy: req.user.userId,
  }

  if (status !== 'all') {
    queryObject.status = status
  }

  // NO AWAIT
  let result = Job.find(queryObject)

  // chain sort conditions

  const jobs = await result

  res
    .status(StatusCodes.OK)
    .json({ jobs, totalJobs: jobs.length, numOfPages: 1 })
}
```

JobType

jobsController.js

```
const getAllJobs = async (req, res) => {
  const { search, status, jobType, sort } = req.query

  const queryObject = {
    createdBy: req.user.userId,
  }

  if (status !== 'all') {
    queryObject.status = status
  }
  if (jobType !== 'all') {
    queryObject.jobType = jobType
  }
  // NO AWAIT
  let result = Job.find(queryObject)

  // chain sort conditions

  const jobs = await result

  res
    .status(StatusCodes.OK)
    .json({ jobs, totalJobs: jobs.length, numOfPages: 1 })
}
```

Search

jobsController.js

```
const getAllJobs = async (req, res) => {
  const { search, status, jobType, sort } = req.query

  const queryObject = {
    createdBy: req.user.userId,
  }

  if (status !== 'all') {
    queryObject.status = status
  }
  if (jobType !== 'all') {
    queryObject.jobType = jobType
  }
  if (search) {
    queryObject.position = { $regex: search, $options: 'i' }
  }
  // NO AWAIT
  let result = Job.find(queryObject)

  // chain sort conditions
  if (sort === 'latest') {
    result = result.sort('-createdAt')
  }
  if (sort === 'oldest') {
    result = result.sort('createdAt')
  }
  if (sort === 'a-z') {
    result = result.sort('position')
  }
  if (sort === 'z-a') {
    result = result.sort('-position')
  }
  const jobs = await result

  res
    .status(StatusCodes.OK)
    .json({ jobs, totalJobs: jobs.length, numOfPages: 1 })
}
```

Search Context Setup

appContext.js

```
const initialState = {
  jobType: 'full-time',
  jobTypeOptions: ['full-time', 'part-time', 'remote', 'internship'],
  status: 'pending',
  statusOptions: ['pending', 'interview', 'declined']
  //
  //
  //
  search: '',
  searchStatus: 'all',
  searchType: 'all',
  sort: 'latest',
  sortOptions: ['latest', 'oldest', 'a-z', 'z-a'],
}

const clearFilters = () =>{
console.log('clear filters')
}

value={{clearFilters}}

// remember this function :)
const handleChange = ({ name, value }) => {
    dispatch({
      type: HANDLE_CHANGE,
      payload: { name, value },
    })
  }
```

Search Container - Setup

SearchContainer.js

```
import { FormRow, FormRowSelect } from '.'
import { useAppContext } from '../context/appContext'
import Wrapper from '../assets/wrappers/SearchContainer'
const SearchContainer = () => {
  const {
    isLoading,
    search,
    searchStatus,
    searchType,
    sort,
    sortOptions,
    statusOptions,
    jobTypeOptions,
    handleChange,
    clearFilters,
  } = useAppContext()

  const handleSearch = (e) => {
    if (isLoading) return
    handleChange({ name: e.target.name, value: e.target.value })
  }

  return (
    <Wrapper>
      <form className='form'>
        <h4>search form</h4>
        {/* search position */}
        <div className='form-center'>
          <FormRow
            type='text'
            name='search'
            value={search}
            handleChange={handleSearch}
          ></FormRow>
          {/* rest of the inputs */}
        </div>
      </form>
    </Wrapper>
  )
}

export default SearchContainer
```

Search Container - Complete

SearchContainer.js

```
import { FormRow, FormRowSelect } from '.'
import { useAppContext } from '../context/appContext'
import Wrapper from '../assets/wrappers/SearchContainer'

const SearchContainer = () => {
  const {
    isLoading,
    search,
    handleChange,
    searchStatus,
    statusOptions,
    jobTypeOptions,
    searchType,
    clearFilters,
    sort,
    sortOptions,
  } = useAppContext()

  const handleSearch = (e) => {
    if (isLoading) return
    handleChange({ name: e.target.name, value: e.target.value })
  }
  const handleSubmit = (e) => {
    e.preventDefault()
    clearFilters()
  }
  return (
    <Wrapper>
      <form className='form'>
        <h4>search form</h4>
        {/* search position */}
        <div className='form-center'>
          <FormRow
            type='text'
            name='search'
            value={search}
            handleChange={handleSearch}
          ></FormRow>
          {/* search by status */}
          <FormRowSelect
            labelText='job status'
            name='searchStatus'
            value={searchStatus}
            handleChange={handleSearch}
            list={['all', ...statusOptions]}
          ></FormRowSelect>
          {/* search by type */}
```

```
          <FormRowSelect
            labelText='job type'
            name='searchType'
            value={searchType}
            handleChange={handleSearch}
            list={['all', ...jobTypeOptions]}
          ></FormRowSelect>
          {/* sort */}

          <FormRowSelect
            name='sort'
            value={sort}
            handleChange={handleSearch}
            list={sortOptions}
          ></FormRowSelect>
          <button
            className='btn btn-block btn-danger'
            disabled={isLoading}
            onClick={handleSubmit}
          >
            clear filters
          </button>
        </div>
      </form>
    </Wrapper>
  )
}

export default SearchContainer
```

Clear Filters

actions.js

```
export const CLEAR_FILTERS = 'CLEAR_FILTERS'
```

appContext.js

```
const clearFilters = () => {
  dispatch({ type: CLEAR_FILTERS })
}
```

reducer.js

```
if (action.type === CLEAR_FILTERS) {
  return {
    ...state,
    search: '',
    searchStatus: 'all',
    searchType: 'all',
    sort: 'latest',
  }
}
```

Refactor Get All Jobs

```
const getJobs = async () => {
  // will add page later
  const { search, searchStatus, searchType, sort } = state
  let url = `/jobs?status=${searchStatus}&jobType=${searchType}&sort=${sort}`
  if (search) {
    url = url + `&search=${search}`
  }
  dispatch({ type: GET_JOBS_BEGIN })
  try {
    const { data } = await authFetch(url)
    const { jobs, totalJobs, numOfPages } = data
    dispatch({
      type: GET_JOBS_SUCCESS,
      payload: {
        jobs,
        totalJobs,
        numOfPages,
      },
    })
  } catch (error) {
    // logoutUser()
  }
  clearAlert()
}
```

JobsContainer.js

```
const JobsContainer = () => {
  const {
    getJobs,
    jobs,
    isLoading,
    page,
    totalJobs,
    search,
    searchStatus,
    searchType,
    sort,

  } = useAppContext()
  useEffect(() => {
    getJobs()
  }, [ search, searchStatus, searchType, sort])
```

Limit and Skip

jobsController.js

```
const getAllJobs = async (req, res) => {
  const { search, status, jobType, sort } = req.query
  const queryObject = {
    createdBy: req.user.userId,
  }
  if (search) {
    queryObject.position = { $regex: search, $options: 'i' }
  }
  if (status !== 'all') {
    queryObject.status = status
  }
  if (jobType !== 'all') {
    queryObject.jobType = jobType
  }
  let result = Job.find(queryObject)

  if (sort === 'latest') {
    result = result.sort('-createdAt')
  }
  if (sort === 'oldest') {
    result = result.sort('createdAt')
  }
  if (sort === 'a-z') {
    result = result.sort('position')
  }
  if (sort === 'z-a') {
    result = result.sort('-position')
  }

  const totalJobs = await result

  // setup pagination
  const limit = 10
  const skip = 1

  result = result.skip(skip).limit(limit)
  // 23
  // 4 7 7 7 2
  const jobs = await result
  res
    .status(StatusCodes.OK)
    .json({ jobs, totalJobs: jobs.length, numOfPages: 1 })
}
```

Page and Limit

jobsController.js

```
const getAllJobs = async (req, res) => {
  const { search, status, jobType, sort } = req.query
  const queryObject = {
    createdBy: req.user.userId,
  }
  if (search) {
    queryObject.position = { $regex: search, $options: 'i' }
  }
  if (status !== 'all') {
    queryObject.status = status
  }
  if (jobType !== 'all') {
    queryObject.jobType = jobType
  }
  let result = Job.find(queryObject)

  if (sort === 'latest') {
    result = result.sort('-createdAt')
  }
  if (sort === 'oldest') {
    result = result.sort('createdAt')
  }
  if (sort === 'a-z') {
    result = result.sort('position')
  }
  if (sort === 'z-a') {
    result = result.sort('-position')
  }

  // setup pagination
  const page = Number(req.query.page) || 1
  const limit = Number(req.query.limit) || 10
  const skip = (page - 1) * limit //10
  result = result.skip(skip).limit(limit)
  // 75
  // 10 10 10 10 10 10 10 5
  const jobs = await result
  res
    .status(StatusCodes.OK)
    .json({ jobs, totalJobs: jobs.length, numOfPages: 1 })
}
```

Total Jobs and Number Of Pages

jobsController.js

```
const getAllJobs = async (req, res) => {
  const { search, status, jobType, sort } = req.query
  const queryObject = {
    createdBy: req.user.userId,
  }
  if (search) {
    queryObject.position = { $regex: search, $options: 'i' }
  }
  if (status !== 'all') {
    queryObject.status = status
  }
  if (jobType !== 'all') {
    queryObject.jobType = jobType
  }
  let result = Job.find(queryObject)

  if (sort === 'latest') {
    result = result.sort('-createdAt')
  }
  if (sort === 'oldest') {
    result = result.sort('createdAt')
  }
  if (sort === 'a-z') {
    result = result.sort('position')
  }
  if (sort === 'z-a') {
    result = result.sort('-position')
  }

  // setup pagination
  const page = Number(req.query.page) || 1
  const limit = Number(req.query.limit) || 10
  const skip = (page - 1) * limit

  result = result.skip(skip).limit(limit)

  const jobs = await result

  const totalJobs = await Job.countDocuments(queryObject)
  const numOfPages = Math.ceil(totalJobs / limit)

  res.status(StatusCodes.OK).json({ jobs, totalJobs, numOfPages })
}
```

PageBtnContainer Setup

- PageBtnContainer.js

JobsContainer.js

```
import PageBtnContainer from './PageBtnContainer'

const { numOfPages } = useAppContext()

return (
  <Wrapper>
    <h5>
      {totalJobs} job{jobs.length > 1 && 's'} found
    </h5>
    <div className='jobs'>
      {jobs.map((job) => {
        return <Job key={job._id} {...job} />
      })}
    </div>
    {numOfPages > 1 && <PageBtnContainer />}
  </Wrapper>
)
```

PageBtnContainer - Structure

PageBtnContainer.js

```
import { useAppContext } from '../context/appContext'
import { HiChevronDoubleLeft, HiChevronDoubleRight } from 'react-icons/hi'
import Wrapper from '../assets/wrappers/PageBtnContainer'

const PageButtonContainer = () => {
  const { numOfPages, page } = useAppContext()

  const prevPage = () => {
    console.log('prev page')
  }
  const nextPage = () => {
    console.log('next page')
  }

  return (
    <Wrapper>
      <button className='prev-btn' onClick={prevPage}>
        <HiChevronDoubleLeft />
        prev
      </button>

      <div className='btn-container'>buttons</div>

      <button className='next-btn' onClick={nextPage}>
        next
        <HiChevronDoubleRight />
      </button>
    </Wrapper>
  )
}

export default PageButtonContainer
```

Button Container

- [Array.from] (https://youtu.be/zg1Bv4xubwo)

PageBtnContainer.js

```
const pages = Array.from({ length: numOfPages }, (_, index) => {
  return index + 1
})

return (
  <div className='btn-container'>
    {pages.map((pageNumber) => {
      return (
        <button
          type='button'
          className={pageNumber === page ? 'pageBtn active' : 'pageBtn'}
          key={pageNumber}
          onClick={() => console.log(page)}
        >
          {pageNumber}
        </button>
      )
    })}
  </div>
)
```

Change Page

actions.js

```
export const CHANGE_PAGE = 'CHANGE_PAGE'
```

appContext.js

```
const changePage = (page) => {
  dispatch({ type: CHANGE_PAGE, payload: { page } })
}
value={{changePage}}
```

reducer.js

```
if (action.type === CHANGE_PAGE) {
  return { ...state, page: action.payload.page }
}
```

PageBtnContainer.js

```
const { changePage } = useAppContext()
return (
  <button
    type='button'
    className={pageNumber === page ? 'pageBtn active' : 'pageBtn'}
    key={pageNumber}
    onClick={() => changePage(pageNumber)}
  >
    {pageNumber}
  </button>
)
```

Prev and Next Buttons

PageBtnContainer.js

```
const prevPage = () => {
  let newPage = page - 1
  if (newPage < 1) {
    // newPage = 1
    // alternative
    newPage = numOfPages
  }
  changePage(newPage)
}
const nextPage = () => {
  let newPage = page + 1
  if (newPage > numOfPages) {
    // newPage = numOfPages
    // alternative
    newPage = 1
  }
  changePage(newPage)
}
```

Trigger New Page

appContext.js

```
const getJobs = async () => {
  const { page, search, searchStatus, searchType, sort } = state

  let url = `/jobs?page=${page}&status=${searchStatus}&jobType=${searchType}&sort=${sort}`
  // rest of the code
}
```

JobsContainer.js

```
const { page } = useAppContext()
useEffect(() => {
  getJobs()
}, [page, search, searchStatus, searchType, sort])
```

reducer.js

```
if (action.type === HANDLE_CHANGE) {
  // set back to first page

  return { ...state, page: 1, [action.payload.name]: action.payload.value }
}
```

Production Setup - Fix Warnings and logoutUser

- getJobs,deleteJob,showStats - invoke logoutUser()
- fix warnings

```
// eslint-disable-next-line
```

Production Setup - Build Front-End Application

- create front-end production application

package.json

```
"scripts": {
    "build-client": "cd client && npm run build",
    "server": "nodemon server.js --ignore client",
    "client": "cd client && npm run start",
    "start": "concurrently --kill-others-on-fail \"npm run server\" \"npm run client\""

  },
```

server.js

```
import { dirname } from 'path'
import { fileURLToPath } from 'url'
import path from 'path'

const __dirname = dirname(fileURLToPath(import.meta.url))

// only when ready to deploy
app.use(express.static(path.resolve(__dirname, './client/build')))

// routes
app.use('/api/v1/auth', authRouter)
app.use('/api/v1/jobs', authenticateUser, jobsRouter)

// only when ready to deploy
app.get('*', function (request, response) {
  response.sendFile(path.resolve(__dirname, './client/build', 'index.html'))
})
```

Security Packages

- remove log in the error-handler
- helmet

  Helmet helps you secure your Express apps by setting various HTTP headers.
- xss-clean

  Node.js Connect middleware to sanitize user input coming from POST body, GET queries, and url params.
- express-mongo-sanitize

  Sanitizes user-supplied data to prevent MongoDB Operator Injection.
- express-rate-limit

  Basic rate-limiting middleware for Express.

```
npm install helmet xss-clean express-mongo-sanitize express-rate-limit
```

server.js

```
import helmet from 'helmet'
import xss from 'xss-clean'
import mongoSanitize from 'express-mongo-sanitize'

app.use(express.json())
app.use(helmet())
app.use(xss())
app.use(mongoSanitize())
```

Limit Requests

authRoutes.js

```
import rateLimiter from 'express-rate-limit'

const apiLimiter = rateLimiter({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 10,
  message: 'Too many requests from this IP, please try again after 15 minutes',
})

router.route('/register').post(apiLimiter, register)
router.route('/login').post(apiLimiter, login)
```

Deploy To Heroku

- heroku login

package.json

```
"engines": {
    "node": "16.x"
  }
"scripts":{
    "build-client": "cd client && npm run build",
    "install-client": "cd client && npm install",
    "heroku-postbuild": "npm run install-client && npm run build-client",
}
```

Procfile

```
web: node server.js
```

- rm -rf .git

- git init
- git add .
- git commit -m "first commit"
- heroku create nameOfTheApp
- git remote -v
- add env variables
- git push heroku main/master