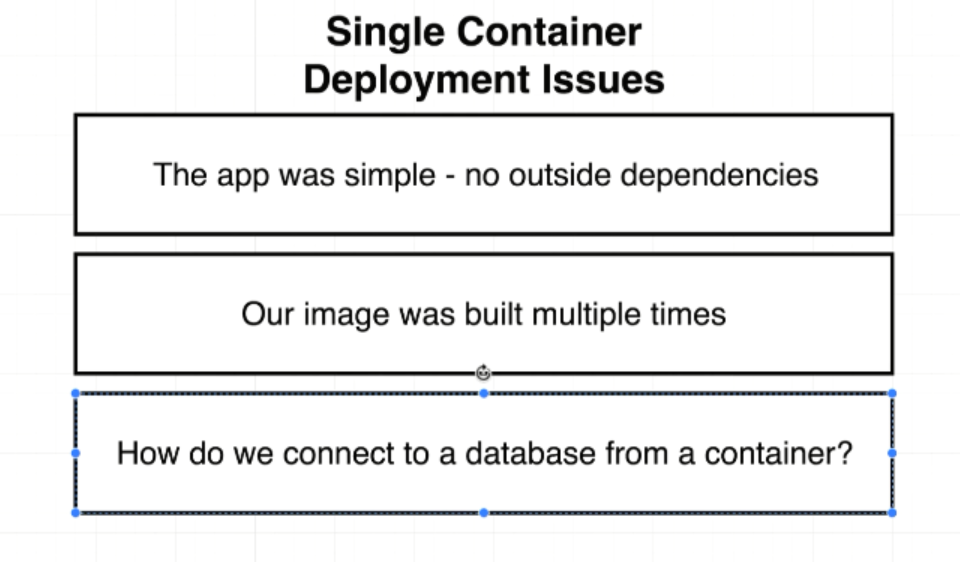
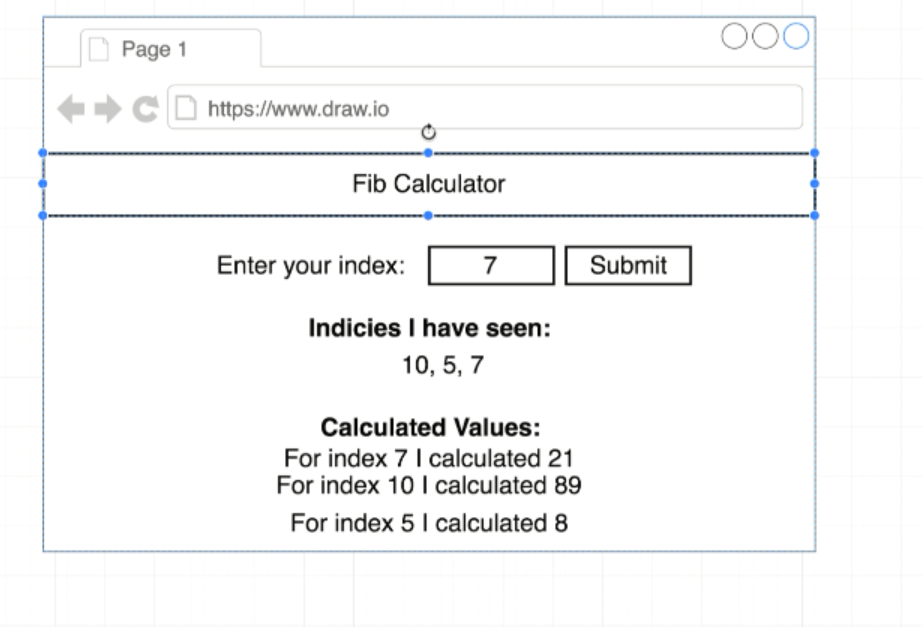
V—113:(Single Container Deployment Issues):

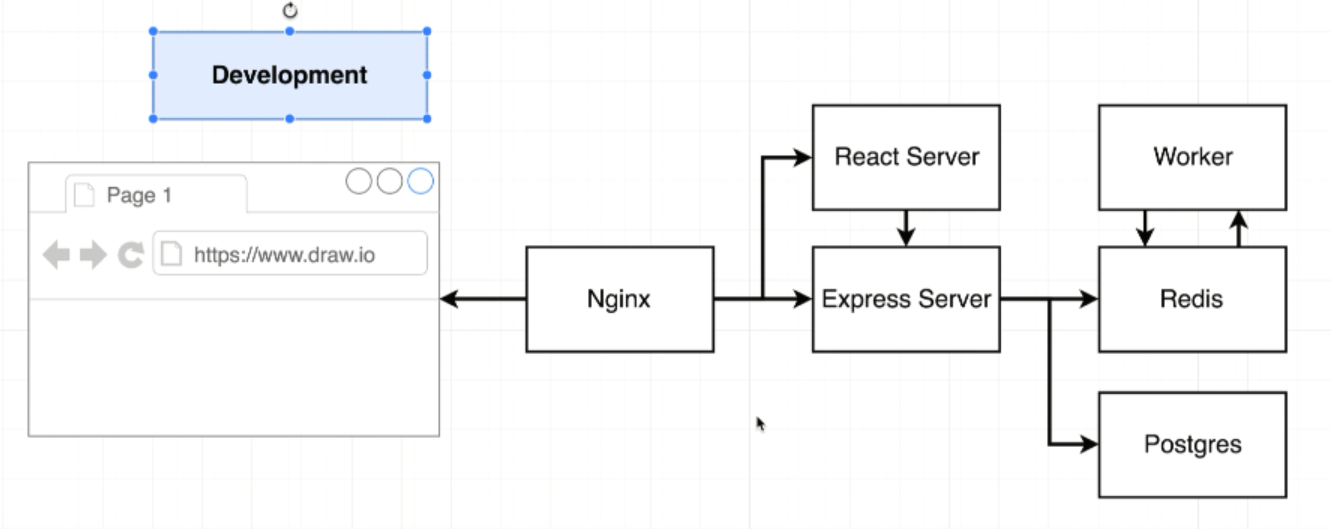


V—114:(Application Overview):



V—116:(Application Architecture):

We will see how to work with multiple components/containers and fit then together



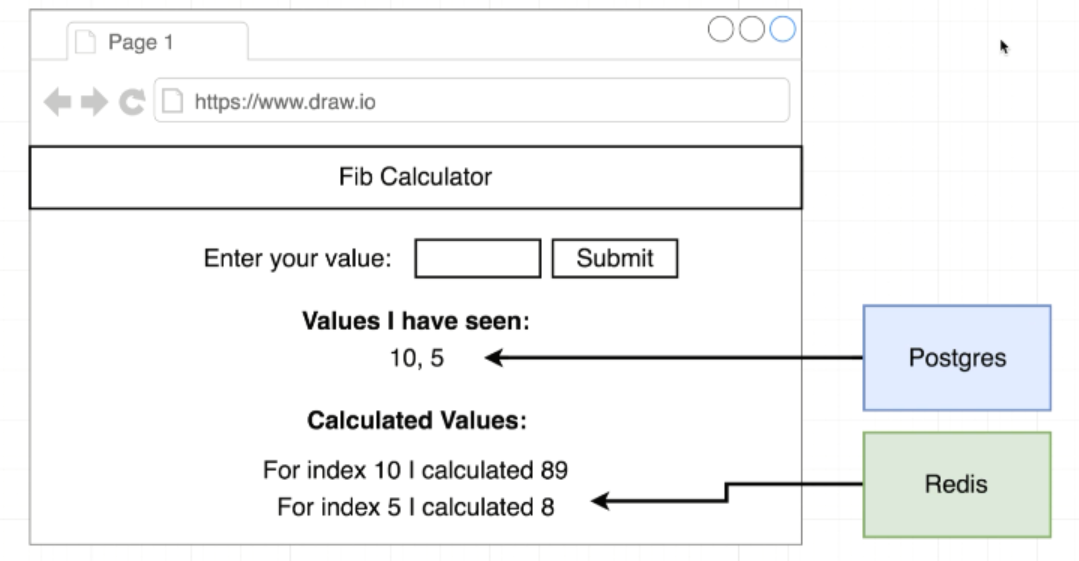
Nginx server do routing.

If we need front end code like html or javascript. Nginix will redirect to React server

If we need back end data then Nginix will redirect to Express Server.

Redis is in-memory data storage.

Postgres is a Database similar to MySql.



When click on submit, React app gonna make a api request or ajax request to the backend express server. When express receives a number to calculate a fabonocci series, it first take that number and store it inside the postgress.

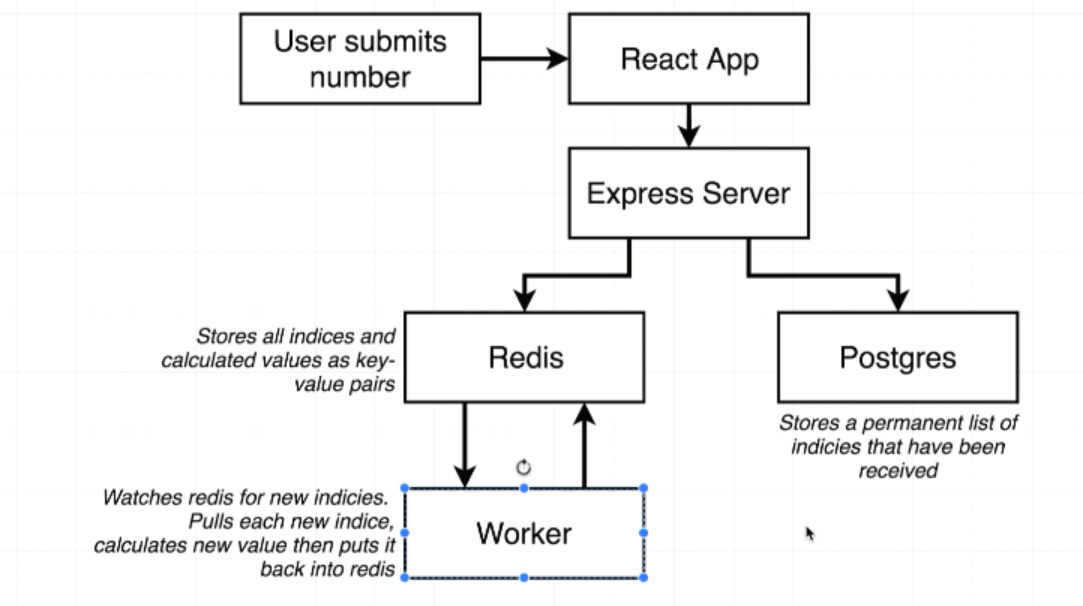
Postgres gonna have a permanent list of all the indices that have submitted to application.

At the same time the Express server is also going to take that index and store it in the Redis database.

When a new number show up in the redis database, it’s gonna wake up separate backend nodejs process that we call it as worker.

Worker watch redis for the new indices that show up. Anytime new value shows up in redis, the worker pull that value and calulate the appropriate fabinoci number. Worker put the calculated value back to the redis.

So it can be requested by the React Application and show up on the screen.



V—117:(Worker Process Setup):



Complex/worker/Keys.js:

module.exports = {

  redisHost: process.env.REDIS\_HOST,

  redisPort: process.env.REDIS\_PORT,

};

Complex/worker/Index.js:

const keys = require('./keys');

const redis = require('redis');

const redisClient = redis.createClient({

  host: keys.redisHost,

  port: keys.redisPort,

  retry\_strategy: () => 1000,

});

const sub = redisClient.duplicate();

function fib(index) {

  if (index < 2) return 1;

  return fib(index - 1) + fib(index - 2);

}

sub.on('message', (channel, message) => {

  redisClient.hset('values', message, fib(parseInt(message)));

});

sub.subscribe('insert');

V—118:(Important Note about PG module version):

In the next lecture, we will be creating a package.json file and adding our dependencies to it. To deal with a Postgres Image update we need to specify a different pg node\_module version.

In your **server/package.json** file:

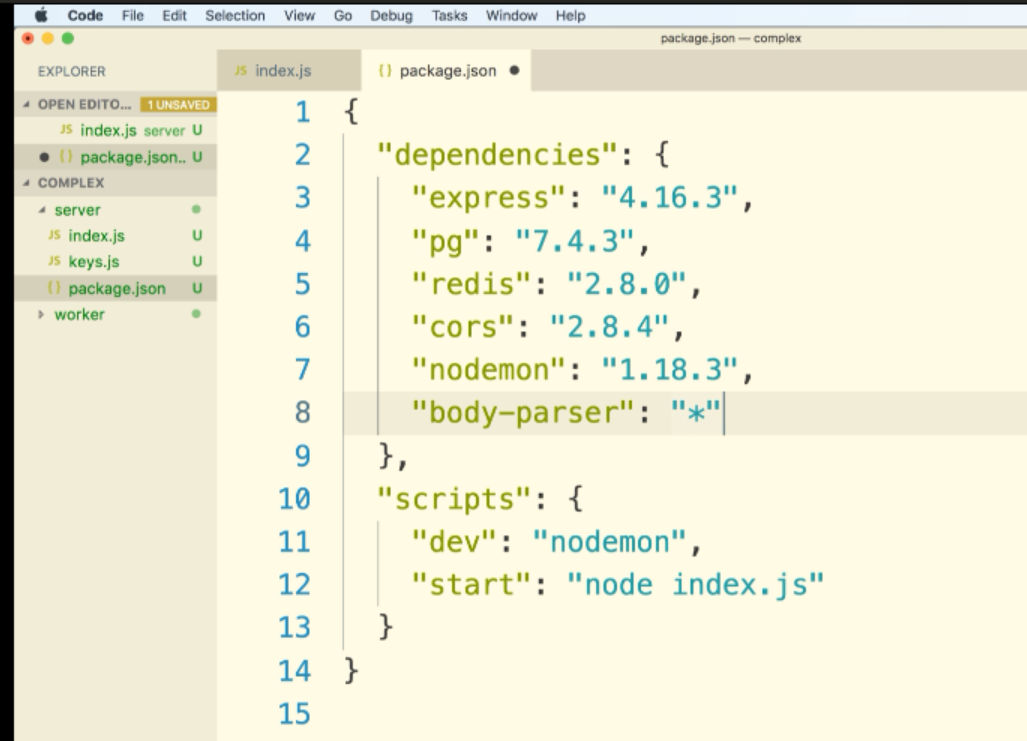
Change from this:

"pg": "7.4.3",

to this:

"pg": "8.0.3",

V—119:(Express API Setup):



Complex/server/keys.js:

module.exports = {

  redisHost: process.env.REDIS\_HOST,

  redisPort: process.env.REDIS\_PORT,

  pgUser: process.env.PGUSER,

  pgHost: process.env.PGHOST,

  pgDatabase: process.env.PGDATABASE,

  pgPassword: process.env.PGPASSWORD,

  pgPort: process.env.PGPORT,

};

V—120:(Important Update For Table Query):

In the upcoming lecture, we will be adding some code to our server's index.js to make a query and create a table. Due to a change in the new major version of the Postgres image, we need to modify this code to ensure that we delay the table query until after a connection is made.

In your **server/index.js** file:

Change these lines:

pgClient.on('error', () => console.log('Lost PG connection'));  
   
pgClient  
 .query('CREATE TABLE IF NOT EXISTS values (number INT)')  
 .catch(err => console.log(err));

to this:

pgClient.on('connect', () => {  
 pgClient  
 .query('CREATE TABLE IF NOT EXISTS values (number INT)')  
 .catch((err) => console.log(err));  
});

V—121:(Connecting to Postgres): && V—122:(More Express API Setup):

Complex/server/index.js:

const keys = require('./keys');

// Express App Setup

const express = require('express');

const bodyParser = require('body-parser');

const cors = require('cors');

const app = express();

app.use(cors());

app.use(bodyParser.json());

// Postgres Client Setup

const { Pool } = require('pg');

const pgClient = new Pool({

  user: keys.pgUser,

  host: keys.pgHost,

  database: keys.pgDatabase,

  password: keys.pgPassword,

  port: keys.pgPort,

});

pgClient.on('connect', () => {

  pgClient

    .query('CREATE TABLE IF NOT EXISTS values (number INT)')

    .catch((err) => console.log(err));

});

// Redis Client Setup

const redis = require('redis');

const redisClient = redis.createClient({

  host: keys.redisHost,

  port: keys.redisPort,

  retry\_strategy: () => 1000,

});

const redisPublisher = redisClient.duplicate();

// Express route handlers

app.get('/', (req, res) => {

  res.send('Hi');

});

app.get('/values/all', async (req, res) => {

  const values = await pgClient.query('SELECT \* from values');

  res.send(values.rows);

});

app.get('/values/current', async (req, res) => {

  redisClient.hgetall('values', (err, values) => {

    res.send(values);

  });

});

app.post('/values', async (req, res) => {

  const index = req.body.index;

  if (parseInt(index) > 40) {

    return res.status(422).send('Index too high');

  }

  redisClient.hset('values', index, 'Nothing yet!');

  redisPublisher.publish('insert', index);

  pgClient.query('INSERT INTO values(number) VALUES($1)', [index]);

  res.send({ working: true });

});

app.listen(5000, (err) => {

  console.log('Listening');

});

When we have a client that is publishing information we have to make a duplicate client because when a connection is turned into a connection that’s going to listen or subscribe or publish information it cannot be used for other purposes. So that’s why we are doing this duplicate things in both locations.

123:(Create React App Generation):

**updated 8-4-2020**

In the next lecture, Stephen will be going over how to install Create React App globally and generate the application. This method of generating a React project is no longer recommended.

**Instead of this:**

npm install -g create-react-app

create-react-app client

**We need to run this command:**

npx create-react-app client

**Important Reminder:**

Once you have generated the React app you will need to delete the local git repository that Create React App may have automatically initialized.

Inside the newly created client directory, run rm -r .git

If you miss this step, the client folder will be considered a submodule and pushed as an empty folder to GitHub.

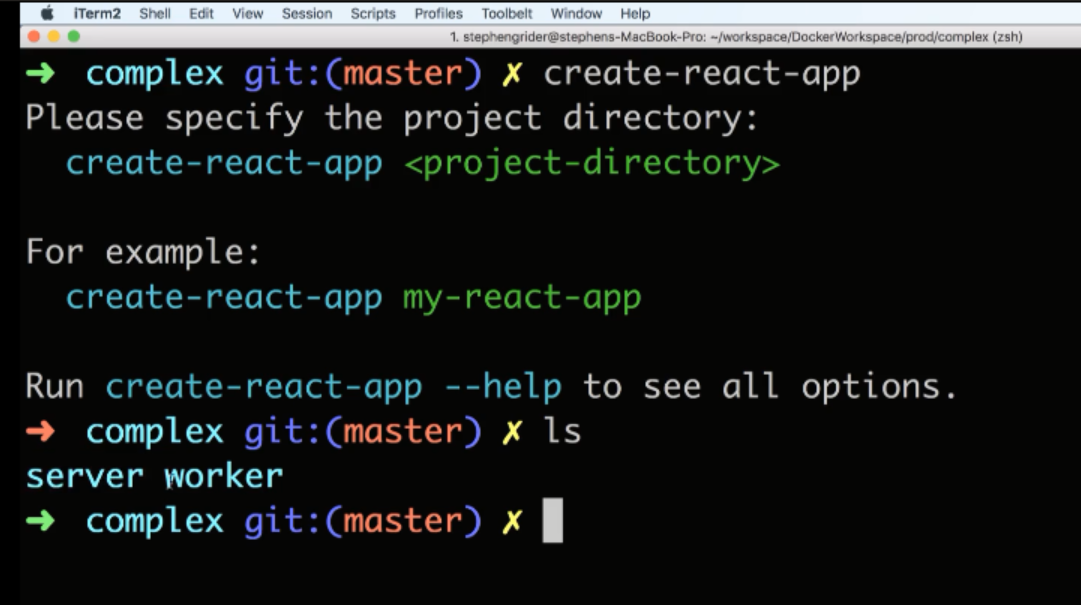
**Documentation:**

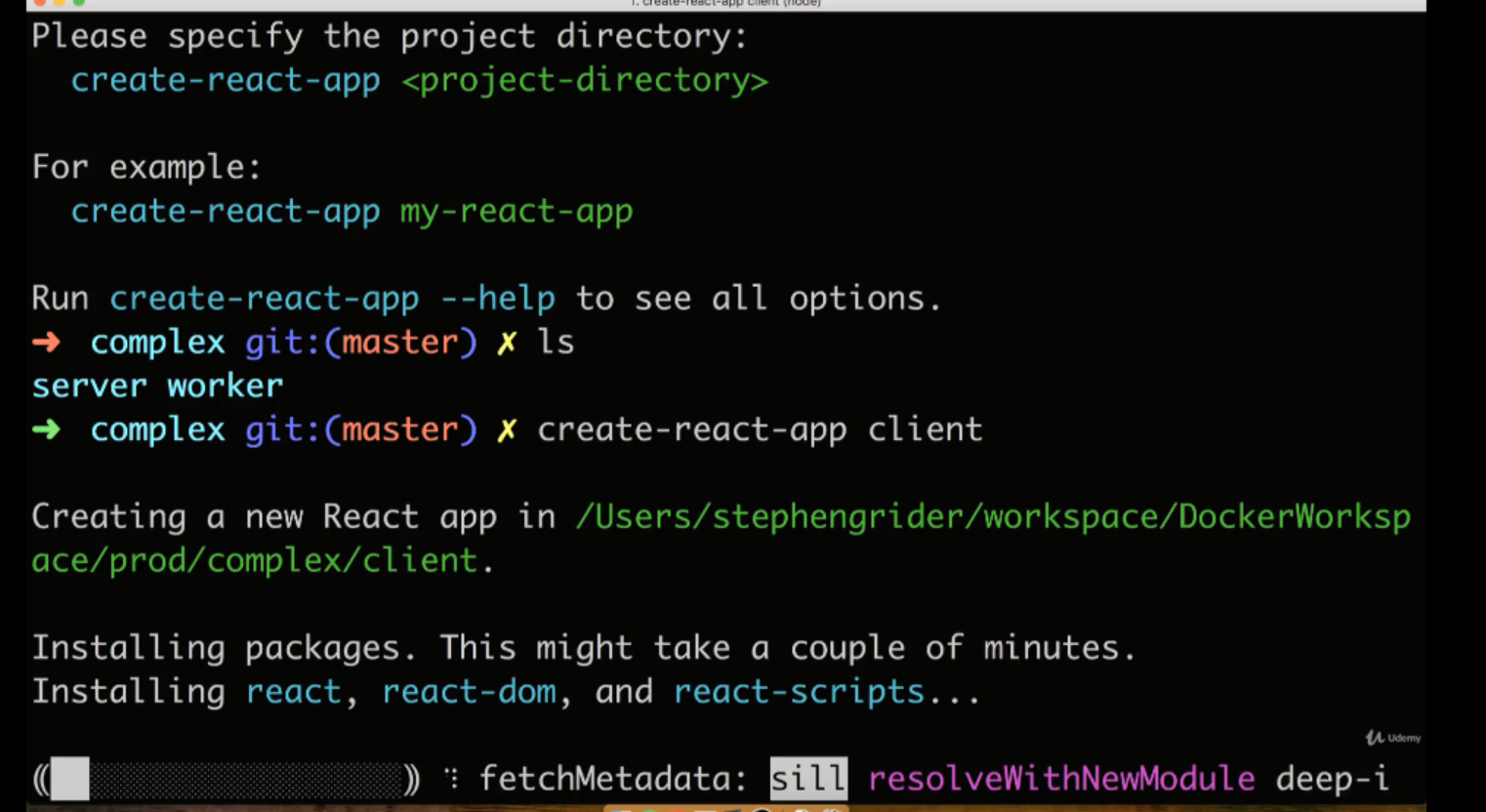
<https://create-react-app.dev/docs/getting-started#npx>

If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app to ensure that npx always uses the latest version.

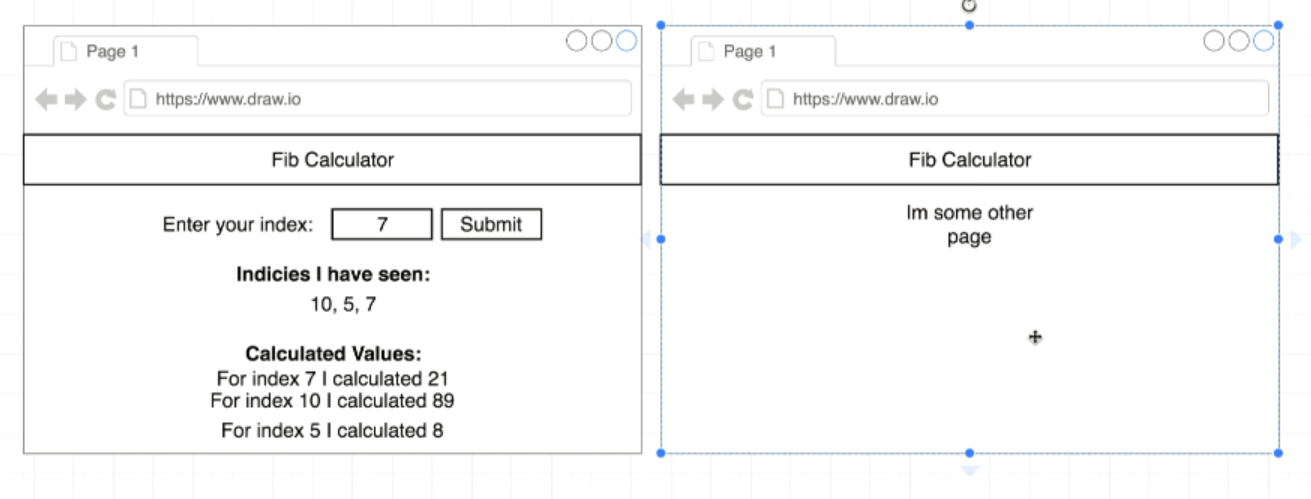
124:(Generating the React App):

Checking Create-react-app:





V—125:(Fetching Data in the React App):



Complex/client/src/Otherpages.js:

import React from 'react';

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

export default () => {

  return (

    <div>

      Im some other page!

      <Link to="/">Go back home</Link>

    </div>

  );

};

Complex/client/src/Fib.js:

Up to 2 sync methods in this video: --- me reference Remaining in V—125(Rendering Logic in the App)

When we pull data from reddis it’s gonna come in an object

import React, { Component } from 'react';

import axios from 'axios';

class Fib extends Component {

  state = {

    seenIndexes: [],

    values: {},

    index: '',

  };

  componentDidMount() {

    this.fetchValues();

    this.fetchIndexes();

  }

  async fetchValues() {

    const values = await axios.get('/api/values/current');

    this.setState({ values: values.data });

  }

  async fetchIndexes() {

    const seenIndexes = await axios.get('/api/values/all');

    this.setState({

      seenIndexes: seenIndexes.data,

    });

  }

  handleSubmit = async (event) => {

    event.preventDefault();

    await axios.post('/api/values', {

      index: this.state.index,

    });

    this.setState({ index: '' });

  };

  renderSeenIndexes() {

    return this.state.seenIndexes.map(({ number }) => number).join(', ');

  }

  renderValues() {

    const entries = [];

    for (let key in this.state.values) {

      entries.push(

        <div key={key}>

          For index {key} I calculated {this.state.values[key]}

        </div>

      );

    }

    return entries;

  }

  render() {

    return (

      <div>

        <form onSubmit={this.handleSubmit}>

          <label>Enter your index:</label>

          <input

            value={this.state.index}

            onChange={(event) => this.setState({ index: event.target.value })}

          />

          <button>Submit</button>

        </form>

        <h3>Indexes I have seen:</h3>

        {this.renderSeenIndexes()}

        <h3>Calculated Values:</h3>

        {this.renderValues()}

      </div>

    );

  }

}

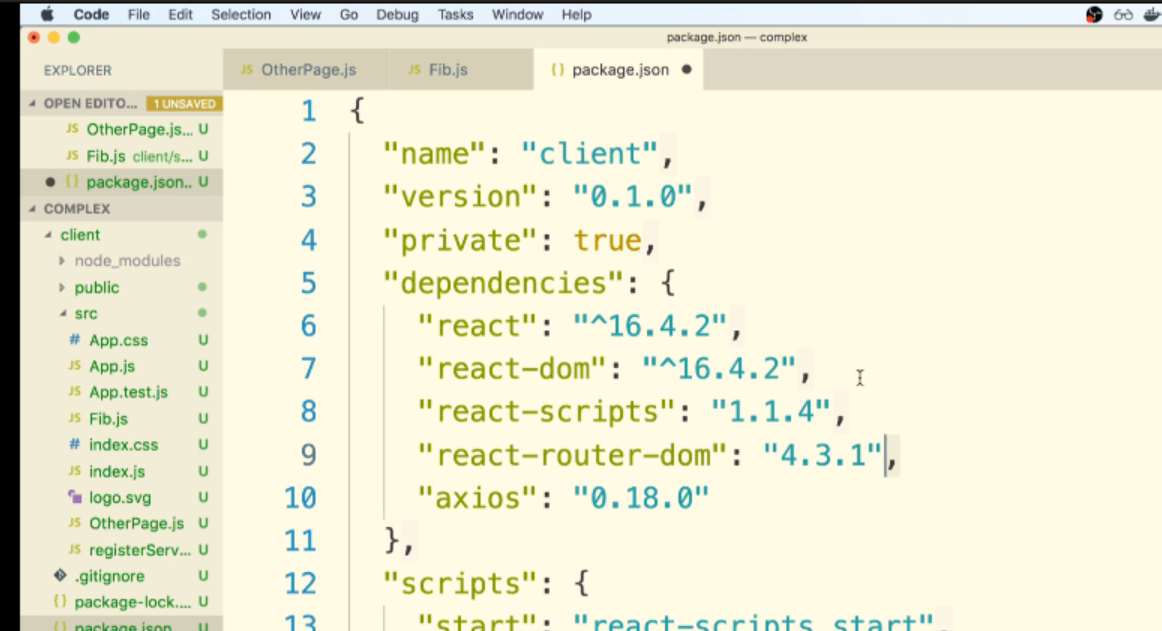
export default Fib;

V—127:(Exporting the Feb Class):

export default Fib;

V—128:(Routing in the React App):

Add react-router-dom and axios dependency :



Complex/client/src/App.js:

import React from 'react';

import logo from './logo.svg';

import './App.css';

import { BrowserRouter as Router, Route, Link } from 'react-router-dom';

import OtherPage from './OtherPage';

import Fib from './Fib';

function App() {

  return (

    <Router>

      <div className="App">

        <header className="App-header">

          <img src={logo} className="App-logo" alt="logo" />

          <a

            className="App-link"

            href="https://reactjs.org"

            target="\_blank"

            rel="noopener noreferrer"

          >

            Learn React

          </a>

          <Link to="/">Home</Link>

          <Link to="/otherpage">Other Page</Link>

        </header>

        <div>

          <Route exact path="/" component={Fib} />

          <Route path="/otherpage" component={OtherPage} />

        </div>

      </div>

    </Router>

  );

}

export default App;