coggle
made for free at coggle.it

Name: Nguyen Duc Hoang
Youtube: https://www.youtube.com/c/nguyenduchoang

- React.JS
  - Create a blank Project:
    Use create-react-app
    - npx = NPM package Runner
      - Firstly Install Nodejs
        npx create-react-app my-app
        cd my-app
        npm start
      - Edit src/index.js:
        import React, {Component} from 'react'
        import ReactDOM from 'react-dom'
        import './index.css'
      - Open http://localhost:3000
        - React.js & JavaScript
          - Javascript ES6
            - Variables
              - Define a variable:
                let x = 10 //inside a block
                var y = 20 //inside a .js file
                - Compare value:
                  var x = 10
                  if(x == '10') {//true}
                  - Compare value and type:
                    var x = 10
                    if(x === 10) {//Return true}
                    - Define an object(key-value):
                      let person = {name:"Hoang", email: "hoang@gmail.com"}
                      Key maybe a string, number
                      - Destructuring an object:
                        const {name, email} = person;
                    - Define an array of objects:
                      var products = [{name: "Iphone X", year: 2018}, {name: "iphone 3", year: 2011}]
                      - Iterate an array:
                        products.forEach((element, index) => {...})
                      - Save local storage:
                        localStorage.setItem('isLoggedIn', "1")
                        key-value must be strings !
                  - Alert a value:
                    alert(`x = ${x}`)
                    - Print a value:
                      console.log(`x = {x}`)
              - Object as "Dictionary":
                var users = {
                "111": {name:"...", email:"..."},
                "222": {name:"...", email:"..."}
                }
                - How to convert "Dictionary" to "Array" ?
                  - Object.keys(dictUsers).map(key => ({...dictUsers[key]}));
            - Functions
              - Basic function:
                function sum(x, y) {return x+y}
                - Arrow function:
                  const sum = (x, y) => {return....}
                  - One-line function:
                    const sum = (x, y) => x+y
                  - Arrow function with 1 param:
                    const square = x => x*x
              - Promise
                - Do something asynchronously
                  - const doPromiseA = (param1, param2) => {return new Promise(....)}
                    - Resolve, reject are functions:
                      new Promise((resolve, reject)=>{
                      //Do something
                      if(success) { return resolve("ok")}
                      else { return reject("failed") }
                      })
                      - Replace Promise with async/await
                        - const doAsyncAwait = async () => {
                          try {...} catch(e) {...}
                          }
                        - Inside try...catch:
                          try {
                          await doPromiseA()
                          } catch(e){....}
            - Array, List
              - Array of objects:
                let products = [{name: "iphone X", year: 2018}, {name: "iphone 6", year: 2016}]
                - Iterate an array:
                  products.forEach((product, index) => {
                  console.log(Name = ${product.name}, index = ${index})
                  })
                  - Map and update an existing array:
                    products = products.map((product, index) => { return ...})
                    - What's inside the "arrow function":
                      if(product.name === "iphone 6") {
                      return {...product, year: 2020}
                      }
                      return product
                  - Delete an item in list:
                    products = products.filter(product => {
                    return product.name !== "iphone 6"})
                  - Find an item:
                    let foundProduct =
                    products.find(product => {return product.year === 2018})
                - Convert array to string:
                  strings = ["name='Hoang'", "age=30"]
                  let outputString = ", ".join(strings)
                - Add more items(mutable):
                  products.push({name: "iphone 5", year: 2015})
                  - Add more items(un-mutable):
                    products.concat({name: "iphone 5", year: 2015})
                    - Sort an array:
                      let users = [{name: "John", age: 22}, {name: "Hoang", age: 30}]
                      users.sort((user1, user2) => {return user1.age - user2.age})
          - React Component
            - 1.Extends Component
              2.Must implement render()
              3.Is a class or function
              4.One components can contains many other components
              - class ProductComponent extends Component {
                render() {
                return (<div className="...">
                <h1>I like {this.props.product.name}</h1>
                </div>)
                }
                }
                - JSX:
                  <h1>I like...</h1>
                  OR
                  <div className...></div>
                  - A bit different with HTML tag:
                    "className", not "class"
                  - JSX in short form:
                    <ProductComponent />
                    - render() is re-run if data changed
                  - JSX with props:
                    <ProductComponent product={{name: "iphone", year: 2019}} />
                - What is component's data ?
                  - props and state
                    - props
                      - 1.Public
                        2.Can be changed from "parent Component"
                        - Define a Component with props
                          - const ComponentA = (props) => {
                            return <div>...</div>}
                          - const ComponentA = ({onClick, productName, productYear}) => {
                            return <div>...</div>}
                            ...
                            export default ComponentA
                            - import PropTypes from 'prop-types'
                              ComponentA.propTypes = {
                              onClick: PropTypes.func.isRequired,
                              productName: PropTypes.string.isRequired,
                              productYear.int.isRequired
                              }
                              - If propTypes is array of Object?
                                - todos:
                                  PropTypes.arrayOf(PropTypes.shape({id: PropTypes.bool.isRequired}))
                    - state
                      - 1.Private
                        2.Can be changed inside "this Component"
                        - setState with object:
                          setState({product: {name: "iphone", year: 2019}})
                          - Manage State using Redux
                            - Install redux(2kB, very small!):
                              npm install --save redux
                              npm install react-redux
                              pm install --save-dev redux-devtools
                              - "store", a global Object
                                - Define in a "store.js" file
                                - import { createStore } from 'redux'
                                  let store = createStore(aReducer)
                                  export store...
                                - console.log(store.getState()
                              - Action:
                                When user click a button
                                - Action is a "function"
                                  - const addNewProduct = (newProduct) => ({ type: 'ADD_PRODUCT', newProduct })
                                    - { type: 'UPDATE_PRODUCT', id:1, updatedProduct: {name: 'iphone 3', year:'2007'}}
                                    - { type: 'DELETE_PRODUCT', id:1}
                                    - Dispatch "without button":
                                      store.dispatch({..an action...})
                                    - Async Actions ?
                              - Reducers:
                                Contains functions which convert oldState => newState
                                - function productsReducer(state = "initial state", action) {
                                  switch action.type = ...{
                                  case ...: return newState;
                                  }
                                  }
                                  - import { combineReducers, createStore } from 'redux';
                                    const ABReducer = combineReducers({ productsReducer,customersReducer})
                                    const store = createStore(ABReducer)
                                - function customersReducer(state=..,action)
                                  {...}
                              - Container
                                - Is a Component, but states(variable + function) => convert => props
                                  - const mapStateToProps = state => ({
                                    products: ...
                                    })
                                    - import { connect } from 'react-redux'
                                      const ContainerA = connect(mapStateToProps,mapDispatchToProps)(ComponentA)
                                      export default ContainerA
                                      - In ContainerA:
                                        this.props.funcA(...)
                                        this.props.products
                                  - const mapDispatchToProps = dispatch => ({
                                    funcA: (paramA) =>dispatch(addNewProduct(paramA))
                                    })
                                    - Another way:
                                      const mapDispatchToProps = dispatch => ({
                                      funcA: {(paramA) => dispatch({type: 'INCREASE_COUNTER'})
                                      })
                        - setState with function:
                          setState((previousState) => {...return newState})
                - Form
                  - class SignUpForm extends Component {
                    render() {return <form onSubmit={...}></form>}
                    }
                    - this.state = {...INITIAL_STATE}
                - Basic components
                  - <button></button>
                    1.className
                    2.onClick={}
                    3.type => "submit"
                    4.disabled={false}
                  - <input.../> props:
                    1.name
                    2.value
                    3.onChange={...}
                    4.type
                    5.placeholder
                    - <input onChange={...}..>
                      ...
                      this.setState({event.target.name: event.target.value})
                    - "input" is a "controlled component"
            - HOC - Higher-Order Component =
              Nested functions
              - import {x} from '...';
                export const withX = Component => props => (
                <div><Component {...props} x = {x}></div>
                );
                - How to call function "withX" ?
                  class ComponentA {...}
                  export const withX(ComponentA);
                - How to call 2 HOC ?
                  export const withY(withX(ComponentA))
                  - npm install recompose
                    - import {compose} from 'recompose';
                      const ComponentB = compose(withX, withY)(ComponentA);
                      export {ComponentB};su
              - Do the same with withY:
                export const withY =...
            - Lifecycle
              - When mounting:
                1.constructor()
                2.render() => required
                3.componentDidMount()
                - When updating(props or state changed):
                  1.shouldComponentUpdate(nextProps, nextState)
                  2.render(), run if (1) = true
                  3.getSnapshotBeforeUpdate(prevProps, prevState)
                  4.componentDidUpdate(prevProps, prevState, snapshot)
                  - componentDidUpdate:
                    must check:
                    if(prevProps.email !== this.props.email)...
                    to prevent "infinite loop"
                    - React.createRef()
                      - this.refA = React.createRef()
                        ...
                        const jsxA = this.refA.current
                        ...
                        <div ref={this.refA}>
                - In constructor:
                  1.Initialize state
                  2.Binding action:
                  this.handleClick = this. handleClick.bind(this)
          - React Router & Firebase Auth
            Firebase bought by Google,2014
            - A file for defining routes:
              export const SIGN_UP = '/signup'
              ...
              - App Component changed
                - import { BrowserRouter as Router, Link } from 'react-router-dom';
                  const App = () => (<Router>
                  <div>
                  <Link to={SIGN_UP}>Sign up</Link>
                  <Link ....>
                  </div>
                  <Router>)
                  - Route to Component Mapping
                    - ...<Router>
                      <Route exact path=...>
                      <Route path={SIGN_UP} component={SignUpPage} />
                      </Router>
                      - Create a "FirebaseManager" class
                        import app from 'firebase/app'
                        in constructor:
                        app.initializeApp(config)
                        - Add auth to FirebaseManager:
                          import 'firebase/auth';
                          this.auth = app.auth()
                          - FirebaseManager:
                            function registerUser:
                            this.auth.createUserWithEmailAndPassword(email, password)
                          - Others:
                            this.auth.signInWithEmailAndPassword(...)
                            this.auth.sendPasswordResetEmail(email)
                            this.auth.signOut()
                            this.auth.currentUser.updatePassword(newPass)
                            - Implement SignupForm
                              - onSubmit:
                                this.props.firebaseManager.registerUser...
                                then(authUser => ..).catch(error=>...)
                                event.preventDefault()
                              - Error:
                                <button />...
                                {this.state.error &&<p>this.state.error.message</p>}
                        - Use "context" API to
                          manage 1 Firebase object
                          - Create context.js:
                            const FirebaseContext = React.createContext(null);
                            - Provider:
                              <FirebaseContext.Provider value={new FirebaseManager()}>
                              <App />
                              </FirebaseContext.Provider>
                            - Consumer:
                              const ComponentA = () => {
                              <FirebaseContext.Consumer>
                              {(firebaseManager) => {
                              return <div>...</div>
                              }}
                              </FirebaseContext.Consumer>
                              } :
                              - Let's create HOC
                                (Higher-Order Component) !
                      - npm install firebase
                        - Session Handling with Firebase
                          - Let's write "check login" in App
                          - Check loggedIn User in Firebase:
                            In componentDidMount:
                            this.listener =
                            firebaseManager.auth.onAuthStateChanged(
                            user => {if(user)...setState...})
                            - Remove listener of unused:
                              componentWillUnmount() {this.listener}
                        - Firebase Realtime DB(React)
                          - In FirebaseManager:
                            import app from 'firebase/app';
                            import 'firebase/auth';
                            import 'firebase/database';
                            ...
                            this.db = app.database();
                            - Firebase Ref
                              - getUserRef = (userId) =>
                                this.db.ref(`users/${userId}`);
                                - firebaseManager.getUserRef('123').set(
                                  {name, email})
                              - getAllUsersRef = () =>
                                this.db.ref('users');
                                - firebaseManager.getAllUsersRef().on
                                  ('value', snapshot =>
                                  {...snapshot.val();})
                                - Remove Listener: in componentWillUnmount()
                                  firebaseManager.getAllUsersRef().off();
            - npm install react-router-dom
            - Open Firebase Console, create Project
              - Choose Web, not IOS, Android
                - You may see:
                  1.A <script> tag
                  2.A "var config"
                  - save secret info in .env file
                    REACT_APP_API_KEY=...
                    ....
                    NODE_ENV === 'development'
                    - const config = {
                      apiKey:
                      process.env.REACT_APP_API_KEY,...
                      }
                      - npm install firebase
                  - Open Authentication
                    1.Choose tab "Sign-in Method"
                    2.Enable Email/Password signin
                    3.Facebook,Google,...
                    - Open Left Tab:
                      Database, choose Realtime DB
                      OR: Cloud Firestore
                      - With Firestore, you can;
                        1.Add collection(table)
                        2.Add document(record)
                        3.Add field
                      - Cloud Firestore, set monthly quotas to prevent payment
                      - Click Left Tab, Hosting
                        - Deploy React App to Firebase:
                          1.npm install -g firebase-tools
                          2.firebase login
                          3.firebase init => choose Hosting
                          select your project
                          4.firebase deploy
    - React + Foundation
      - npm install react-foundation --save
        ...
        import Foundation from 'react-foundation';
        - Basic components
          - Buttons
            - Button:
              1.color={Colors.SUCCESS}
              2.isHollow => "outline button"
              3.isExpanded
              - PRIMARY, SECONDARY, SUCCESS, ALERT, WARNING
            - ButtonGroup:
              1.size
              2.can contains many buttons
              3.isExpanded
              4.stackFor={Breakpoints.SMALL}
            - Link:
              1.size={Sizes.TINY}
              2.color={Colors.PRIMARY}
              3.isDisabled
              4.isDropdown
              - TINY, SMALL, LARGE
            - Switch:
              1.input={{ type: InputTypes.RADIO, name: '...', defaultChecked: true }}/>
              2.size
              3.active={{ text: 'Yes' }} inactive={{ text: 'No' }
          - Menu
            1.alignment={Alignments.RIGHT}
            2.isExpanded
            3.isVertical
            4.isVertical
            5.isSimple
            6.isNested
            7.iconsOnTop
            - Contains MenuItem
              - MenuItem can contains <a>
                - Inside <a> may contains Icon:
                  1.name="fi-list"
              - MenuItem:
                isActive => has background color
            - isNested => Menu is inside a MenuItem
          - Pagination:
            1.aria-label="Pagination"
            2.isCentered
            - Contains:
              PaginationPrevious
              PaginationItem
              <PaginationEllipsis/> => "..." sign
              PaginationNext
              - Inside PaginationItem:
                <a aria-label="Page 12">12
                - PaginationItem
                  1.isCurrent
                  2.isDisabled

- NodeJS + Postgres
  - Nodejs & Express
    - Create a blank Nodejs:
      npm init -y
      - Install express, pg for Nodejs:
        npm i express pg
        - import some packages:
          const express = require('express')
          const bodyParser = require('body-parser')
          const app = express()
          const port = 3000
          ...
          app.listen(port, () => {})
          - Use middleware for parsing JSON:
            app.use(bodyParser.json())
            app.use(bodyParser.urlencoded({extended: true}))
          - First simplest api:
            app.get('/', (request, response) => {
            response.json({...})
            })
            - Now connect Nodejs+Postgres
              - const {Pool} = require('pg').Pool
                const connectionPool = Pool({user: 'hoangnd',
                host:,..database:..,password:.., port: 5432})
                - In database.js
                  Function for GET:
                  const getUsers = (req, res) => {
                  connectionPool.query("SELECT * FROM...", (error, results) => {response.status(200).json({....})})
                  }
                  ...
                  module.exports = {getUsers,..}
                  ...
                  In other file:
                  const db = require('./database.js')
                  app.get('/users', db.getUsers)
                  - Query with params:
                    .query('SELECT * FROM users WHERE id = $1', [id], (error, results)...
                  - INSERT INTO users (name, email) VALUES ($1, $2)', [name, email]...
                  - 'UPDATE users SET name = $1, email = $2 WHERE id = $3'
                  - DELETE FROM users WHERE id = $1', [id]
                - Test Restful API(POST, GET, PUT, DELETE) using Postman
  - PostgreSQL, since 1997
    - Install on Windows: using Installer
      - Search psql, click to run
    - on MacOS:
      brew install postgresql
      - Start/stop service
        brew services start postgresql
        - Default user/database = postgres/postgres
          - psql postgres
            - \conninfo
            - \c => Connect to a new DB
            - \l => List all databases
              - CREATE ROLE hoangnd WITH LOGIN PASSWORD 'mypasssword';
              - ALTER ROLE hoangnd CREATEDB;
                - Login again
                  - psql -d postgres -U hoangnd
                    - CREATE DATABASE apiTutorials;
                    - CREATE TABLE users (
                      userId SERIAL PRIMARY KEY,
                      name VARCHAR(100),
                      email VARCHAR(100));
              - \du => for checking
            - \dt => Display all tables
            - \du => Display all users/roles

- Deploy app to Heroku
  - React App
    - cd your_react_app_folder
      npm install --save serve
      - Write some scripts in package.json:
        "dev": "react-scripts start",
        Production build:
        "start": "serve -s build",
      - Login Heroku dashboard, create new App
        Open "Deploy" tab, choose Github
        Build and run in heroku:
        "heroku-postbuild": "npm run build"