NEXTJS COURSE

Section 1. Introduction

1. Welcome to the course. Presents a home booking application.

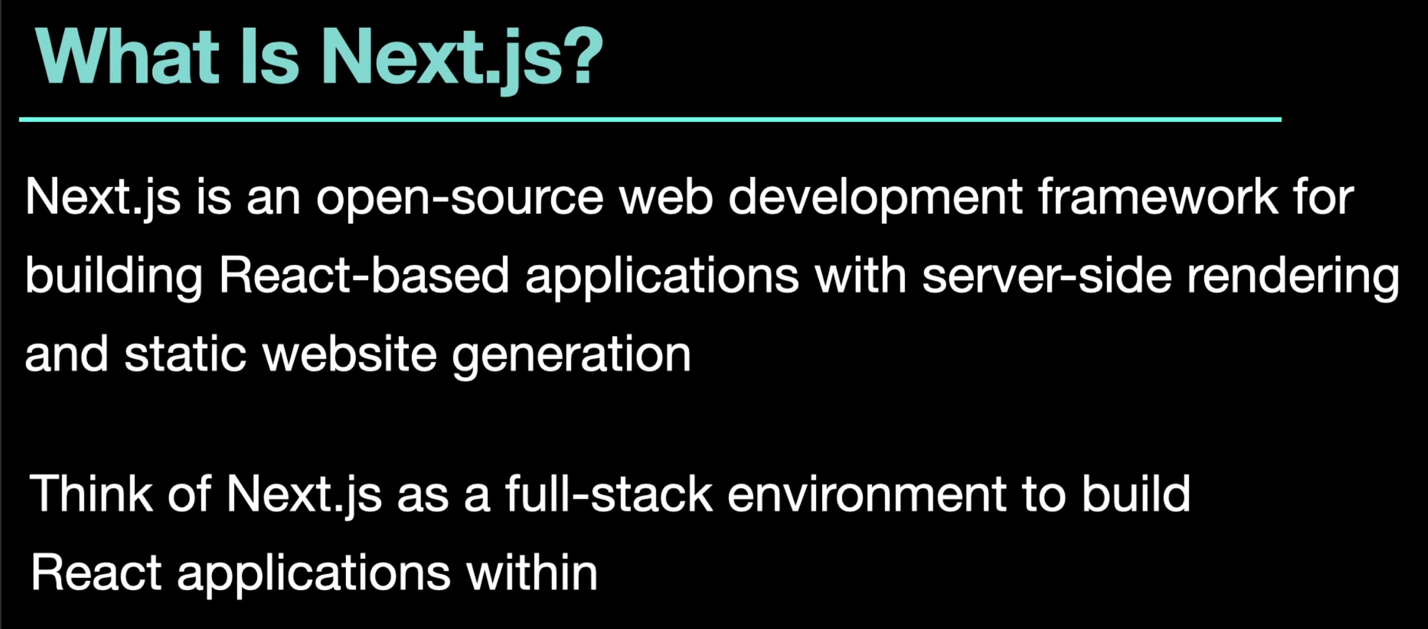
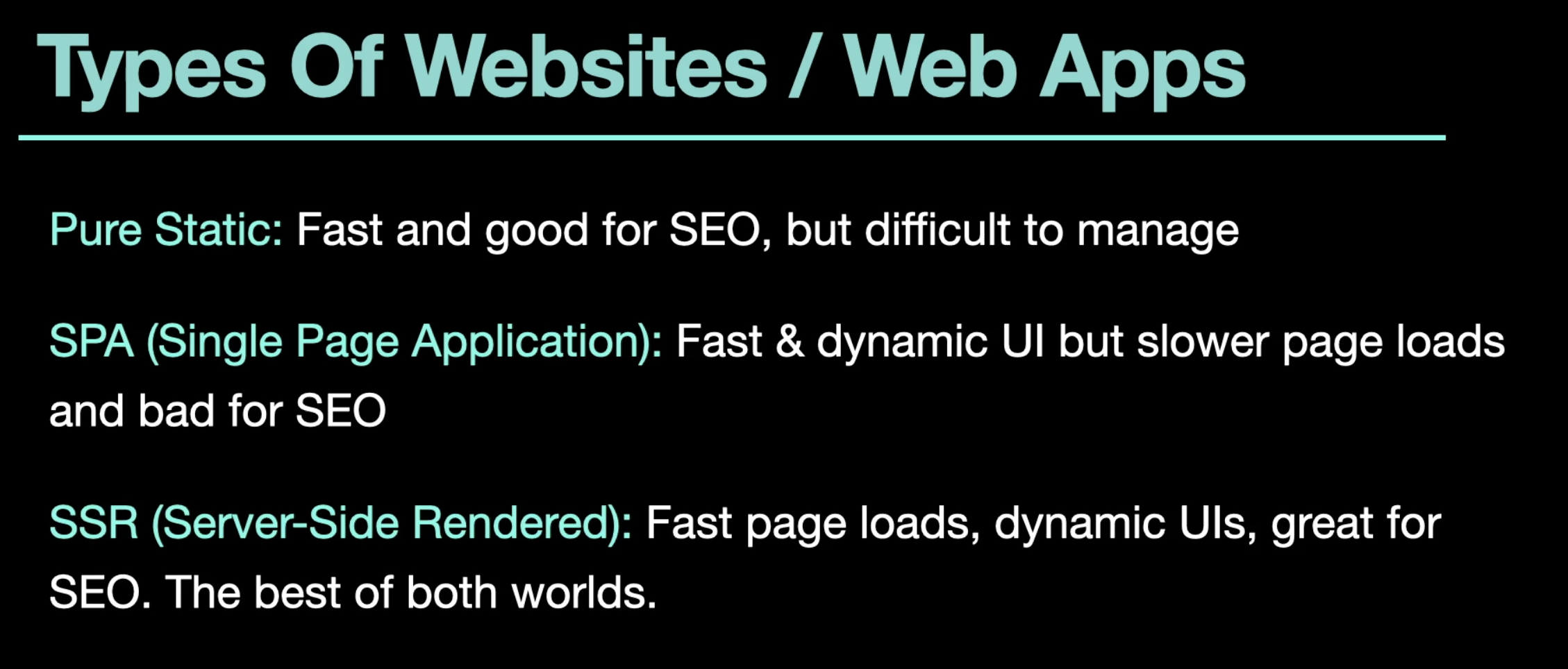
* Notable features
* Book marking, share to multimedia platforms, internal messaging system,

geocoding

1. Property Pulse Demo

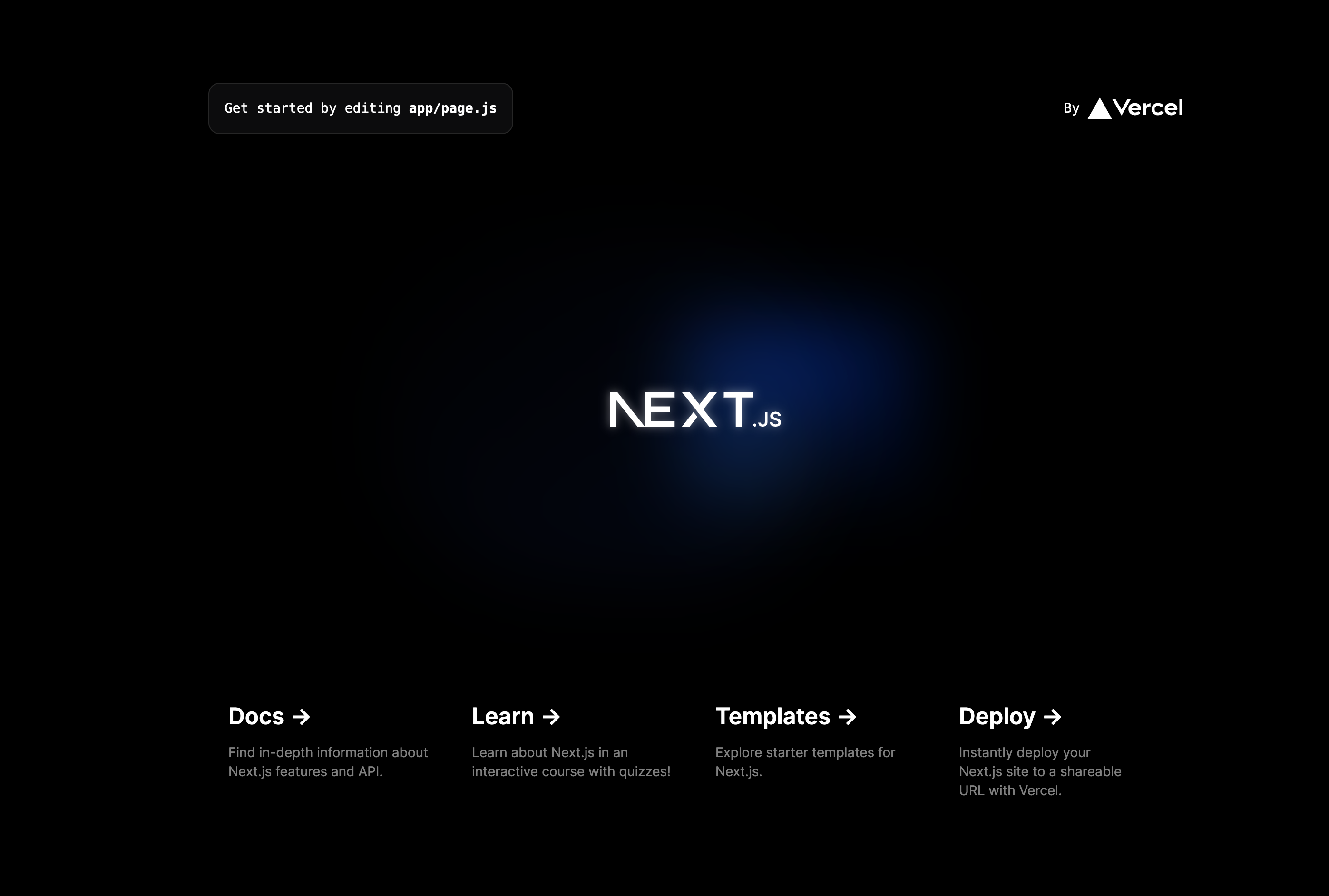
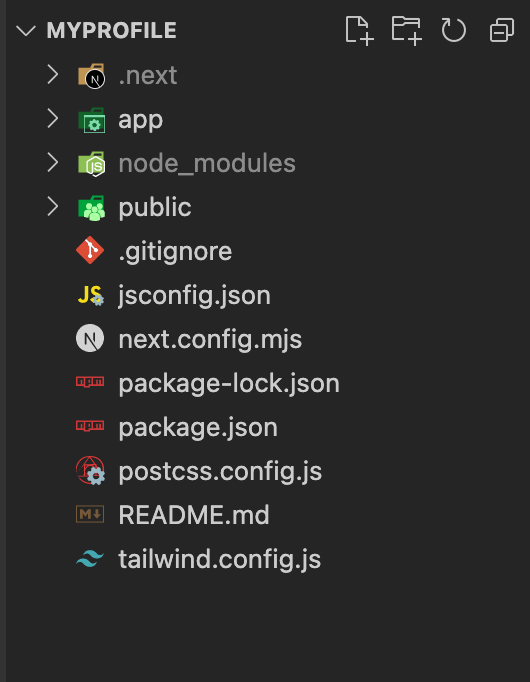
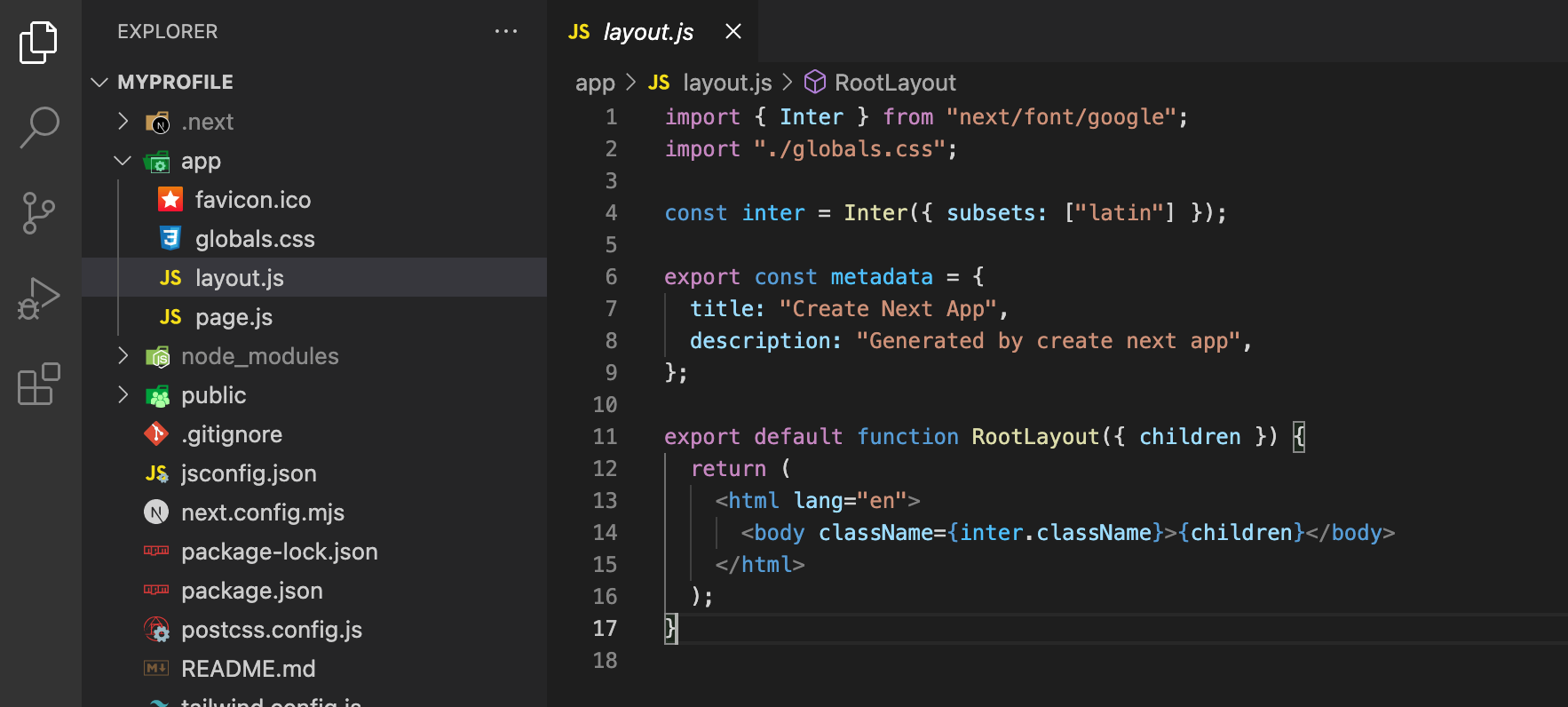
* Save data in mongoDB then use google authentication for users
* Search Box function
* Can set the Featured properties to be true or false so that people can pay the site
* Properties displayed is in random
* Property page upon click
* Cloudinary is to be use to store property pictures
* iImplementation of Geolocation of the property
* Use the React SHare for the property
* Implement Pagination and search box
* Bookmarking function
* Contact Property Manager form
* Profile Page
* feature the list of listings
* add and remove bookmark
* Multiple image upload to cloudinary
* Convert the address to lat and long and display it on the map

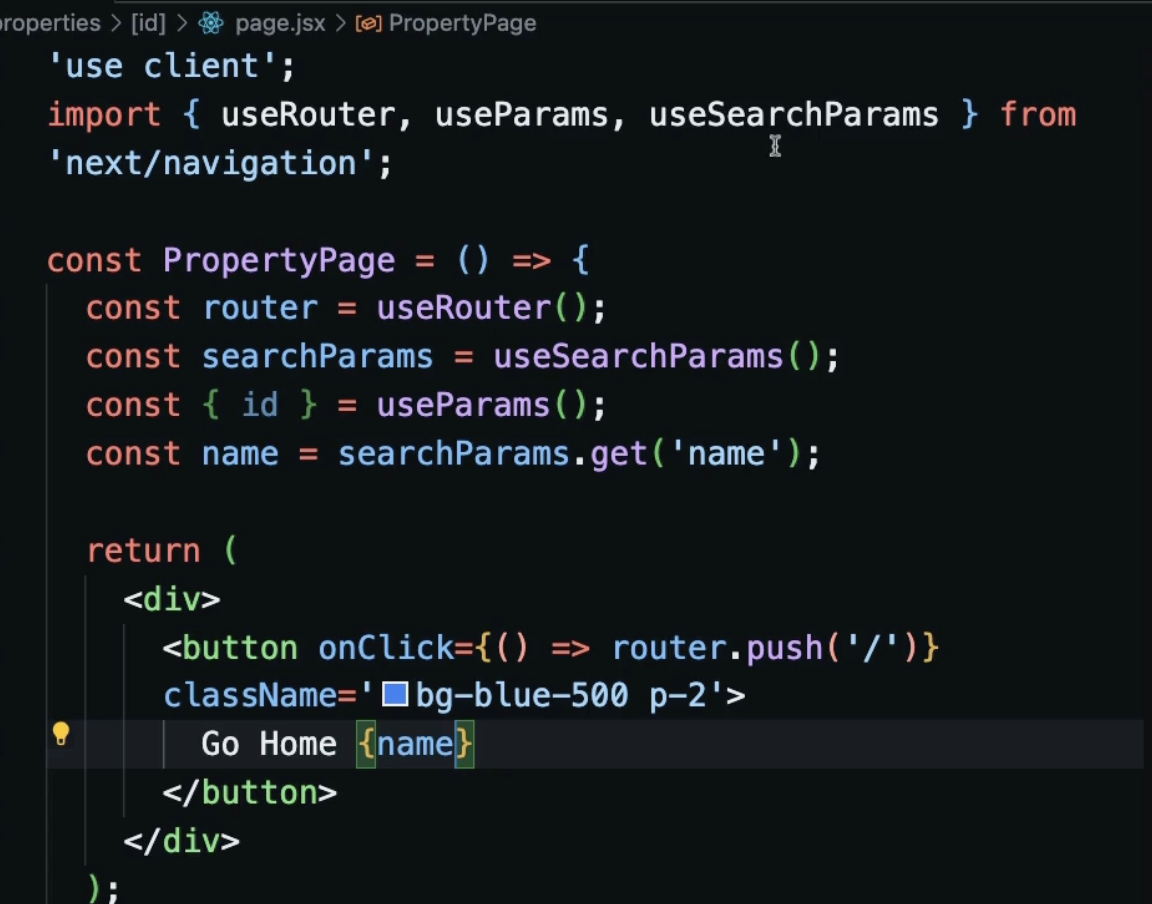
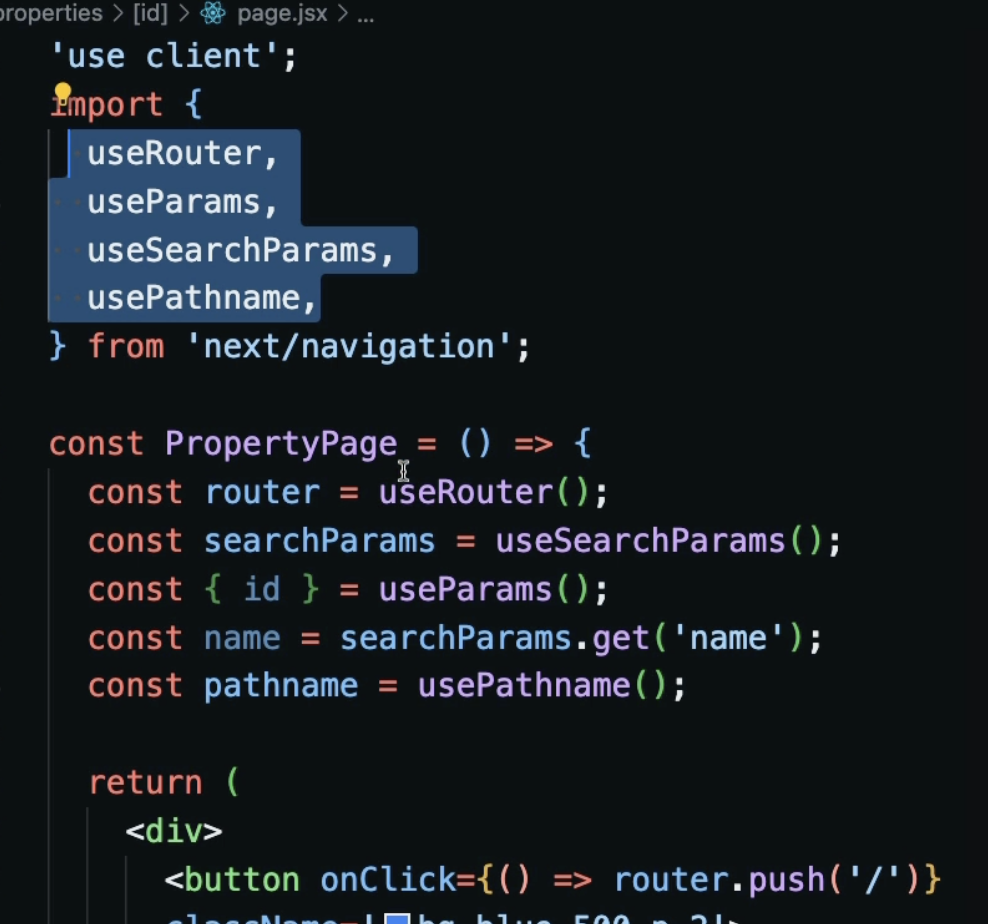
1. What is NextJS

* 
* React is a Single Page App, while NextJs provide ways for React projects to be server side rendering rather than everything bind in the client
* NextJs helps to generate static website which is very fast
* 
* Types of websites
  + Pure Static – difficult to manage as there’s bunch of html files. Need to go to each html if there’s a change needed. Each page is less interactive
  + SPA (single page rendering) – as it is all loaded by javascript thus making it bad for SEO. It’s fast and dynamic thought
  + SSR (server side rendering) – Best of both SPA and Static website. It is static which makes it good for SEO and at the same time Javascript makes it interactive. Initial load time is fast
  + SSG (static site generated) – it will generate html and css files and send it to the browser. This means that the server will not do any work. Good for blogs and portfolios but not good for fast and very dynamic sites
* NextJS Features
  + File based routing – no need to use react-router. Just create component and put it inside the app folder. It can create nested and dynamic routes
  + API Routes – can interact with databse with the use of mongoDB and mongoose. No need to use expressJS
  + React Server Component (RSC) – Reacy components that are rendered in server. Great for SEOs. Faster load time, can handle API keys but cannot use React Hooks
  + Data fetching even without useEffect
  + Supports .env (environmental variables) which stores API keys, mongoDB connect strings
  + Customize head tags, meta tags. Great for SEO
  + Image optimization – allow to optimize images and utilize lazy loading
  + Automatic code splitting – resulted to smaller bundles that improves the loading
  + Typescript support
  + CSS modules, tailwind, global css files and Support
  + Fast refresh, instant feedback during development

1. Requires node, editor like vs code, git installed in the computer. Use the github repo upon uploading to Vercel
   * VS Extensions
     + ES7 React/redux/react-native snippet extension
     + Prettier
     + Javascript ES6 snippets

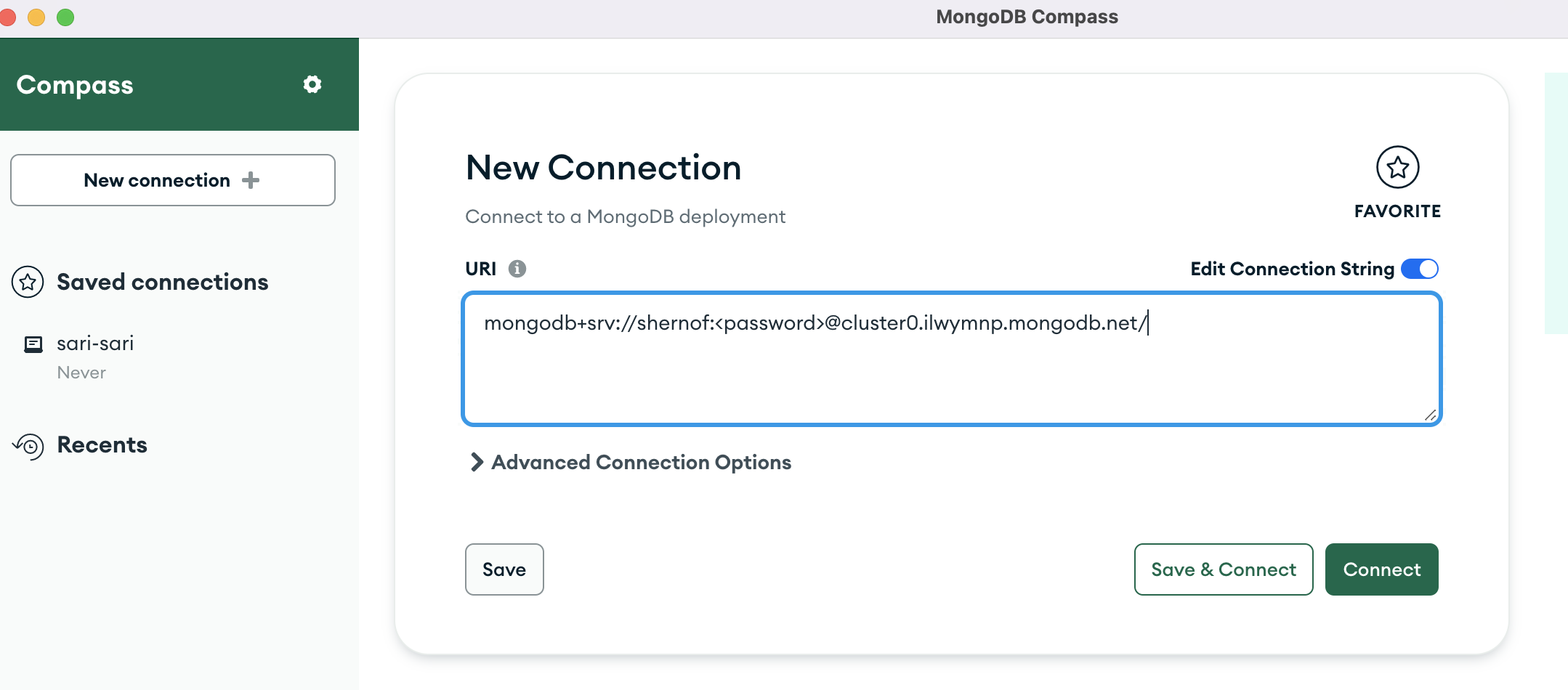
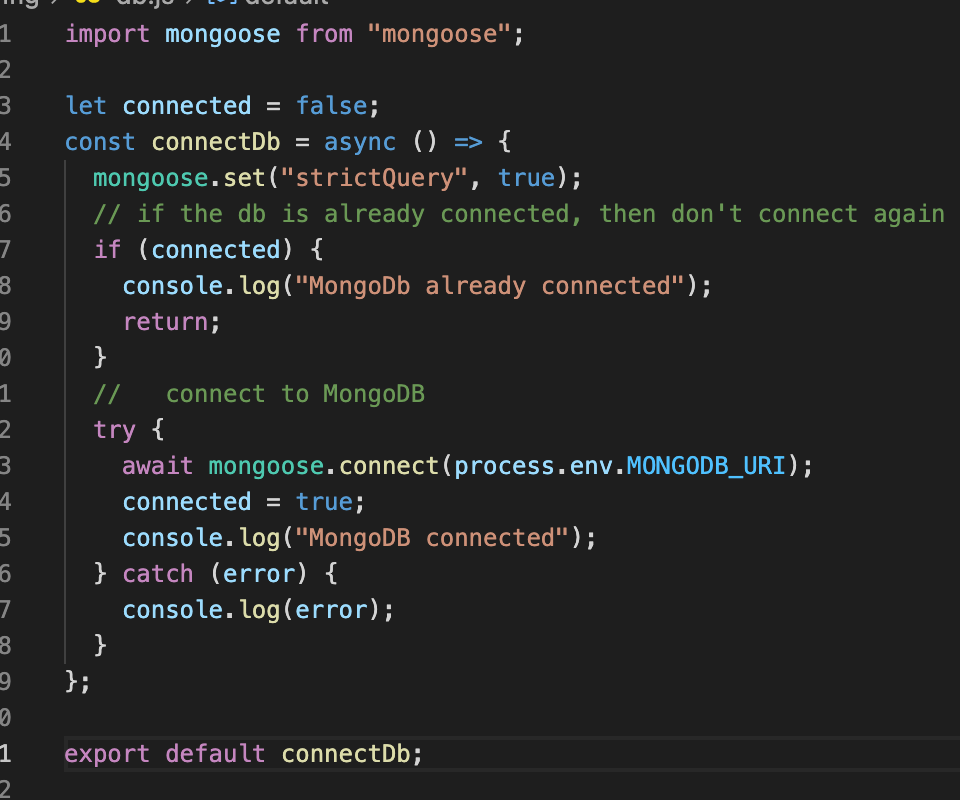
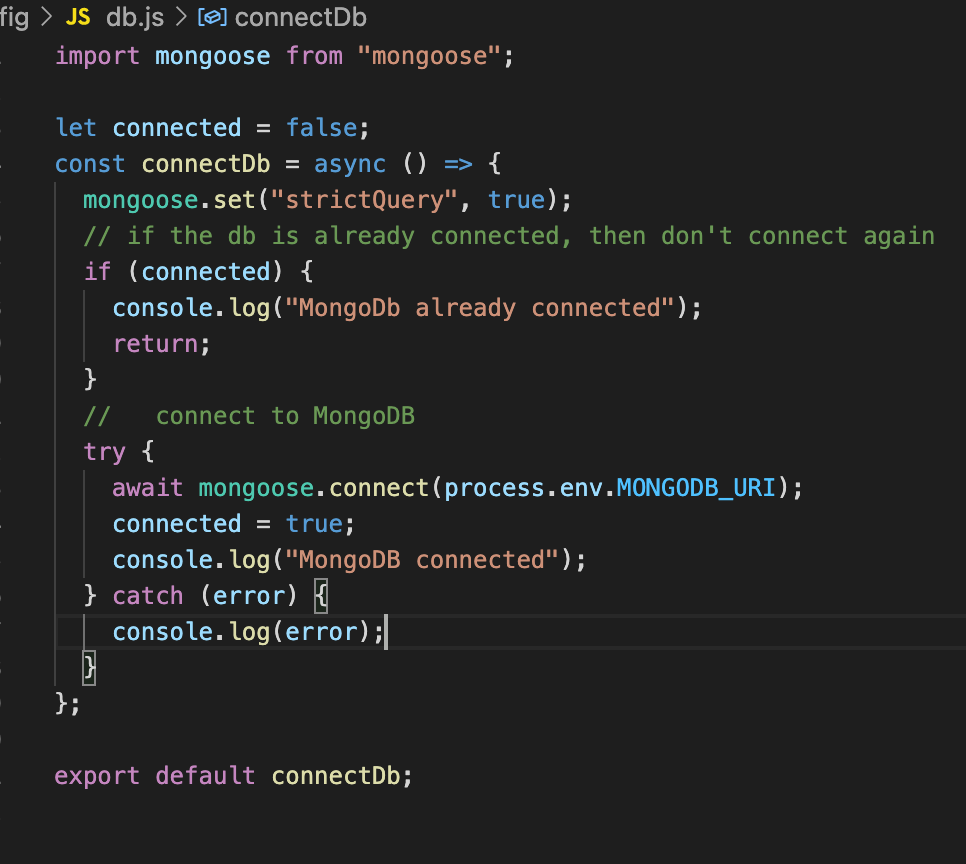
Section 2. NextJs fundamentals and project start

1. New project and folder structure
   * Resources – theme files
   * Create a new project = **npx create-next-app@latest.** Fill up the question and just disable the eslint and typescript.
     + Yes to App Router
   * **Npm run dev** – to run the app
   * ****
   * Folder structure
     + ****
   * App is the main folder
     + It can contain folders and react component for nesting
     + Layout.js – this wrap around everything
       1. 
     + Page.js – the home page
   * Delete the app folder to reconstruct the layout.js and page.js to understand the file structure
2. Layout, homepage and Metadata
   * Create the app folder, layout.js and page.js
   * Layout.js and page.js are react components that are in the server
   * Create a folder named assets > styles > globals.css
     + Create the tailwind directives
     + Import in the layout.jsx >>> import ‘./assets/styling/global.css.
   * Create the meta-data in the layout.js
   * Add .env in .gitignore file
   * Upload in github
3. File base routing
   * No need to install react-router
   * Just create folder inside app and create components inside it
   * This is one of the best feature of SSR
   * Add folder inside the properties folder is a sample of nested routing
   * For dynamic route create a folder inside the properties and named it as [id]
   * […id], means catch all.
   * Import Link from ‘next/link’ – to be use for routing. Same as the Link in react-router dom
     + <Link href=’/’>Home</Link>
4. Server vs Client components & Router hooks

* When to use server or client component
* Server side component is default
* 
* Add ‘use client’ for client components
* If console.log is executed in a server component then look the output from the terminal.
* If console.log is executed in a client component then look the output from the browser.
* Import { useRouter } from ‘next/navigation’
* useRouter
  + - 
    - It brings it to the Home Page
* useParams – use to get the id from URL
  + - 
    - This will display the id in the button
    - Provided that there’s an id in th euRL
* useSearchParams
  + - 
    - Displays the name in the button provided that there’s a name=xxxx in the url
* usePathName
  + - 
    - Displays the pathname in the button from the URL

1. Navbar
   * Create the folder components outside app folder. This are for the components not the pages
   * Create the Navbar.jsx
     + Define the html
     + Put the images in the assets folder
   * Import Navbar.jsx in the layout.js
2. Navbar links, dropdowns and react icons
   * Import Link from ‘next/link’ to replace a tag with Link and set the href to ‘/’ and properties.htmkl to href=’properties’
   * For sign out changed the a tag to button
   * For login or logout button use react icons **npm I react-icons**.
   * For the mobile menu button (dropdown)
     + Create a useState (ismobileMenuOpen) to bring a toggle functionality to mobile dropdown. Set the navbar to ‘use client’ As this will now be a client not server component
     + Create a onCLick(() => setisMobileMenuOpen(true)). Use the isMobileMenuOpen state to toggle the appearance of the id=’mobile-menu’
     + Do it similarly with the id=user-menu’ create a useState isProfileMenuOpen to toggle the appearance of the profile menu
3. Active links and Conditional Rendering
   * Make the black box be set to the link being click
   * Make The Login or Register Button as well as the Add Property button to be conditionally rendered. It will only render if the user is logon
   * Use usePathname to get the pathname and use it as a state to determine have a conditional appearance of each button (Home, Property, Add Property)
   * Conditional render for Add Property, Profile Image and Message buttons if login or not
     + Create the isloggedIn state
     + If isloggedIn is false, do not display the Add Property, Profile Image and Message Buttons and vice versa
     + If isloggedIn is true, display the Login or Register button and vice versa
4. Homepage Components
   * Create the Hero, reusable Info boxes and footer
   * Create the Hero.jsx component and import it to the main page.jsx
   * Create the InfoBoxes.jsx Component
   * Create a. reusable InfoBox.jsx component
     + Make it dynamic by using the object created
     + Use this component to the InfoBoxes.jsx
   * Create the footer
5. Properties Page
   * Temporarily used the properties.json to display the properties details
   * Code the page.jsx inside the app/properties
   * Import properties from ‘@/properties.json’
   * Create a conditional rendering for the properties
6. Property Card Dynamic Data
   * Create the PropertyCard.jsx and import in the /app/properties/page.jsx
   * Pass the property={property} as props
   * Use the props in the PropertyCard.jsx and make the supply the dynamic data from {property}
   * Create a function (getRatesDisplay()) to display the rate mode (monthly, weekly, nightlhy)
7. Home Property Listing
   * Create the HomeProperties.jsx and import in the pages.jsx
   * Import json file
   * Create the recentProperties()
   * Import and implement the <PropertyCard />
8. Custom Not Found and Loading Pages (inside the /app)
   * Create the not-found.jsx. Take note of the letter casing
   * Create the loading.jsx
     + Can implement here the npm I react-spinners from <https://github.com/davidhu2000/react-spinners>
     + Set the component as a ‘use client’

Section 3 Database, API Routes and Property Components

1. Create MongoDb Database
   * Create a mongoDb. Skip if already have
   * Create the db (collection) for propertypulse
   * Get the connection string
     + Click Database/connect/drivers to get the environmental variables
     + Create the .env in the root of the app and paste the string
     + MONGODB\_URI=mongodb+srv://shernof:<password>@cluster0.ilwymnp.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
     + Input the correct password
   * Create the local server in the .env
     + NEXT\_PUBLIC\_DOMAIN=http://localhost:3000
     + NEXT\_PUBLIC\_API\_DOMAIN=http://localhost:3000/api
   * Check if the app is connected to db by console.log(process.env.MONGODB\_URI) in the HomePage.jsx
     + It should display the mongoDb value in the terminal of the vs code
2. MongoDb Compass
   * + Download the compass >>> <https://www.mongodb.com/products/tools/compass>
     + Install compass
     + At mongoDb atlas, click connect/compass
     + Copy the string and paste it to Compass
       1. 
3. DB connection and Mongoose
   * Create the config folder in the root and create the database.js
   * **Npm I mongodb mongoose**
   * Create the connection script in the db.js
   * 
4. The first API route
   * Import or export data from FE to BE and vice versa
   * Create the api/properties/route.js inside the app folder
   * Structure is same as the folder structure of the FE folders and files
5. Property and User Models
   * Create the models folder in the route
   * The ref:’User in Property.js means it is connected to User.js. while the ref: ‘Property’ means that it is connected to User.js
   * This is to connect the property to who is the owner and let the user only manipulate its own property.
6. Fetch data using server component
   * Create api routes to use mongoose to fetch data from db. Then in front end use the api routes to display the text and images from db
   * In /api/properties/route.js
     + Import Property from ‘Property.js’
     + Const properties = await Property.find({})
     + 
   * In /app/page.jsx, import the connectDb

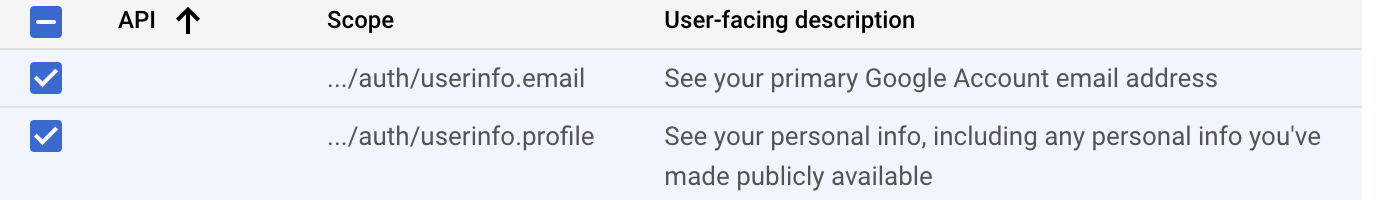
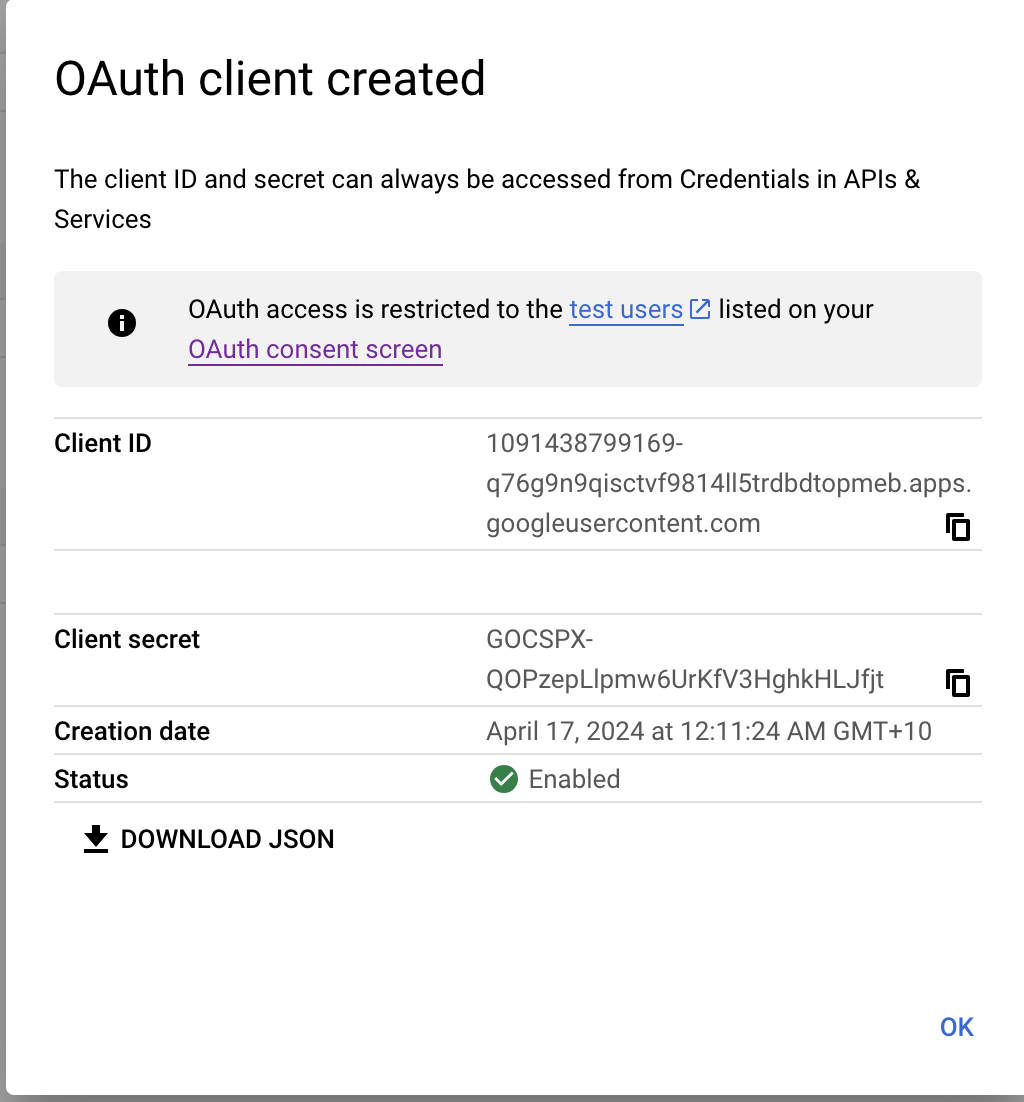
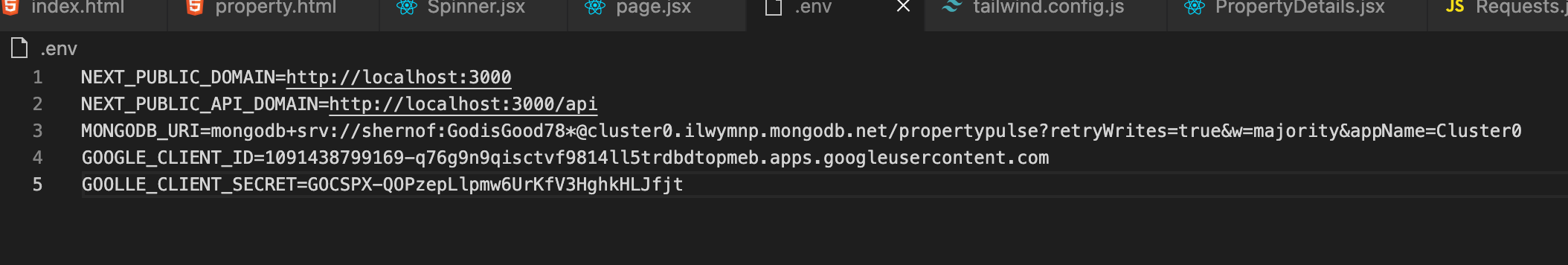
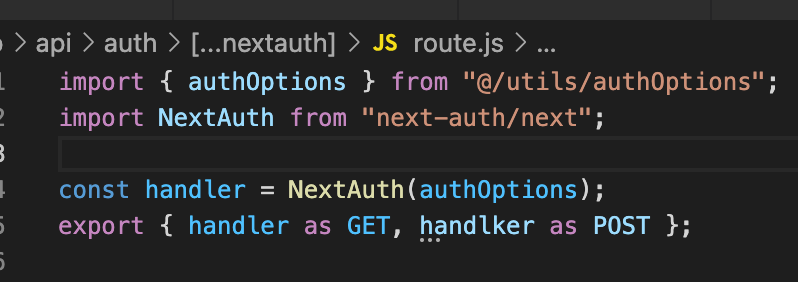
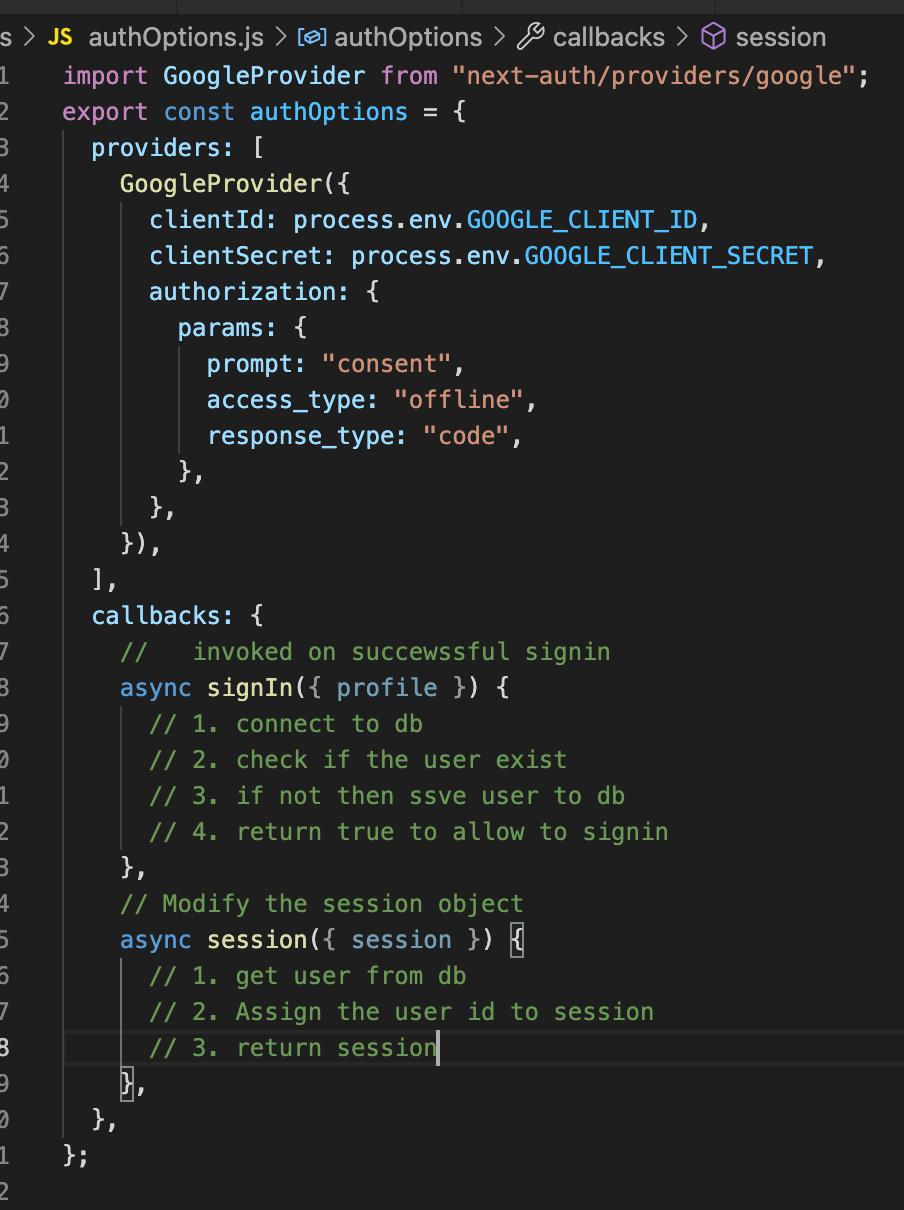
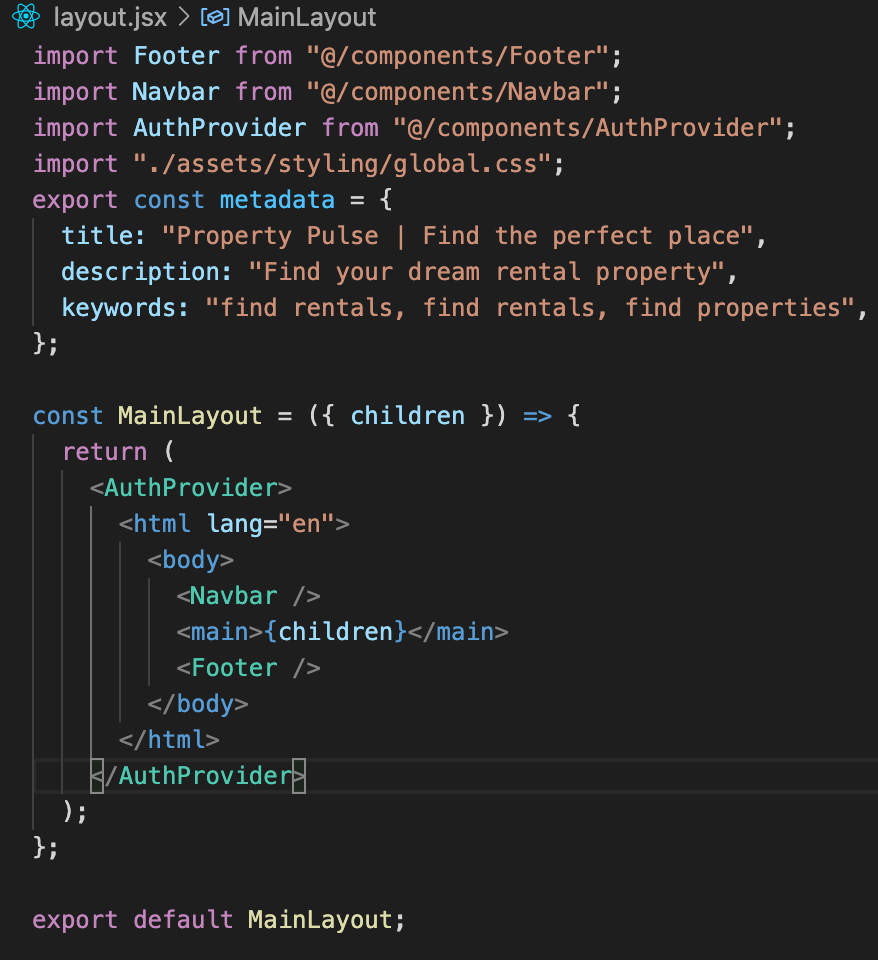
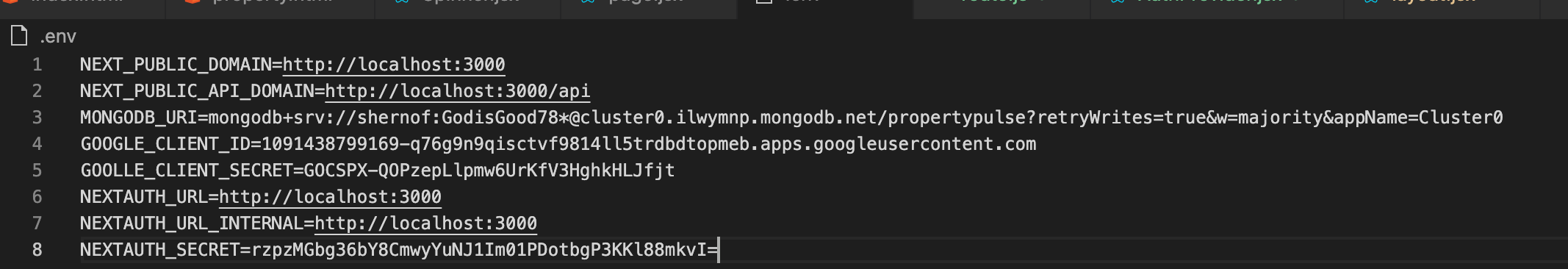
create an async function to fetch data from the server and not from properties.json anymore.

* + Run the fetchProperties() in the PropertiesPage = async () =>{

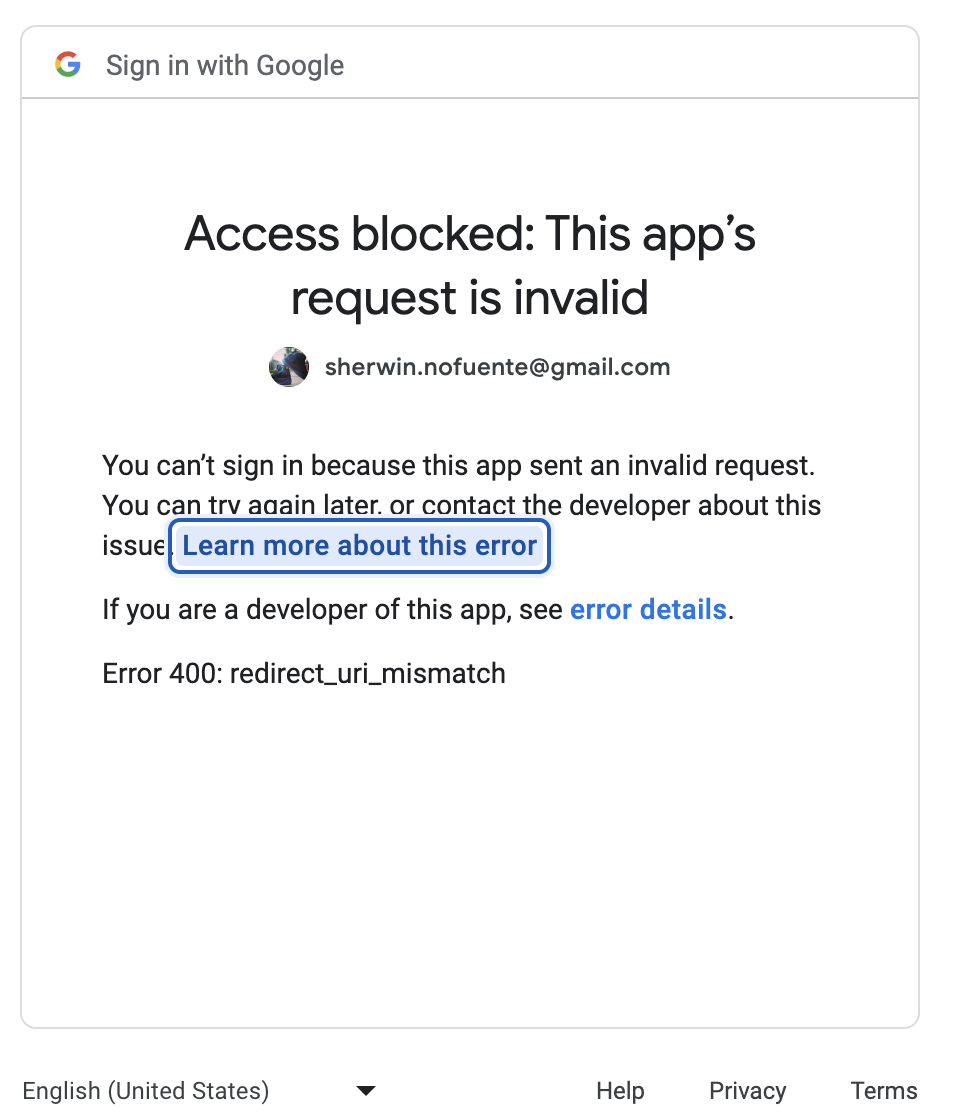
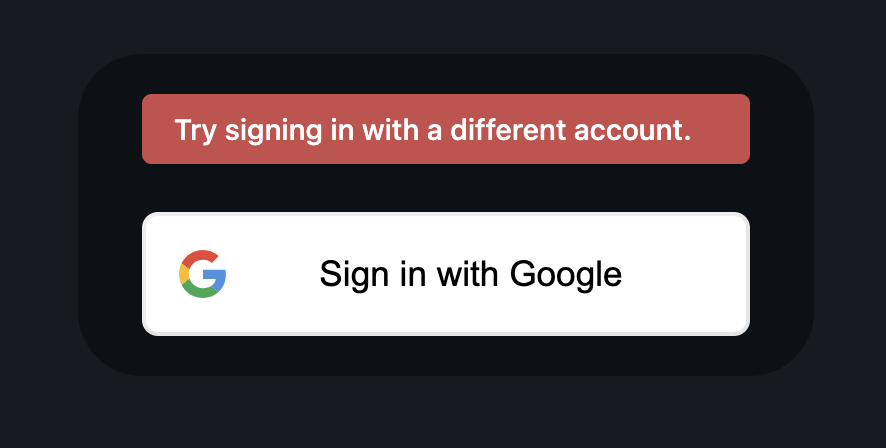
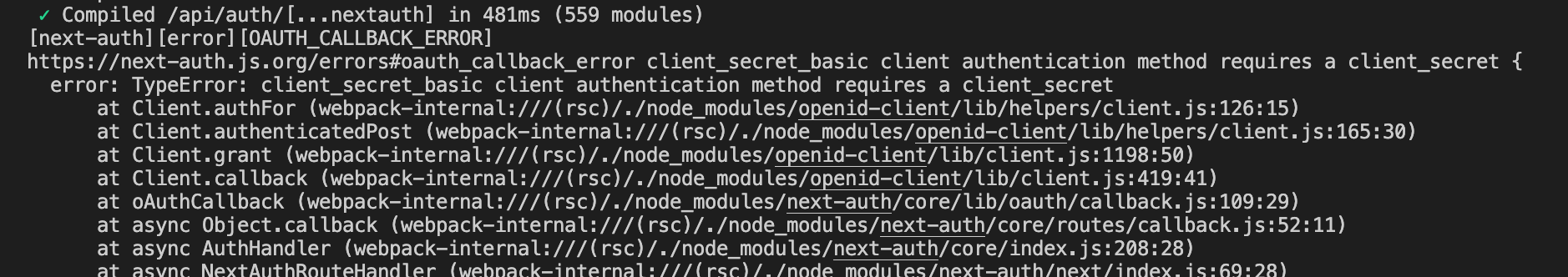
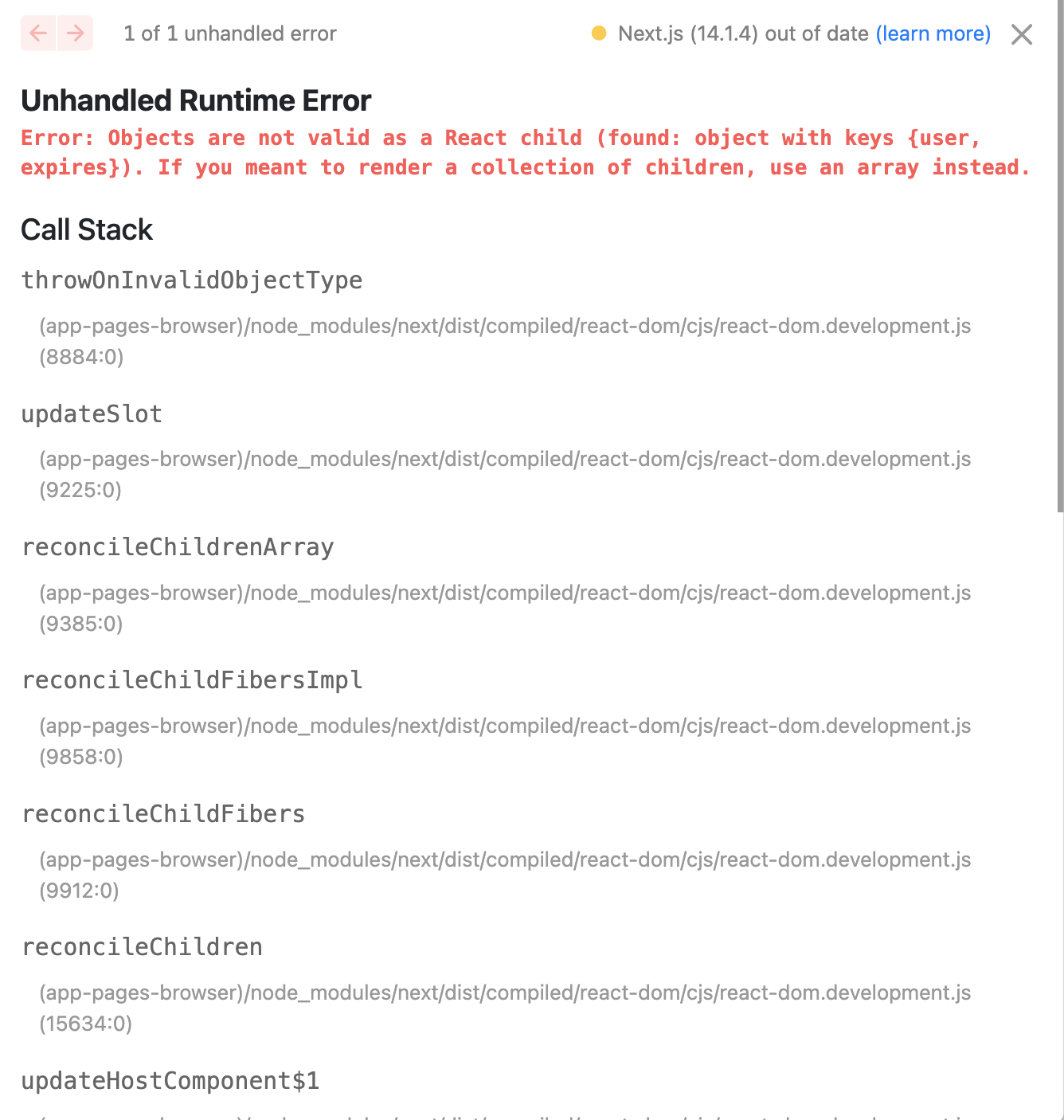
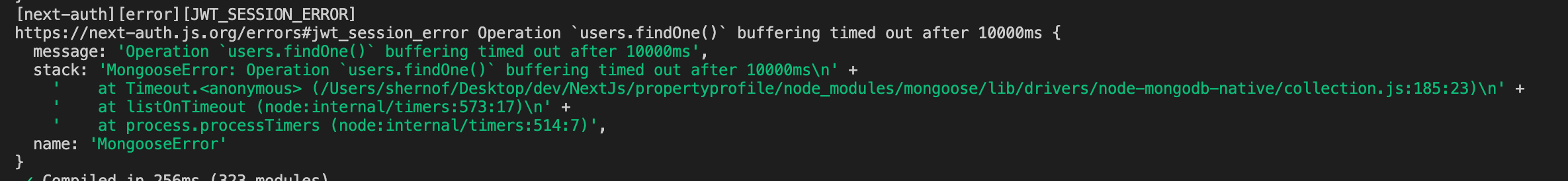
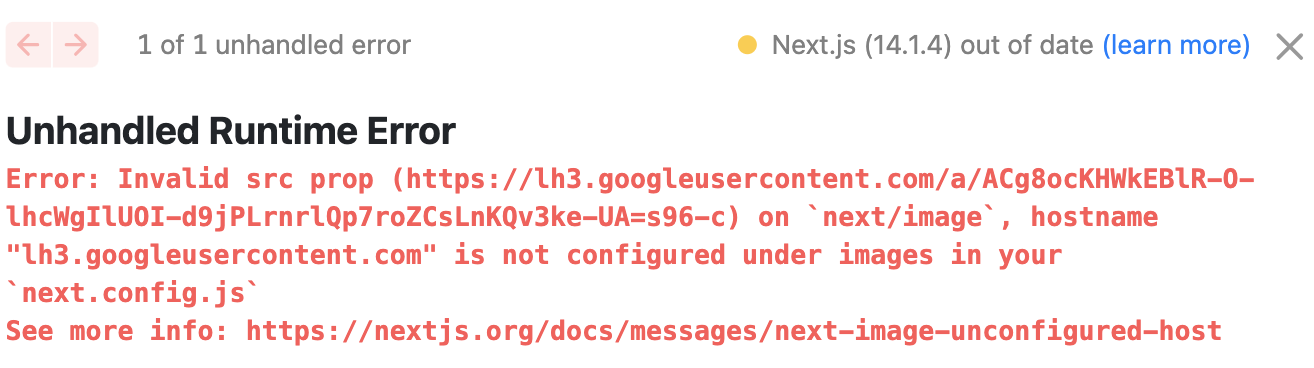
const properties = await fetchProperties()}

* + 

1. Requests File
   * This is useful for a JS that needs to be use repeatedly like the fetchProperties()
   * Create a utils folder in the root and create the Requests.js
   * Copy and paste the fetchProperties from page.js. export { fetchProperties }
   * Rempve the fetchProperties from page.jsx and HomeProperties.jsx
   * Import and implement { fetchProperties } to page.js and HomeProperties.jsx
   * Handle the domain if not yet available. This is by creating the apiDomain
   * A screen shot of a computer program

     Description automatically generated
2. Fetch a single property
   * This is for the Details button in the <PropertyCard />
   * Create the [id]/properties.js inside the api/properties forlder
     + Issue encountered. The appropriate json file did not display the information when the api run like ‘http://localhost:3000/api/properties/660ea249283012d5520acd3f’.
     + Cause of problem: created the /api/properties/[id]/routes.js is wrong. It should be /api/properties/[id]/route.js
   * Convert the server component /properties/[id]/page.jsx as it needs to store useParams, useState and useEffect
   * Get the id from the URL using import { useParams } from ‘next/navigation’
   * Store the fetch function in /api/properties/[id]/route.js to utils/requests.js as the single ‘property’ is also to be use in other components. Create the. fetchProperty()
   * Use the fetchProperty() in the /properties/[id]/page.jsx
3. Single Property Page
   * Initially create the hard coded HTML and split to individual component later
   * Create the /app/components/PropertyHeaderImage.jsx
   * Import /app/components/PropertyHeaderImage.jsx to /app/properties/[id]/page.jsx
   * Create the Go Back button
   * Create the html for Property info
   * Customize a 70/30 display split for the sidebar to be visible in tailwind.config
4. Property Details Component
   * Replace the hardcoded data with dynamic data from db
   * Create the /app/components/PropertyDetails.jsx. this is the left side of the property info
   * Wrap the Rates with conditional
5. Create a Spinner component for the ‘use client’ component
   * Create a Spinner.jsx in /app/cpmponents/Spinner.jsx
   * Copy the loading.jsx code and put it in Spinner.jsx
   * Implement the { loading && <Spinner loading={loading}/> } in /app/cpmponents/properties/[id]/page.jsx
6. Google Oath Setup
   * Requires API key, client id, google client secret
   * Create a new project in <https://console.cloud.google.com/welcome?pli=1&project=equipment-tracker-332712>
   * Click API and Services. Click credentials, click ‘Create Credentials’ and select Oath client ID
   * Click configure consent screen
   * Click create and give an app name. Supply the developer contact info
   * Click ‘Add or Remove Scope’ and select the following:
   * 
   * Click update, click save and continue
   * Enter the email addresses for the test user account. More than 1 account is better it will able to test the account. Save and continue
   * Click back to dashboard
   * Click Credentials
   * Click + Create Credentials and create a Oath Client Id. Application type = Web Application
   * Click Add URI and enter <http://localhost:3000>
   * Click Authorized redirect URI. Enter <http://localhost:3000/auth/callback/google>
   * Click Create. Client Secret and client id
   * 
   * Copy the client id and paste it to .env
   * 
7. NextAuth and Provider Setup
   * Create a provider array and implement the google provider
   * Go to <https://next-auth.js.org/providers/google>
   * **npm I next-auth**
   * Create /app/api/auth folder
   * Create /app/api/auth folder/[…nextauth]/route.js
   * 
   * Create the authOptions() in utils/authOptions.js
   * 
   * Import the google provider
8. Session wrapper provider
   * Bring in the SessionProvider and wrap the app by it.
   * Create app/components/AuthProider.jsx
   * 
   * Wrap the layout.jsx by the AuthProvider
   * 
   * Add the following environmental variables in .env
   * NEXTAUTH\_URL = When deploying to production, set the environment variable to the canonical URL of your site.
   * NEXTAUTH\_SECRET = Used to encrypt the NextAuth.js JWT, and to hash [**email verification tokens**](https://authjs.dev/getting-started/adapters#verification-token). This is the default value for the option in [**NextAuth**](https://next-auth.js.org/configuration/options#secret) and [**Middleware**](https://next-auth.js.org/configuration/nextjs#secret).
     + Run Openssl rand -base64 32 at the terminal
     + Copy thegenerated hash and paste it as a value for NEXTAUTH\_SECRET
   * NEXTAUTH\_INTERNAL\_URL = If provided, server-side calls will use this instead of NEXTAUTH\_URL. Useful in environments when the server doesn't have access to the canonical URL of your site. Defaults to NEXTAUTH\_URL.
   * at <https://next-auth.js.org/configuration/options>
   * 
9. Sign in button
   * This feature will enable to signin using google account
   * Import { signIn, signOut, useSession, getProviders } from ‘next-auth/react at Navbar.jsx
   * Create the const { data: session } = useSession() and changed all the isLoggedIn state to session state
   * Create the const [ providers, setProviders ] = useState(false)
   * Use the useEfffect to get the google authentication using the getProviders
   * A computer screen shot of blue letters

     Description automatically generated
   * Configure the buttons
     + A screen shot of a computer code

       Description automatically generated
10. Save user to database and session
    * Operation is in the login button of the Navbar
    * Import { signIn, signOut, useSession, getProviders } from ‘next-auth/react’
    * Initialize the session and use it to replace the isloggedIn state
      + Const { data: session } = useSession()
    * Use the useEffect so that it when the page load it will setAuthProviders by using the getProviders()
      + Create a providers state in this useEffect
    * With the providers state, the signIn and signOut state cn now be use. These states can’t be use without the providers
    * Use the providers in the login buttons and use the onClick={signIn(providers)}
    * Features
      + Check if the user is in db if not then add the user
      + Assign a user id from db to session
      + In utils/authOptions
    * Error encountered
    * 
    * Found out that the Authorized redirect URI = <http://localhost:3000/auth/callback/google>. It should be <http://localhost:3000/api/auth/callback/google>
    * It fixed the issue
    * 
11. Save User to Database and Session
    * Features
      + Check if the user is in the db
      + Add the user if not
      + Import connectDb and import User model
      + Complete the functionalities defined in authOptions.js
      + 
      + Encountered the following errors
      + 
      + 
      + Found out that the GOOGLE\_CLIENT\_SECRET was misspelled as GOOLLE\_CLIENT\_SECRET
      + Error encountered
      + 
      + 
12. Profile Image
    * Create const profileImage = session?.user?.image
    * Use this as src={profileImage} of the profile
    * Encountered error
    * 
    * In nextjs.config.mjs
    * 
13. Signout and Protect routes
    * Add onClick={()=> setIsProfileMenuOpen(false )signOut()}
    * Protect the route. This is to prevent user to go to <http://localhost:3000/properties/add> if he/she is logout
    * create middleware.js in the root
    * 

Section 5. Create Properties and Cloudinary Integration

1. Create the Add Propery page app/properties/add/page.jsx
   * Separate the form by creating the client component /cmponents/PropertyAddForm.jsx
   * Create the fields state which contains he information of the property
   * Create the const handleChange = () => {} and use it in the radio buttons of the Property Type
   * Create the const handleAmenitiesChange = () => {} and use it in the checked buttons of the Amenities Type
     + Learned how to use state to define the checked buttons
   * Create the const handleImageChange = () => {} and use it in the input for the images
2. Form Input Handlers
   * Add functionality to the 3 handlers
     + Const handleChange
     + Learned how to handle the state with nested property
     + Const handleAmenitiesChange
     + Const handleImageChange()
     + This is uploading the image
3. Add Property POST API
   * Create action=’/api/properties’ method=’POST’ encType=’multipart/form-data’ in the form of the PropertyAddForm.jsx
   * Then create the /api/properties route in api/properties/routes.js