| **BLOG-BYTE Progress File-4** |
| --- |

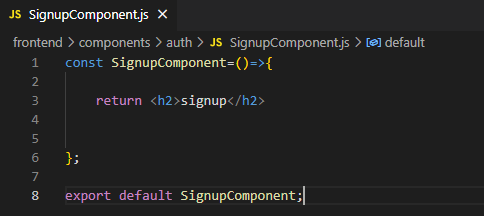
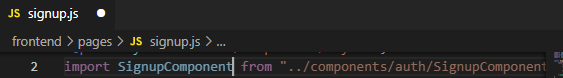
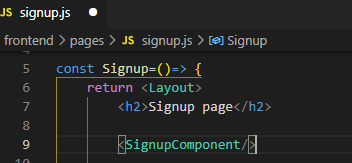
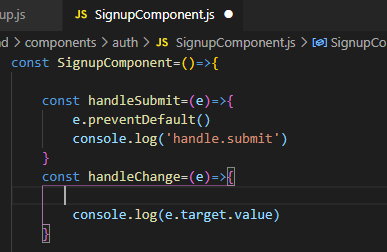
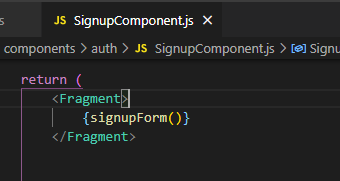
# **Frontend Sign-in Sign-up:**

1. **Signup Form**
2. **Signup Action & Alerts**
3. **Sign In**
4. **Saving user and token in cookie and localstorage**
5. **Signout and conditionally showing nav links**
6. **Redirect authenticated user from signin and signup page**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# **TASK-1 Signup Form**

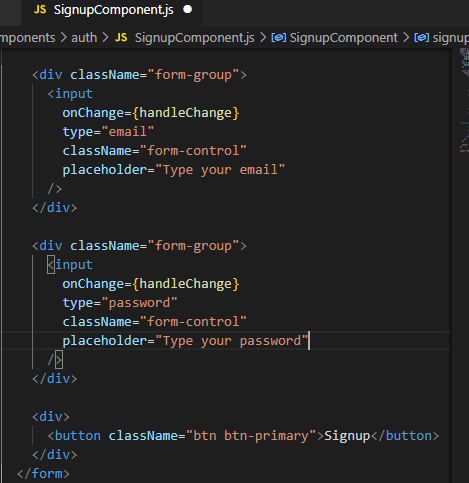
**STEPS :**

* In **frontend** folder, go to **components**, and create a new folder named **auth** to keep all authenticated components inside.
* Inside **auth** create a new component file named **SignupComponent.js.**
* 
* Create SignupComponent. And import it in signup page in **pages** folder,.
* 
* Use SignupComponent here instead of Link.
* 
* Now creating a Signup form inside SignupComponent.
* 
* Creating required functions.
* 
* **e.preventDefault** => to avoid page reloading (e: event).
* Changing the return value to react fragment.
* 
* Start the server and see the output, we have a signup form now.

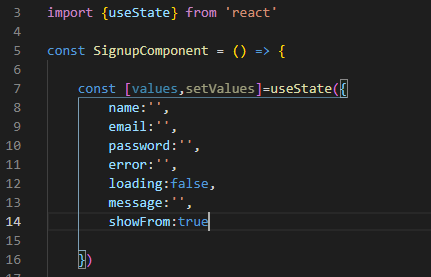
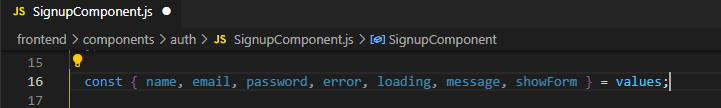
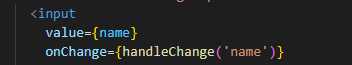
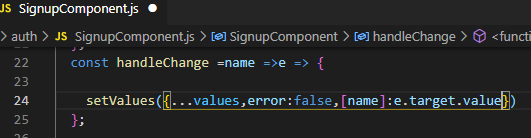
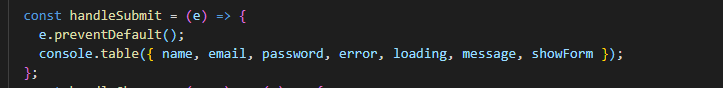
|  |
| --- |

* Anything we type in input field will be console logged.

|  |
| --- |

* Continuing with our form fields.Adding email, password and submit button for signup.
* 

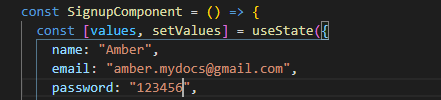
|  |
| --- |

* Adding useState to grab onChange values from user to backend
* 
* Destructuring values,so we can use them directly.
* 
* Adding values in input fields and handleChange method.
* 
* Adding implementation in handleChange method (function returning another function).
* 
* To see check and information when user clicks submit button, lets pass it to handleSubmit.
* 
* Start the server and check form submission results.

|  |
| --- |

* Console displaying all input field values after pressing submit.

|  |
| --- |

* To avoid typing values again and again during development, just passing values by default for the time being.
* 

**=>Signup from is working, Task Completed… :)**

# **TASK-2 Signup Action & Alerts**

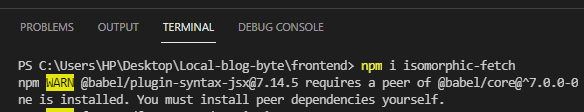
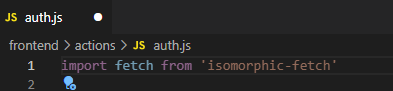
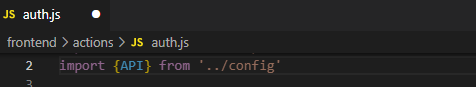
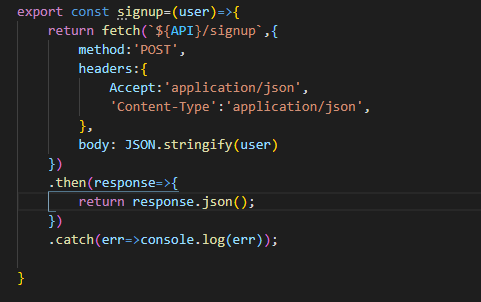
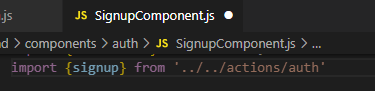
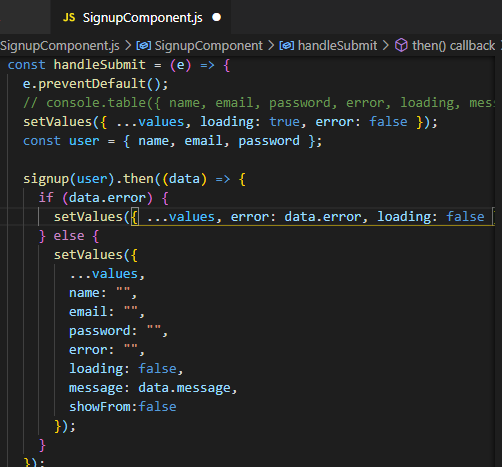
**STEPS :**

* Now we need to send signup information to backend.
* For this we need a **http client** and for this we are going to use **fetch**.

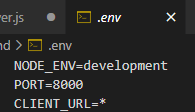
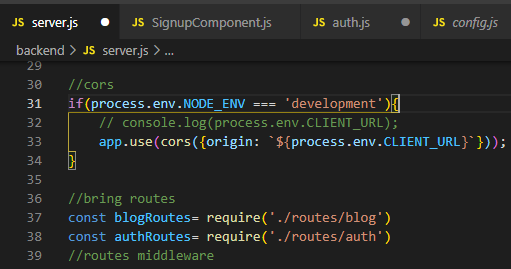
**Fetch API** provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses.

* To install it open frontend folder in terminal and run command

**npm i isomorphic-fetch** *(isomorphic fetch: works both in client and server side)*

* 
* Installation complete.
* Now inside **frontend** , create a new folder called **actions**. Create a file named **auth.js** inside it. All the auth methods will be written inside it.
* Import fetch inside it.
* 
* Also being API inside it.
* 
* Now we can write our signup method inside it . Returning fetch with URL and user information.
* 
* Once we pass user information to this method, it will make request to backend and create a new user.
* Now using this signup action inside **SignupComponent.js**.
* 
* Using it in handleSubmit method.
* 
* If theres any error display it otherwise send data to backend and set field to empty and disable submit button once user is created.

**CORS Error**

* In **.env** file adding \* in CLIENT\_URL to allow access.
* 
* To resolve it some changes in **server.js** file.
* Adding cors before routes.
* 
* Now sending request from client to server.

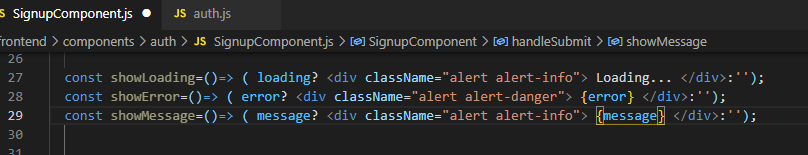
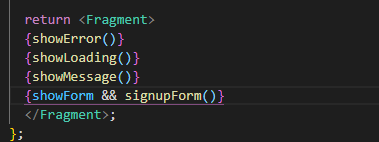
|  |
| --- |

* Now sending request by new email.

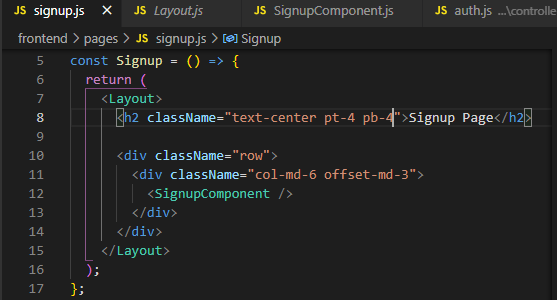
|  |
| --- |

* Data successfully added in mongo-atlas collections.

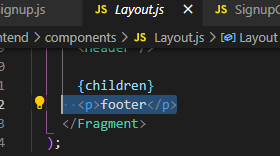
|  |
| --- |

* Adding functions in **SignUpComponent.js** file.
* 
* Assigning functions.
* 
* Messages displayed accordingly on signup.

|  |
| --- |

* Go to **signup.js** page and put it inside container and add some classes.
* 
* Signup component looks like this now.

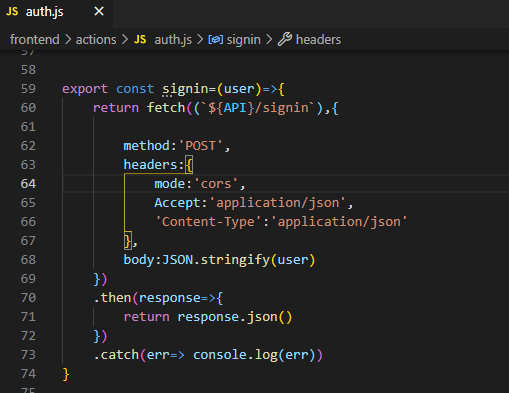
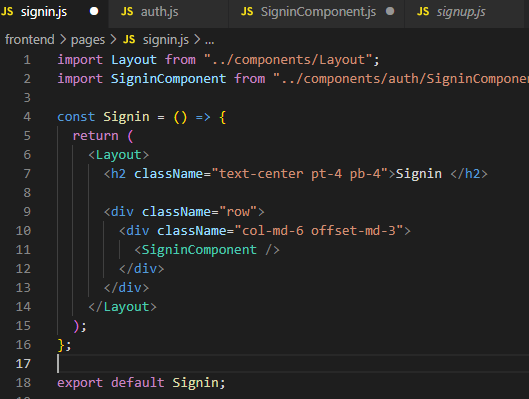
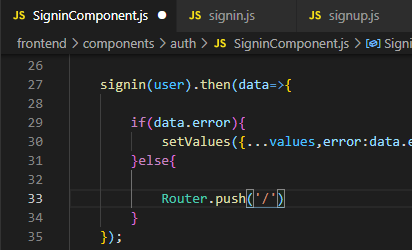
|  |
| --- |

* Remove footer from layout.
* 

**=>Signup Action & Alerts is working, Task Completed… :)**

# **TASK-3 Sign In**

**STEPS :**

* In **actions** folder, go to **auth.js** file and add signup method there.
* 
* Go to **pages** folder and add **signin.js**.
* 
* Go to **components** => **auth** folder and create **SigninComponent.js**.
* Creating Sign In form same as Signup, with email and password fields.
* On Sign In, we need to save user information.(in next task)
* Redirecting user to home on sign in.
* 
* Now try Signin.

|  |
| --- |

* We will be redirected to home page.

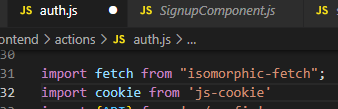
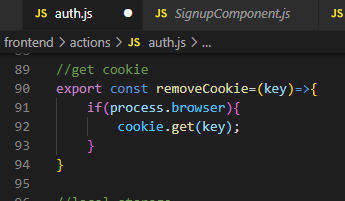
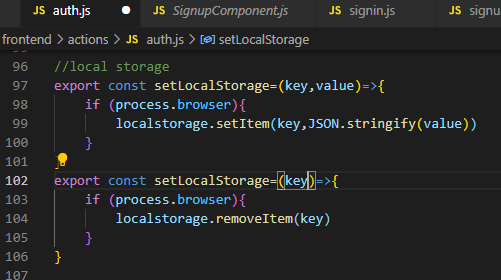
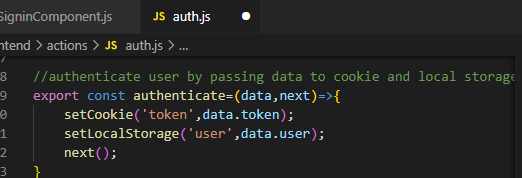
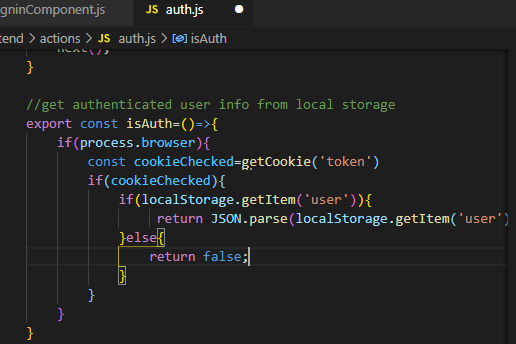
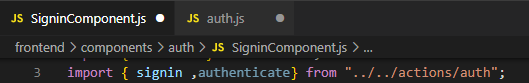
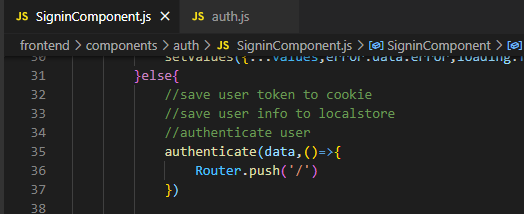
| **=> Response in Network** |
| --- |

**=>Sign In created, Task Completed… :)**

# **TASK-4**

# **Saving user and token in cookie and localstorage**

**STEPS :**

* Writing some helper methods that will allow us to save token in cookie , user information in local storage.
* To use cookies install it in **frontend** folder, using command **npm i js-cookie**.
* In **actions** go to **auth.js** file and create methods there.
* Import cookie.
* 
* Setting cookie that will expire in one day. (**key**=> name passed to cookie)
* 
* Get cookie.
* 
* Updating local storage.
* 
* Middleware functioning. Authenticate method will receive data,save that cookie in local storage and execute next().
* 
* Getting authenticated users from local storage
* 
* In **components auth folder** go to **SigninComponent.js** file and update methods there.
* Import authenticate method here.
* 
* Redirect in authenticate function.
* 
* Now give it a try, run the server [**http://localhost:3000/**](http://localhost:3000/)**.**
* Inspect go to **application tab** (You can go to this tab and see every cookie that a page has.)
* No sign in yet, so no user in local storage. And no cookie names token.

| **Local storage** |
| --- |
| **Cookie** |

* Now sign in and then see local storage, user will be created. And in cookies token will be generated.

| **Local storage** |
| --- |
| **Cookies** |

* To manually delete token.

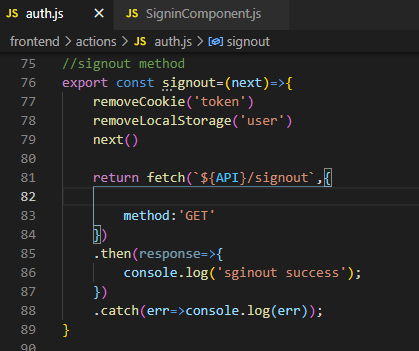
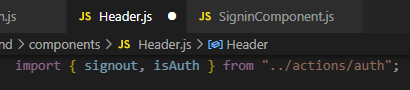
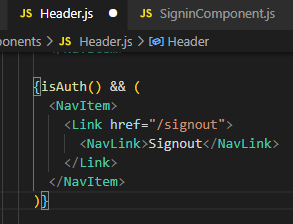
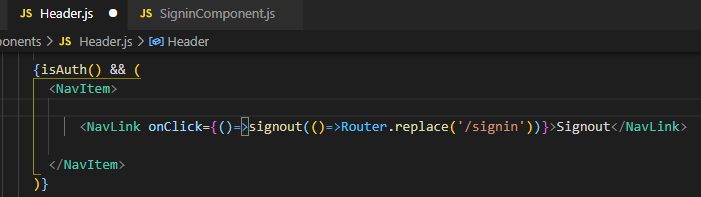
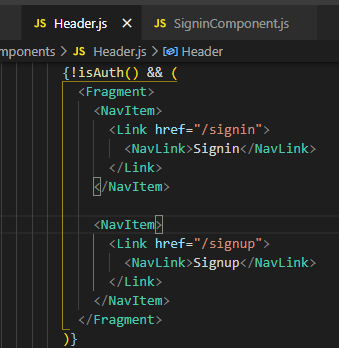
|  |
| --- |

**=>User and token saved successfully, Task Completed… :)**

# **TASK-5**

# **Signout and conditionally showing nav links**

**STEPS :**

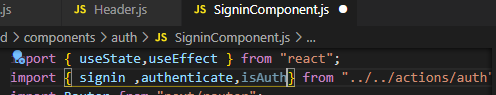
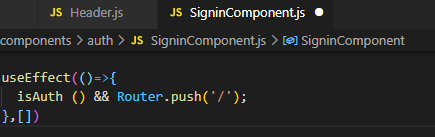
* In **frontend** end **actions** **auth.js** file creating signout method.
* Removing local storage and cookies from browser, then sending request to backend server.
* 
* Showing signout link only if the user has signed in.
* In **components** go to **Header.js** and import functions here.
* 
* Checking if user authenticated.
* 
* Redirecting user to signin page on sign out.
* 
* Now run the server [**http://localhost:3000/**](http://localhost:3000/) (user present in local storage & token present in cookies) click signout, it will redirect to signin page (user removed from local storage & token removed from cookies).
* Adding changes in signin link. Show sigin and signup only if user is not authenticated.
* 
* Now we will only see signout after signin.

**=>Signout and conditionally showing nav links, Task Completed… :)**

# **TASK-6**

# **Redirect authenticated user from signin and signup page**

**STEPS :**

* In **components auth folder** go to **SigninComponent.js** file and import useEffect hook (life cycle method).Also bring isAuth method.
* 
* Adding usestate method, so everytime there is a change in state it will run automatically.
* If user authenticated, it will be redirected to home page.
* 
* Now try going to signin page manually, it will be redirected to home page.
* 

|  |  |
| --- | --- |

* Do the same for signup page.

**=>Redirect authenticated user from signin and signup page, Task Completed… :)**

# 