| **Progress File-2** |
| --- |

# **User Sign-in Sign-up**

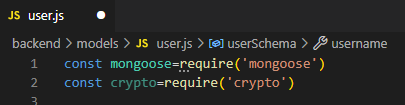
1. **User Model and Schema**
2. **Sign-up Route and Controller setup**
3. **Installing npm packages from Auth and Blog**
4. **Sign-up Validator**
5. **User schema virtual fields and methods**
6. **User Sign-up**
7. **Using Mongo-atlas**
8. **User Sign-in**
9. **Sign-out and protected Routes**

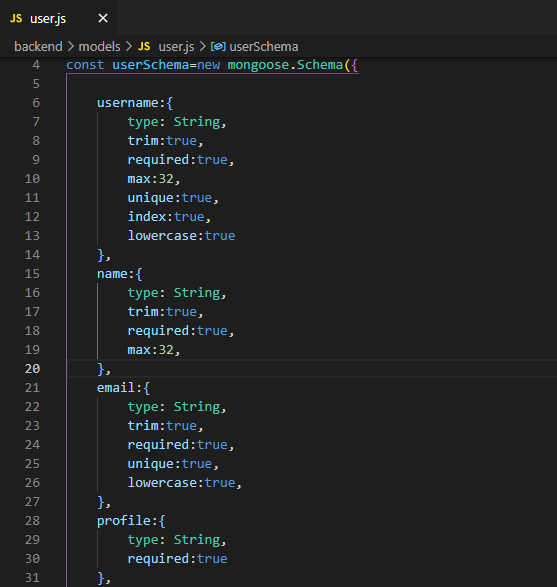
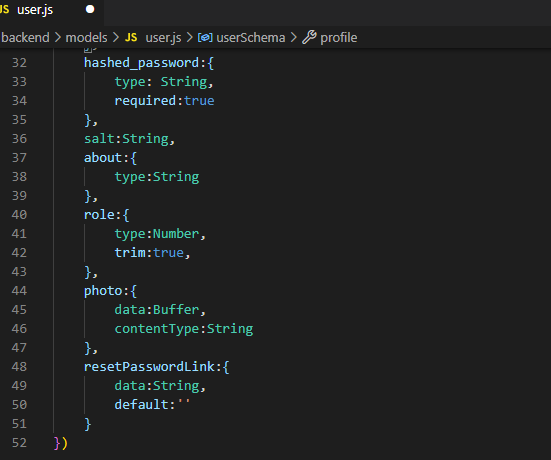
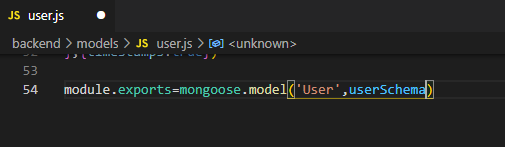
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# **TASK-1 User Model and Schema**

**STEPS :**

* Building authentication system, so we can start registering the users.The users login so they can create a new blog and so on.
* First we need to create a **user model**. User model will have a schema that will tell what sort of fields will be there to create new users.
* In **backend** folder create a new folder named **models**. Create a new file named **user.js** in it.
* Start by bringing **Mongoose** because we are going to create mongoose schema.
* We also need to bring **crypto** which is core nodeJs module and we are going to use it to has password later.



* Now lets create a user schema (mongoose.Schema: function thats takes object as argument). Write properties inside it.
* 
* Salt to define how strong we are going to have our password.It will have some numeric value.
* Role for role based authentication system.
* Photo type buffer, it will be saved in binary data format in database.(MongoDb is perfect for saving binary data).
* 
* Add timestamp, so we don't need to create it ourselves manually. It will automatically add and update date in database.
* 
* Indicate user and userSchema
* 
* So this is our mongoose schema for user model . Now we can use this user model to create a new user and save in the database.

**=>User Model & Schema have been setup successfully,**

**Task Completed… :)**

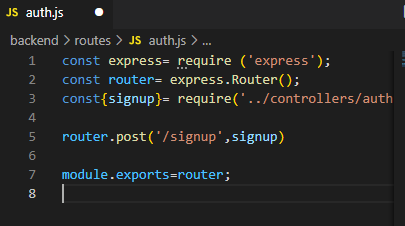
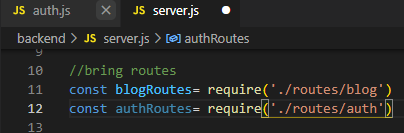
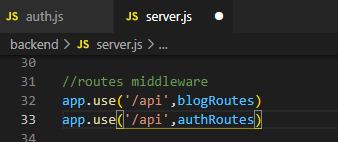
# **TASK-2**

# **Signup Route & Controller Setup**

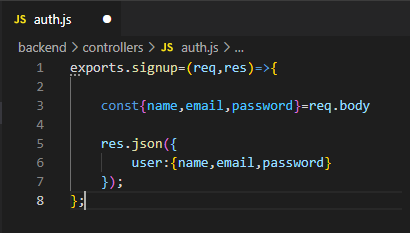
**STEPS :**

* To create a new user , we need to setup routes and controls.

**ROUTES**

* Inside **backend** folder, got to **routes** folder and create a new file names **auth.js**.
* Handle **signup** request here and create signup method in controller.
* Now bring it in **server.js** file, like we did for blog routes.
* 
* We can use this as middleware.
* 

**CONTROLLERS:**

* Now creating auth controllers.
* Inside **backend** folder, got to **controllers** folder and create a new file names **auth.js**.
* We need to create signup method here.
* To create a new user, we need username, email, password and we will get that information from request body.
* So destructure this information form request body.
* 
* To check it, lets respond with same information and send it back to user.
* 
* Go to postman and make a **post** request. And URL will be **http://localhost:8000/api/signup**

|  |
| --- |

* Now we need to go to body, select **raw,** and select text as **JASON(application/jason).**

|  |
| --- |

* Now we can write JASON data here.

|  |
| --- |

* Go to Headers and make sure that content type is application.json.

|  |
| --- |

* We can send this information to backend to create new user.
* Now press send. And we will get json response with same information that we send to backend.
* Now rather than getting this information back, we need to save it to our database.

**=>Signup Route & Controller have been setup successfully,**

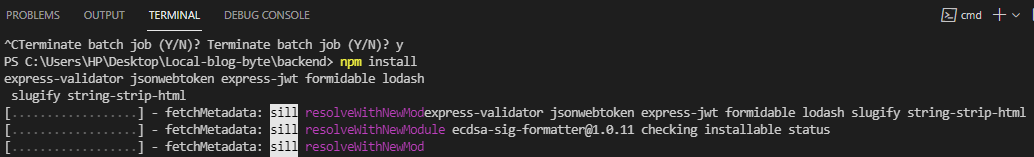
**Task Completed… :)**

# 

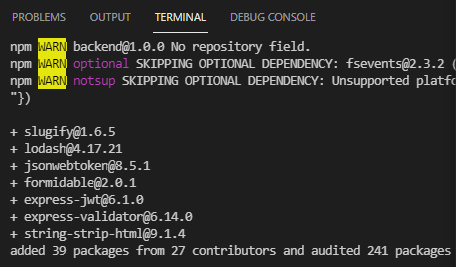
# **TASK-3**

# **Installing npm packages from Auth and Blog**

**STEPS :**

* Before saving users in database, we need to do some checks for validation.
* For this we need to install some npm packages.
* **npm install express-validator jsonwebtoken express-jwt formidable lodash slugify string-strip-html**
* ****
* **Express-validator:** for validation
* **Jsonwebtoken:** to generate json web token so that we can send that to our client. This will allow us to implement authentication.
* **Express-jwt:** This will take if generates token is still valid or not.
* **Formidable:** This will allow us to receive form data (because we will be receiving form data from client side to server side). Different than json type, so we need formidable for it.
* **Lodash:** To use some helper methods from lodash library.
* **Slugify:** We need to create slug for each blogpost that we save in the database.And we will query the database based on that slug not by blog id. We are doing it for SEO purpose.

|  |
| --- |

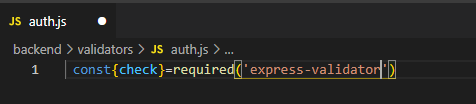
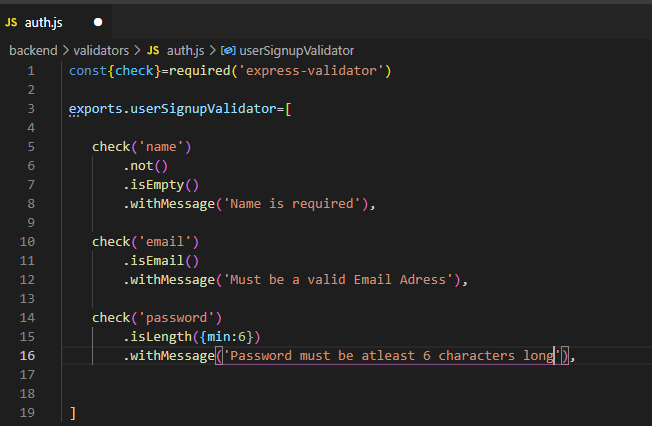
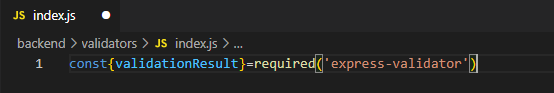
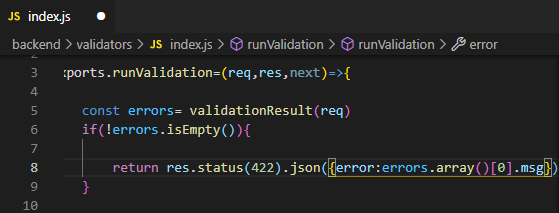
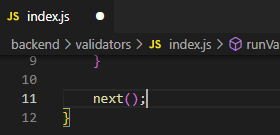
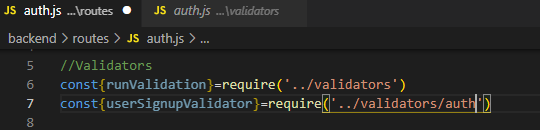
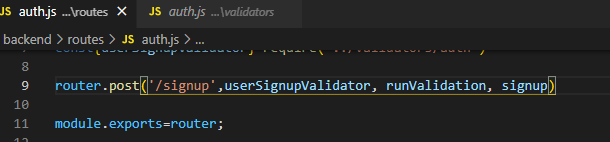
* **String-strip-html:** When we create a new blog, starting few lines can be used to generate meta description.(important for SEO)
* 
* Installation complete.

**=>npm packages from Auth and Blog successfully Installed,**

**Task Completed… :)**

# **TASK-4 Sign-up Validator**

**STEPS :**

* To implement user signup, we need to validate user data.
* In **backend** folder create a new folder named **validators**. Create a new file named **index.js** in it.
* In **validators** folder create another new file named **auth.js** in it.
* First bring in check form express validator.
* 
* Now we can write validation checks for name,email and password.
* 
* If there would be any errors during validation, we will get them in **index.js** file.
* 
* Because this is middleware so we need request, response and next callback function.
* 
* As soon as we get any error, we send that very first error msg with status code of **422** which is for **unprocessable entity**.
* Execute this callback so that our application doesn't get halt.
* 
* Now we can use this in routes. In **routes** folder go to **auth.js** file and add validators here.
* 
* Now we have validators, so we can apply it in middlewares.If validation is passes then code in signup controller method will be executed.
* 
* Now try making a request without name to check. Error will be displayed in response.

|  |
| --- |

* If email is invalid.

|  |
| --- |

* If password is less than 6 characters.

|  |
| --- |

* So our validation is working perfectly.

**=>Signup validation has been successfully implemented,**

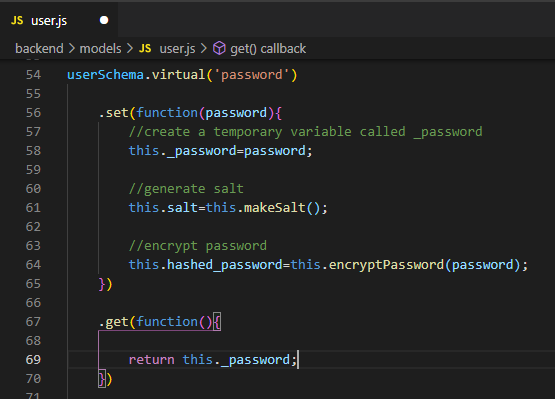
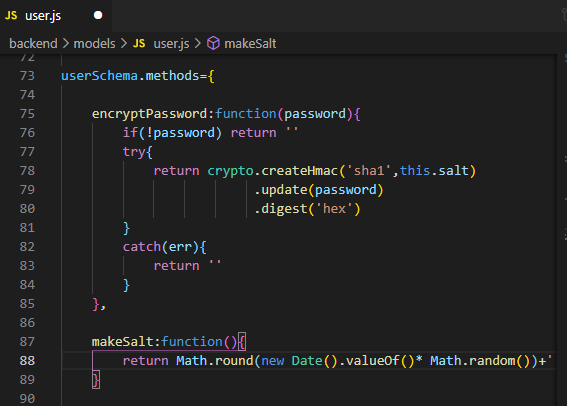
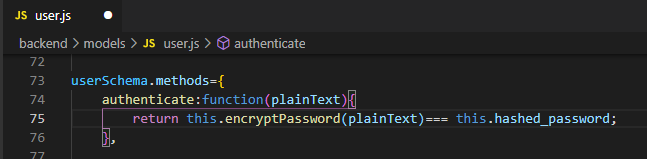
**Task Completed… :)**

# 

# **TASK-5**

# **User schema virtual fields and methods**

**STEPS :**

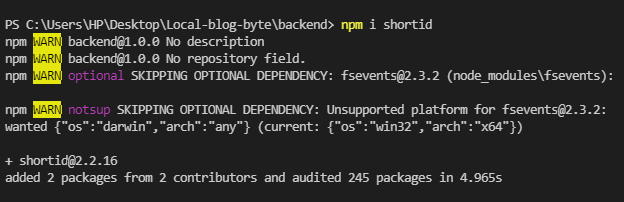
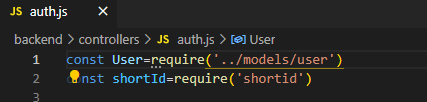
* Now we are going to implement password and hashing of password.
* In **backend** folder go to **models** folder. Open **user.js** and implement here.
* We can add **virtual fields** here.Getting password from user and saving it as hashed password in database.
* Setting password as virtual field.
* 
* Now creating required functions for userSchema.
* 
* Encrypt password will hash the password using creatHmac() Hashing function.
* Make salt() will generate a random value for hashing purpose.
* 
* Authenticate to compare the password. It will encrypt the plain password and compare it with hashed version.If they match, we can authenticate users.

**=>User schema virtual fields & methods have been successfully implemented,**

**Task Completed… :)**

# **TASK-6 User Sign-up**

**STEPS :**

* Our users will have username, that will be automatically generated when they signup for first time.
* To generate unique usernames we have to use a package called **shortid**. It can be installed using **npm i sharpie.**
* 
* Package is installed.
* Now in **controllers** folder, go to **auth.js** file and import in it.
* 
* Now to save the user, first we need to check if user already exists.
* FindOne() method in mongoose to find a particular user from database.
* 
* If email already taken it will display error msg and if user does not already exist, it will create new user.
* Lets try it in our Postman for testing.

| **Success msg** |
| --- |

* User has been created with unique username,hashed password and profileURL.

|  |
| --- |

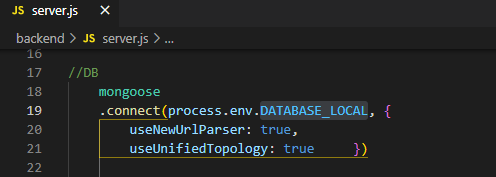
* Error in case user already exist.

**=>User Signup successful, Task Completed… :)**

# 

# **TASK-7 Using Mongo-atlas**

**STEPS :**

* To use mongo-atlas, go to backend folder, open server.js file and replace **DATABASE\_LOCAL** with **DATABASE\_CLOUD**.
* 
* From postman send a request.

|  |
| --- |

* Open the link of your mongo-atlas connection.
* <https://cloud.mongodb.com/v2/6173b687f1f1fe3e412d4988#clusters?fastPoll=true>
* Click on browse collections to view the data send to database.

|  |
| --- |

* Here is the data stored in database.

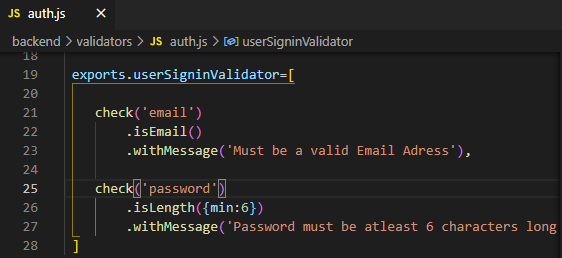
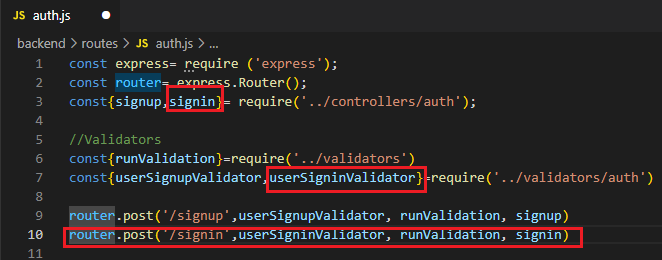
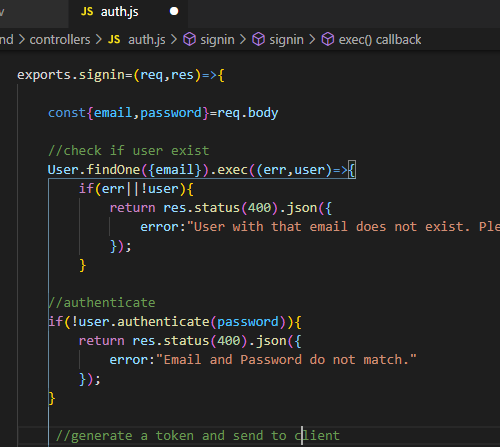
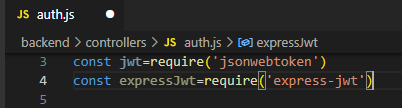
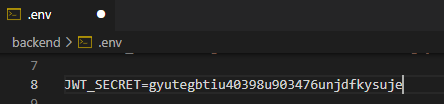
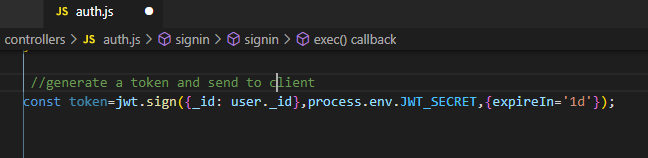
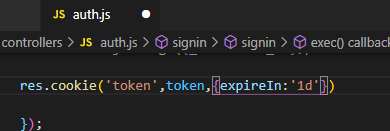
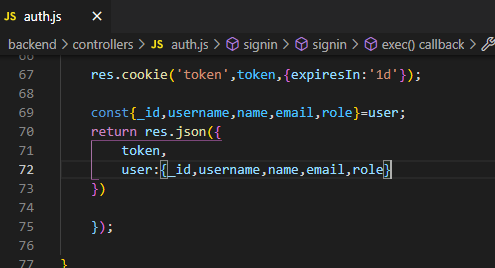
|  |
| --- |

**=>Data has been successfully stored in Mongodb-Atlas,**

**Task Completed… :)**

# **TASK-8 User Sign-in**

**STEPS :**

* After signup, now we have to implement user sign-in.
* In **backend** folder, go to **validators** folder and open **auth.js** file.
* Now add sign-in validator in it. For sign-in we require only email and password.
* 
* In **backend** folder, go to **routes** folder and open **auth.js** file. And create routes for sign in here.
* 
* Now go to **controllers** folder and open **auth.js** file. And add the signin method here.
* First check if user exist. Then authenticate email and password. And after this we will generate a json web token that will include userId.
* 
* Bring in packages for json web token.
* 
* Before we generate token, we need to create a secret key(some random words that people can’t guess). For that go to **.env** file and ass value in **JWT\_SECRET**.
* 
* Now creating signed token using userId, jwt\_secret and expiry date.
* 
* No sending this token to cookie as well.
* 
* Sending response to user.
* 
* Now lets try sign-in on postman.Give URL **http://localhost:8000/api/signup** and send post request.

|  |
| --- |

|  |
| --- |

* If email is not registred, it will show the above error.

|  |
| --- |

* If email is registered but password is not correct, above error will be displayed.

|  |
| --- |

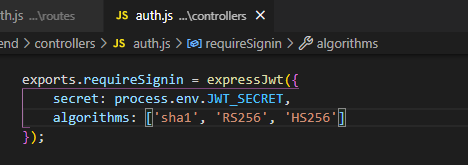
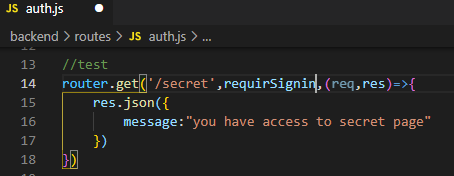
* If email is registered and password is correct, user will be responded wil user info and token(used when we need to access protected routes.).

**=>User Sign-in Successfully, Task Completed… :)**

# 

# **TASK-9 Signout and protected routes**

**STEPS :**

* To implement sign-out. In **backend** folder, go to **controller** folder and open **auth.js** file.
* 
* Create a signin middleware.It will check incoming token secret and compare with that we have in our **.env** file. If it will match and is not expired than user will have access to it.
* 
* In **routes** folder and open **auth.js** file. Now we have to create route for signout.
* 
* To check only login users have access to a certain page.
* 
* Checking signout,go to postman and give URL **http://localhost:8000/api/signup** and send get request.

|  |
| --- |

|  |
| --- |

* Now trying access to secret page.It will give error because we dont have access to this page.

|  |
| --- |

* We can only have access to this route if we have token.
* So sign in again, and send token (copy the token from sign in response) in headers in authorization as given below.

|  |
| --- |

* Now sending request to secret page will be successful because we have authorized token.

|  |
| --- |

**=>Sign-out and Protected routes implemented Successfully,**

**Task Completed… :)**

# 