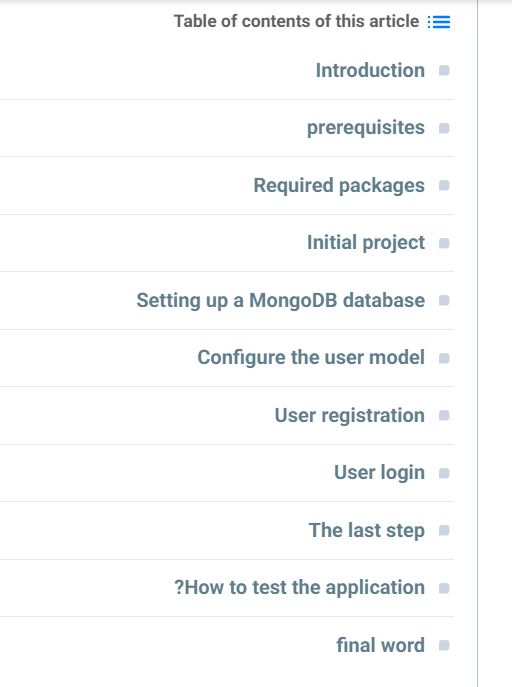
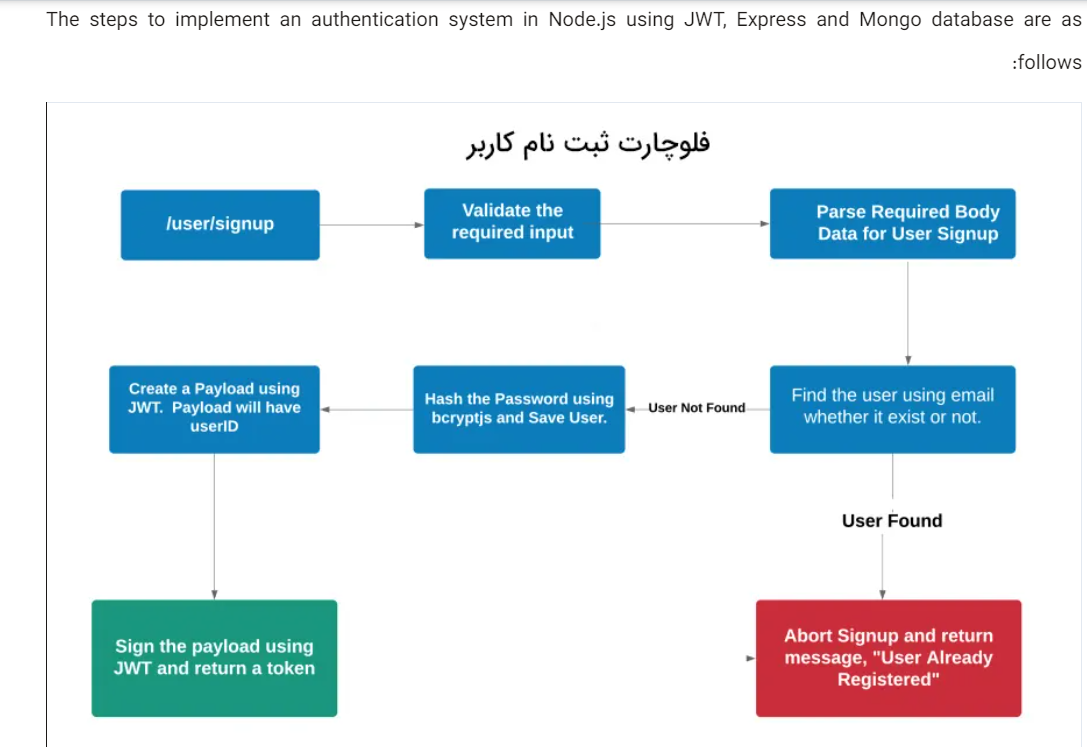
**01-Implementing authentication in Node.js with Express and JWT — from zero to hundred**

Link-

https://blog.faradars.org/implementing-authentication-in-nodejs-with-express-and-jwt-codelab/

1)



2) 

3)

* Express-it is a small and flexible node js web application framework that provides a complete set of features for web and mobile applications
* Express-validator- used to validate data on the server in the express framework. It is actually a server-side data validation library. So even if a user rejects client-side validation, they will get stuck here and an error will be issued
* Body-parser- it is a node js middleware for parsing body data
* Bcryptjs- library to hash the password and then store it in the database. In this way, even application administers cannot access a user’s account
* JWT- it is used to encrypt data payload during registration and token return. We can use that token to authenticate ourselves on secure pages like the dashboard. There is also an option to set the validity of these tokens and hence you can set how long the will last
* Mongoose- Mongoose is a mongodb object modelling tool designed to work in an asynchronous environment . Mongoose supports both promises and callbacks

4) const express = **require**("express");

const bodyparser = **require**("body-parser");

const app = **express**();

*// PORT*

const PORT = process.env.PORT || 5000;

app.**get**("/", (req, res) => {

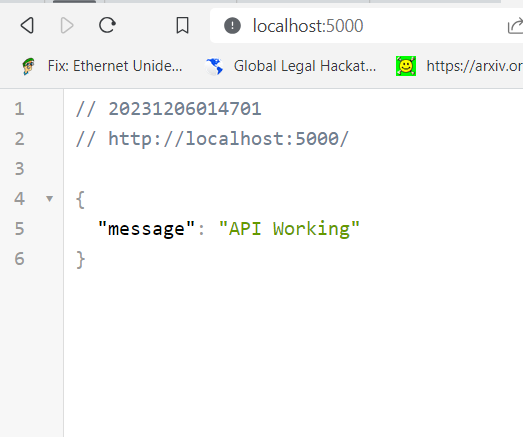
  res.**json**({ message: "API Working" });

});

app.**listen**(PORT, (req, res) => {

  console.**log**(`Server Started at PORT ${PORT}`);

});



5) Setting up Mongodb

* Copy the url from the mongo shell and paste in our code

6)Configure the user model

* Db.js

const mongoose = **require**("mongoose");

*// Replace this with your MongoURl*

const MONGOURl = "mongodb://127.0.0.1:27017";

const **InitiateMongoServer** = async () => {

  try {

    await mongoose.**connect**(MONGOURl, {

      useNewUrlParser: true,

    });

    console.**log**("connnected to db");

  } catch (e) {

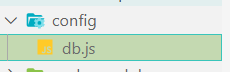
    console.**log**(e);

    throw e;

  }

};

module.exports = InitiateMongoServer;

* 

7)now the connection to the database is done , we create a user model to store the users logged into the application. For this purpose, **create a new folder name “model”** and create a file named “User.js”

* 

User.js

const mongoose = **require**("mongoose");

const UserSchema = mongoose.**Schema**({

  username: {

    type: String,

    required: true,

  },

  email: {

    type: String,

    required: true,

  },

  password: {

    type: String,

    required: true,

  },

  createdAt: {

    type: Date,

    required: Date.**now**(),

  },

});

*// export model user with UserSchema*

module.exports = mongoose.**model**("user,UserSchema");

now the work of connecting the database and the user schema so we can update the file index.js file to allow our api to connect with the database

**index.js**

const express = **require**("express");

const bodyparser = **require**("body-parser");

const InitiateMongoServer = **require**("./config/db");

*// Initialise the mongo server*

**InitiateMongoServer**()

const app = **express**();

*// PORT*

const PORT = process.env.PORT || 5000;

app.**get**("/", (req, res) => {

  res.**json**({ message: "API Working" });

});

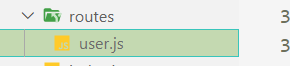
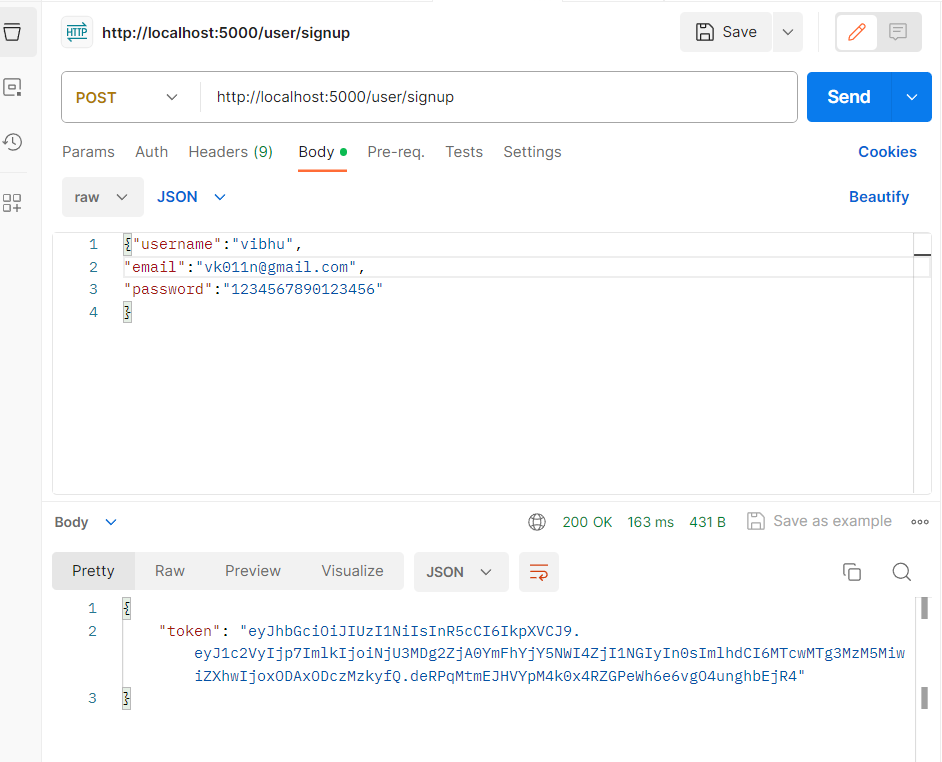
app.**listen**(PORT, (req, res) => {

  console.**log**(`Server Started at PORT ${PORT}`);

});

In this way we managed to connect the application to the MongoDb server. Now the next task is to create a path as /user/signup for the new user to register

8) User Registration

* The user registration path is user/signup/ create a folder call routes in this folder create a file called user.js and specify
* 
* 

**user.js**

* const express = **require**("express");
* const { check, validationResult } = **require**("express-validator/check");
* const bcrypt = **require**("bcryptjs");
* const jwt = **require**("jsonwebtoken");
* const router = express.**Router**();
* const User = **require**("..model/User");
* *//  \* @method - POST*
* *//  \* @param - /signup*
* *//  \* @description - User SignUp*
* router.**post**(
* "/signup",
* [
* **check**("username", "Please Enter a valid Username").**not**().**isEmpty**(),
* **check**("email", "please enter a valid email").**isEmail**(),
* **check**("pasword", "please enter a valid password").**isLength**({
* min: 10,
* }),
* ],
* async (req, res) => {
* const errors = **validationResult**(req);
* if (!errors.**isEmpty**()) {
* return res.**status**(400).**json**({
* errors: errors.**array**(),
* });
* }
* const { username, email, password } = req.body;
* try {
* let user = await User.**findOne**({ email });
* if (user) {
* return res.**status**(400).**json**({
* msg: "User already Exists",
* });
* }
* user = new **User**({
* username,
* email,
* password,
* });
* const salt = await bcrypt.**genSalt**(10);
* user.password = await bcrypt.**hash**(password, salt);
* await user.**save**();
* const payload = {
* user: {
* id: user.id,
* },
* };
* jwt.**sign**(
* payload,
* "randomstring",
* {
* expiresIn: 10000,
* },
* (err, token) => {
* if (err) {
* throw err;
* }
* res.**status**(200).**json**({
* token,
* });
* }
* );
* } catch (err) {
* console.**log**(err.message);
* res.**status**(500).**send**("error is saving");
* }
* }
* );
* Now the user registration is done in the routes/user.js file so we need to import it into index.js for it to work therefore we update the index.js file

9)USER LOGIN

* Now its time it implement the login router located at /user/login
* router.**post**(
* "/login",
* [
* **check**("email", "Please enter a valid email").**isEmail**(),
* **check**("password", "Please enter a valid password").**isLength**({
* min: 6,
* }),
* ],
* async (req, res) => {
* const errors = **validationResult**(req);
* if (!errors.**isEmpty**()) {
* return res.**status**(400).**json**({
* errors: errors.**array**(),
* });
* }
* const { email, password } = req.body;
* try {
* let user = await User.**findOne**({
* email,
* });
* if (!user)
* return res.**status**(400).**json**({
* message: "User Not Exist",
* });
* const isMatch = await bcrypt.**compare**(password, user.password);
* if (!isMatch)
* return res.**status**(400).**json**({
* message: "Incorrect Password !",
* });
* const payload = {
* user: {
* id: user.id,
* },
* };
* jwt.**sign**(
* payload,
* "secret",
* {
* expiresIn: 3600,
* },
* (err, token) => {
* if (err) throw err;
* res.**status**(200).**json**({
* token,
* });
* }
* );
* } catch (e) {
* console.**error**(e);
* res.**status**(500).**json**({
* message: "Server Error",
* });
* }
* }
* );
* module.exports = router;

10)THE LAST STEP

User registration and user login are now operational and you get a token in response. So our next job is to recover logged users using tokens. We will add this function further. The user/me/ route return the relevant user if the :token is sent in the header. Add the following in the root.js

*// @method-post*

*// @description-Get LoggedIn User*

*// @param- user/me*

router.**get**("/me", auth, async (req, res) => {

  try {

*// request.user is getting fetched from Middleware after token*

    const user = await User.**findById**(re.user.is);

    res.**json**(user);

  } catch (e) {

    res.**send**({ message: "Error in fetch" });

  }

});

Now we add middleware authentication that you can see on the user/me/me path. Next we create a folder called middleware and inside it we create a file called author.js is used to authenticate tokens and retrieve a user based on the token payload



const jwt = **require**("jsonwebtoken");

module.exports = function (req, res, next) {

  const token = req.**header**("token");

  if (!token) return res.**status**(401).**json**({ message: "Auth Error" });

  try {

    const decoded = jwt.**verify**(token, "randomstring");

    req.user = decoded.user;

**next**();

  } catch (e) {

    console.**log**(e);

    res.**status**(500).**send**({ message: "Invalid Token" });

  }

};