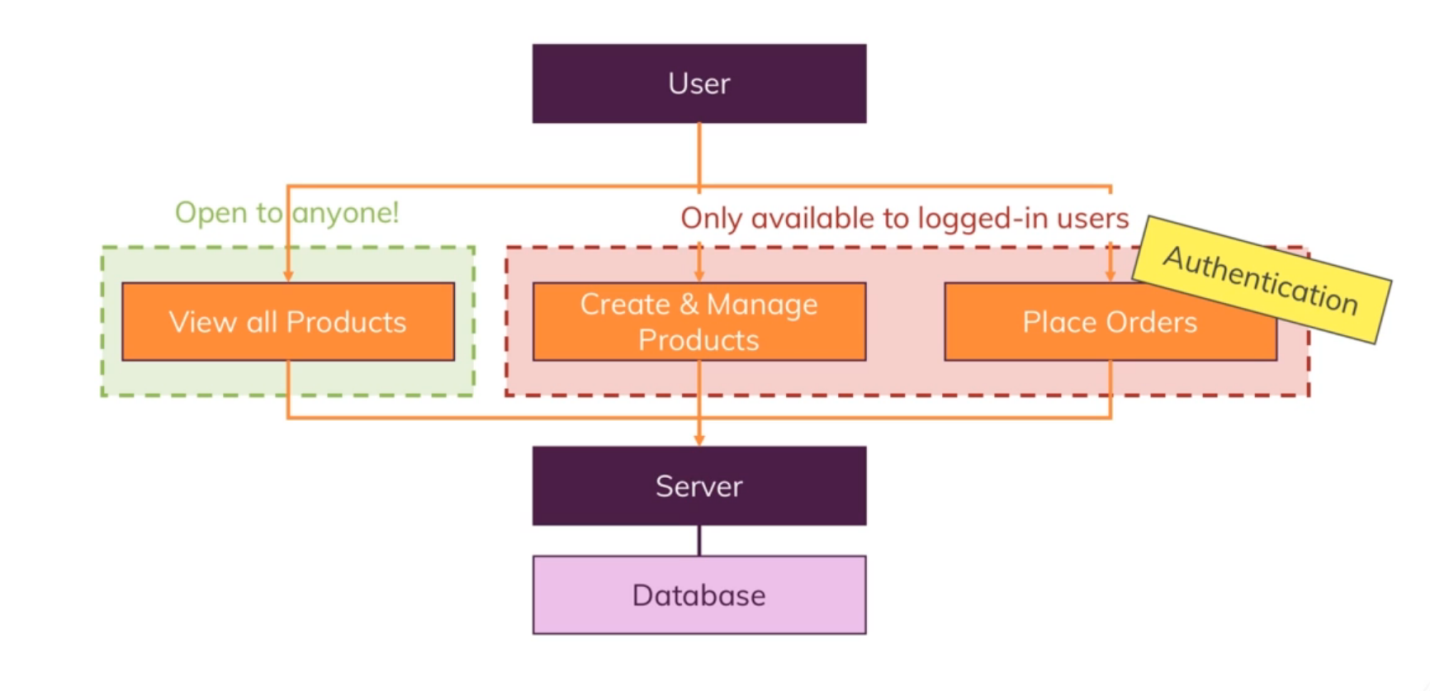
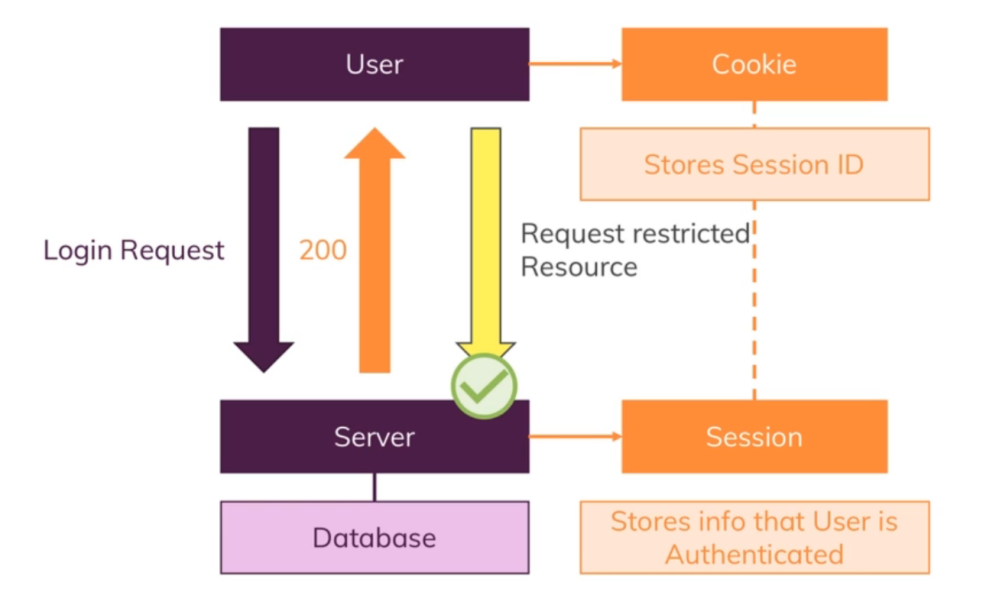
* **Authentication**

so authentication is used to stop unauthenticated user to interact with authenticated pages , so we want that place order only visible for authenticated user



**So how the authentication works**



So first use sends the login request then it goes to server where serve creates a session which stores the info that user is authenticated or not then response 200 is go to user which means it working and the cookie created with the id of the session so when the request go to the server it can check the user is authenticated or not

So now we have to give user signup functionality instead of providing a dummy user and we can do it in the auth.js controller

Point to remember that we have to provide the password in encrypted form in the database to secure the password and for that we have to install that package in our directory



const bcrypt = require('bcryptjs'); // used to encrypt the string

// now we are adding signup functionality

exports.postSignup = (req, res, next) => {

  const email = **req.body.email;** // we are getting these from our auth.ejs

  const password = **req.body.password;**

  const confirmPassword = req.body.confirmPassword;

  User.findOne({ email: email })

    .then(userDoc => { // if user is present find the email and redirect immediately

      if (userDoc) {

        return res.redirect('/signup');

      }

      return bcrypt // if not so we hash or encrypt the password

**.hash(password, 12)**

**.then(hashedPassword => {**

          const user = new User({

            email: email,

            password: hashedPassword,// save the hash/encrypt password

            cart: { items: [] }

          });

          return user.save(); // saving the user

        })

        .then(result => {

          res.redirect('/login');

        });

    })

    .catch(err => {

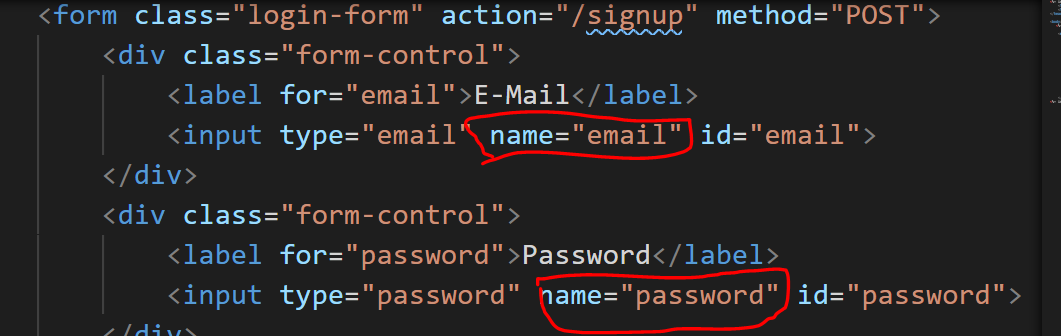
      console.log(err);

    });

};

So above we encrepted the password by **using bycript.hash(String,securityLevel) (first one is string of password an second is level of security (higher the level stronger the security increase the delay))**

so we get our email and password by req.body when we write in front end by signup ejs file



Now we create a login page if we provide false info so we should be rejected from login

// we want to change the type of login

exports.postLogin = (req, res, next) => {

  const email = req.body.email;

  const password = req.body.password;

  user.findOne({email:email}) // find the email present in database or not

  .then(user => {

    if(!user) { // if not present then redirect

      return res.redirect('/login')

    }

    // so here we encrypt the password and then compare it with user encrypted password

    bcrypt.compare(password,user.password)

    .then(doMatch => {

      if(doMatch) {

        req.session.isLoggedIn = true;

        req.session.user = user; // save the user in the session and save

        return req.session.save(err => {

          console.log(err);

          res.redirect('/');

        });

      }

      res.redirect('/login');

    })

  })

  .catch(err => console.log(err));

};

So above we find the email or not if not then redirect to login page as we provide wrong email if provide correct email then we check the password as we know password saved in encrypted format in our database so instead of decrypting it(which is not possible because we can not revert the encrypt) so we encrypt our user password and then compare it with the password in db then store the user in session for user ease and then redirect to main page.

But now we also want that no one can use the admin routes functionality as well some user functionality but the authentication person can do

And we can do it by checking is the user is logged in the session(true/false) as we provide the isloggedin = true at the time of login we create a session above

So for that instead of writing that logic for every routes we can create a middle ware folder in our project inside of it we create a isAuth.js file in which we write our code

// we creating this middleware to check the session isLoggedIn

// if yes then we move to next controller function

// we use this in our routs

module.exports = (req,res,next) => {

    if(!req.session.isLoggedIn) {

        return res.redirect('/login');

    }

    next();

}

So now we provide the exports in our admin and user routes in which we wants to like this

// check for user is authenticated or not

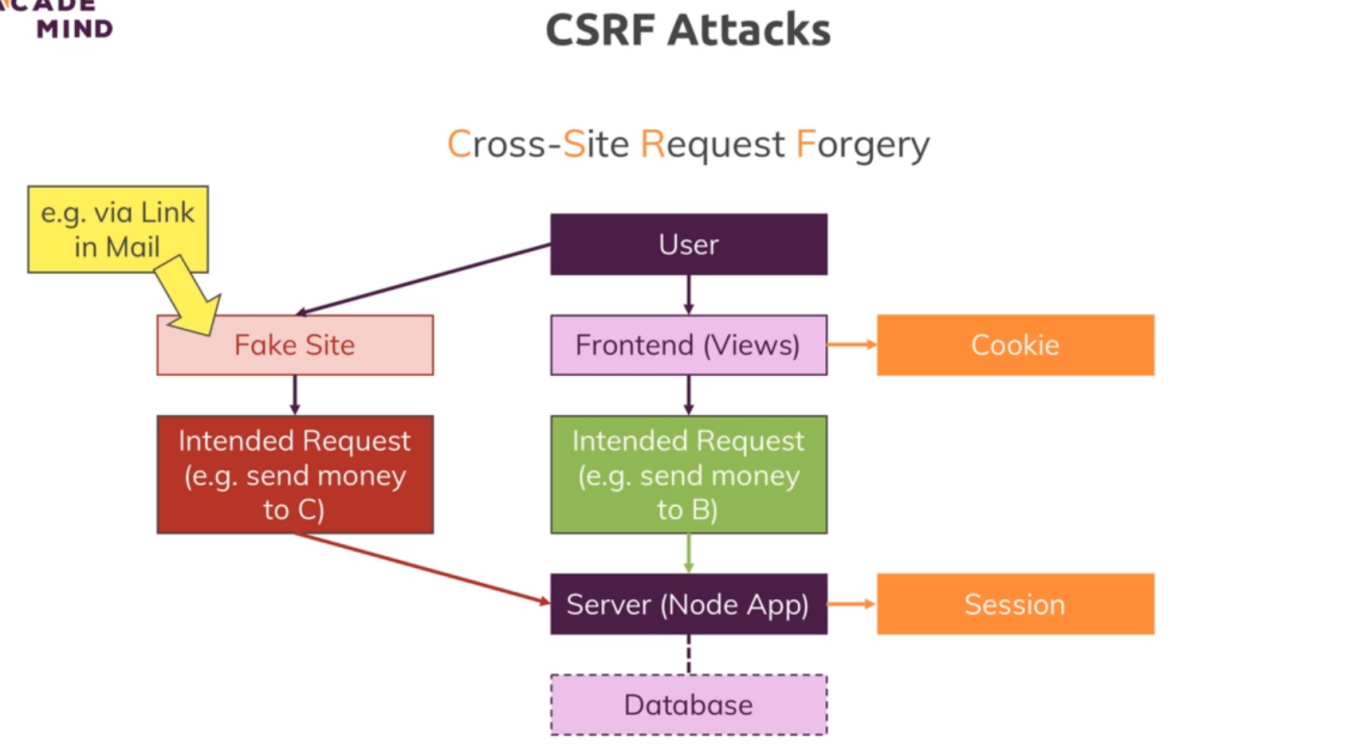
const isAuth = require('../middleWare/isAuth');

// /admin/add-product => GET

router.get('/add-product'**, isAuth,** adminController.getAddProduct);

// /admin/products => GET

router.get('/products'**,isAuth**, adminController.getProducts);

so it only runs when next calls otherwise redirect to the same page

* CSRF ATTACKS

So in csrf attack a hacker will create a page which look like yours so their when you will login the request go to the actual server and it generates the session and the cookies of that will go to the hacker where it can be dangers.

So to be saved by these attack we have to use a token into our front end page and every time we get a request at the time we will check that token is matched with our server if not then rejected. We are doing it because hacker can not put the same long token and token will change at all time so in this the token is generated by csurf package



So then we require this package and use it after session

const csrf = require('csurf');

const csrfProtection = csrf();

app.use(

  session({

    secret: 'my secret',

    resave: false,

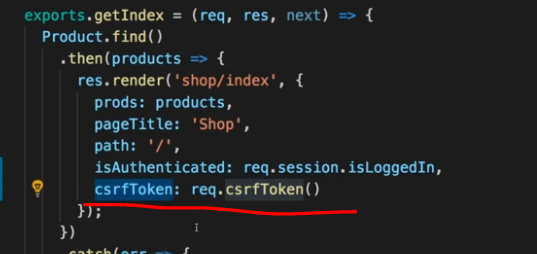
    saveUninitialized: false,

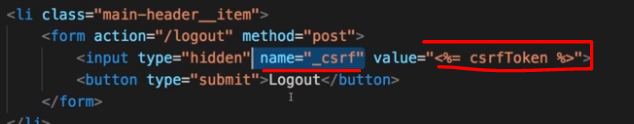
    store: store

  })

);

app.use(csrfProtection);

so we just use the csrf in our node now we just have to pass the csrf token in all our routs at time of getting and render the function so we can use it in our ejs file



so we can excess our csrf protection by req.csrfToken() and we are saving it at the object while render so we can use it in our front end ejs as value and name as \_csrf because the csrfProtection in our server can only understand the **\_csrf** name from the frontend and work on

but as you notice we are writing these token and is Authentication at each render request so what we can do is instead of writing it all over we can call it for all locally

and use it before calling the all the routes

app.use((req, res, next) => {

**res.locals.isAuthenticated = req.session.isLoggedIn;**

**res.locals.csrfToken = req.csrfToken();**

  next();

});

app.use('/admin', adminRoutes);

here we can use isAuthentcation an csrfToken in our view/ejs page.

* **Error while login in nice way**

So now we want to flash the error message in our front end if the user provide wrong email so what we do is we can use the flash but firs we have to install the flash



So we can require it and the have to use it to implement in our code

const flash = require('connect-flash');

app.use(flash());

now from above we can use the flash function by req as a key value pair

exports.postLogin = (req, res, next) => {

  const email = req.body.email;

  const password = req.body.password;

  User.findOne({ email: email })

    .then(user => {

      if (!user) {

**req.flash('error', 'Invalid email or password.');**

        return res.redirect('/login');

      }

So above we use req.flash pass the key and value as paired we can get the value when required

exports.getLogin = (req, res, next) => {

**let message = req.flash('error');**

  if (message.length > 0) {

    message = message[0];

  } else {

    message = null;

  }

  res.render('auth/login', {

    path: '/login',

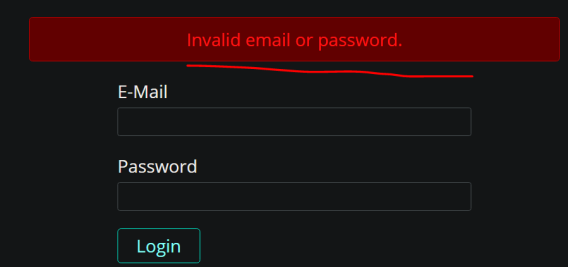
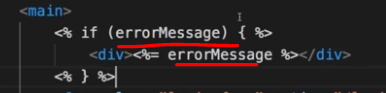
    pageTitle: 'Login',

**errorMessage: message**

  });

};

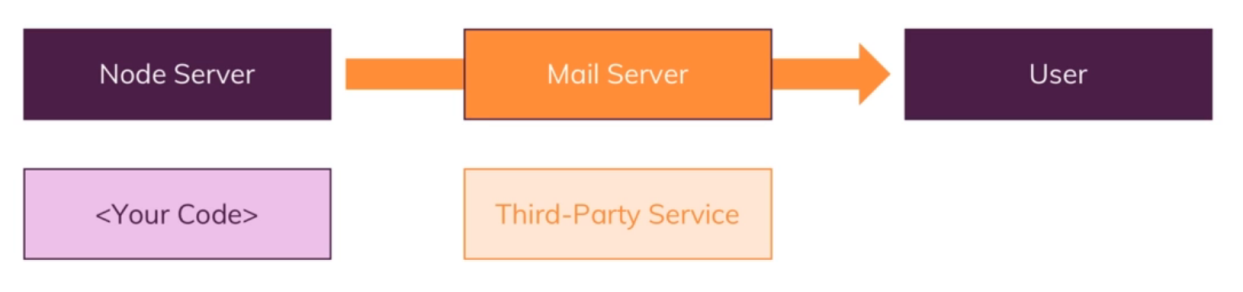
So flash give the key value in the form of array now we check if the message.length = 0 if so then we pass the null at message otherwise pass the message and we are doing this so we can get the value of it inside our view folder where print the error



So if the value is null it will not run that is why we put the value of errorMessage as null or so.

* Sending emails

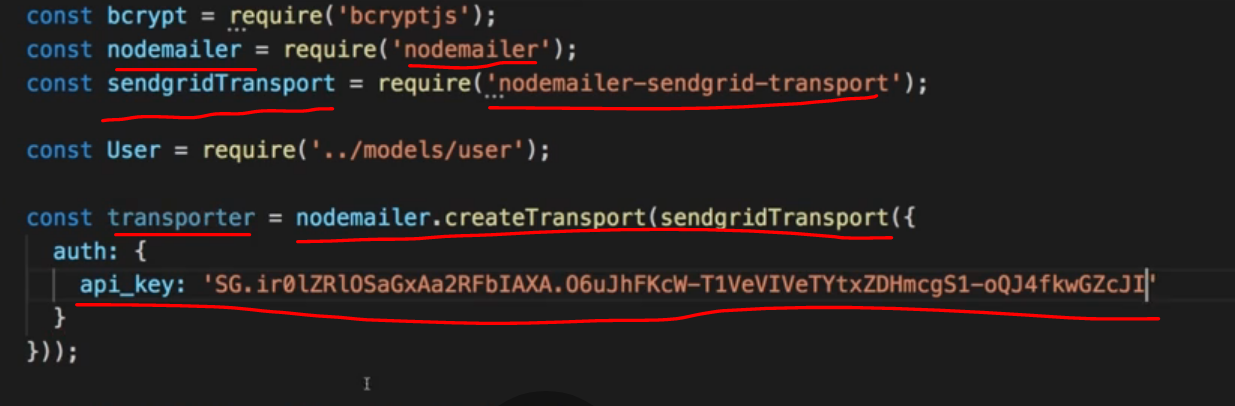
So sending email is difficult to manage by our own because it requires a lots of data in it



So we use third party mail service like aws so for that first we install the nodemailer

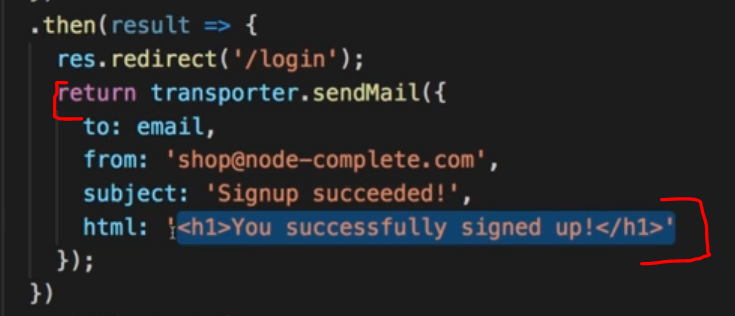


So we installed the nodemailer to se the interface and nodemailer-sendgrid-transport as third party mail server so we can connect it with that for that we also have to create a free account



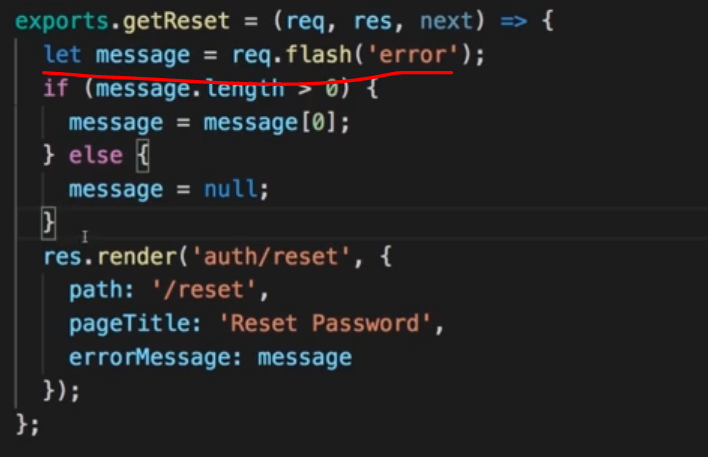
So here what we have to do is first require our both packages nodemailer to intract with third party and sendgrid which is the third party so we are doing this by create atransport between the interface and the third party with the api key of sendgrid account so we can store the data their by that once we done we can use it after signup/post Signup

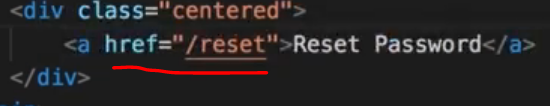
So in post signup we can call transport.sendMail() so inside of it all the data with pair will save



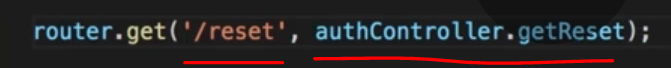
* Use reset password

So now we create a forgot password button and the page for the user for that and now we have create a controller and routes for that for controller we create a exportsof get reset in auth js





So when we click on forget password the route of reset call below



Now we have to create a post reset password which gives user the page to post your new password

exports.postReset = (req, res, next) => {

**crypto.randomBytes(32, (err, buffer) => {** //generate a random byte

    if (err) {

      console.log(err);

      return res.redirect('/reset');

    }

    const token = buffer.toString('hex'); // convert a random Byte to hexa decimal value (A14…etc)

    User.findOne({ email: req.body.email })

      .then(user => {

        if (!user) {

          req.flash('error', 'No account with that email found.');

          return res.redirect('/reset');

        }

        user.resetToken = token;

        user.resetTokenExpiration = Date.now() + 3600000;

        return user.save();

      })

      .then(result => {

        res.redirect('/');

        transporter.sendMail({

          to: req.body.email,

          from: 'shop@node-complete.com',

          subject: 'Password reset',

          html: `

            <p>You requested a password reset</p>

**<p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a new password.</p>**

          ` // token will be written here as dynamic content

        });

      })

      .catch(err => {

        console.log(err);

      });

  });

};

So crypto is defined in javascript we do not have to install it but require it though

const crypto = require('crypto');

so after that we added that token into our address as a dynamic content so that we can use it in our routes as dynamic controller and call in our view or ejs by the name we provide in our routs

router.get('/reset**/:token'**, authController.getNewPassword);

so as we called our get new password in this routes and can pass the token value as well

exports.getNewPassword = (req, res, next) => {

  const token = req.params.token; // passing token value from params

  User.findOne({ resetToken: token, resetTokenExpiration: { $gt: Date.now() } }) // find value where datanow() s greater then expiry date

    .then(user => {

      let message = req.flash('error'); //getting it from post new pas

      if (message.length > 0) {

        message = message[0];

      } else {

        message = null;

      }

      res.render('auth/new-password', {

        path: '/new-password',

        pageTitle: 'New Password',

        errorMessage: message,

        userId: user.\_id.toString(),//\_id from database

        passwordToken: token // token from dynamic routes

      });

    })

    .catch(err => {

      console.log(err);

    });

};

So now we call post new password but for this first we add some values into our user model like password and the password token to check for update

exports.postNewPassword = (req, res, next) => {

  const newPassword = req.body.password;

  const userId = req.body.userId;

  const passwordToken = req.body.passwordToken;

  let resetUser;

  User.findOne({//find one by all where dateNow is greater then expiry

    resetToken: passwordToken,

    resetTokenExpiration: { $gt: Date.now() },

    \_id: userId

  })

    .then(user => {

      resetUser = user; // save our found user in resetUser

      return bcrypt.hash(newPassword, 12); // encrypt

    })

    .then(hashedPassword => {

      resetUser.password = hashedPassword;

      resetUser.resetToken = undefined;

      resetUser.resetTokenExpiration = undefined;

      return resetUser.save();// update and save

    })

    .then(result => {

      res.redirect('/login');

    })

    .catch(err => {

      console.log(err);

    });

};

For update we found that user from our database and update them into our database

But now we still facing a little problem of the users. It is we did not define the user is admin or a end user because if they clashes then the user can add and delete the product from the admin the main so we want to restrict the user to excess the admin

so we can do it by doing the crud operation of admin only when we find the exect mach of the product and the user.\_id also so if the user id is equal to the id in the data base that means it is a admin so we pass the find check in all the crud operation in admin controller

exports.postAddProduct = (req, res, next) => {

  const title = req.body.title;

  const imageUrl = req.body.imageUrl;

  const price = req.body.price;

  const description = req.body.description;

  const product = new Product({

    title: title,

    price: price,

    description: description,

    imageUrl: imageUrl,

**userId: req.user** // pass at the time of add product

  });

exports.postEditProduct = (req, res, next) => {

  const prodId = req.body.productId;

  const updatedTitle = req.body.title;

  const updatedPrice = req.body.price;

  const updatedImageUrl = req.body.imageUrl;

  const updatedDesc = req.body.description;

  Product.findById(prodId)

    .then(product => {

// we can not add product until product userId = req.userid

**if (product.userId.toString() !== req.user.\_id.toString()) {**

        return res.redirect('/');

      }

exports.getProducts = (req, res, next) => {

**Product.find({ userId: req.user.\_id })**// run when find both

exports.postDeleteProduct = (req, res, next) => {

  const prodId = req.body.productId;

// run when find both

  Product.deleteOne({ \_id: prodId, userId: req.user.\_id })