Server requirements:

In the command prompt type npm install express --save in order to save the express on node.

Npm install –g nodemon –for automatic refresh of our server.

//1. used for installing express on nodejs

var express=require('express');

//2.app uses express now

var app=express();

//3.the file should be in static public folder in order to run on the desired port ie.the index file should be in public folder

app.use(express.static('public'));

// 4.used for port number and to run on it

app.listen(3000,function(){

console.log("some running message");

});

//5. if we are sending some data from server to controller where the controller requests the server for the data

step1.the controller asks the server

step2.server sends the data

step.3 the controller takes the data

//6.app.get is always used for sending the data to controller.here /detail is the route or say medium between the server adn controller they should always talk through this particular medium only in order to send or recive data from controller

app.get('/detail',function(req, res, next){

data will be stored here

});

-----------Here /detail is the address that send to

res.send(some data) or return res.send() --is the term used for sending the data to controller

7. $http is the get request to the server it asks the server for help ie any kind of data.so you must include this dependency injector in our controller.

app.get('/detail',function(req, res, next){

res.send(some data) ---this is from server

}); ---- /detail is the common address or medium they are talking in

$http.get('/detail').then(function(response){

console.log(response.data); ----------------------in the form of response we will get the data

some data will print in the console part ---- this is from controller

});

8.Now we will talk about getting data from controller to server..hmm ya posting data from view page how to get that data to server as of now no database is included it just war between server and controller.

Let consider like this we click on a submit button thye data must transfer from view to controller and then to server lets rock it then

For posting to server from controller we need $http.post—it just requires url and data

Ex: $scope.submit=function(){

console.log($scope.newalert);---the value of what we are going to post

$http.post('/detail', -----this is url part

{newalert:$scope.newalert} --------------this is the data that is sending in the form of object)

.then(function(){ --------------------this is call back function

})

};

To write something on the server you cant write it man just like that you need bodyparser for that mind it

npm install body-parser –save to install it

write var bodyParser=require('body-parser'); --------------in your server file

app.use(bodyParser.json()); -----------------to use it

To recive the data from the server end we just need----- app.post

app.post('/detail',function(req, res, next){

console.log(req) ----------req tells about the data coming from the controller

suppose if we want something to add in the web page we can use

datafile.push(req.body.comingdata)

datafile is the place where all the data has been stored

comingdata is the data that is we adding in the web page.

res.send() -------------------this is to ruin the call back function in the controller

});

Coming to the database part-----mongodb

Npm install mongodb –save

The below two are used for connecting the database

var MongoClient=require('mongodb').MongoClient;

var db=null;

MongoClient.connect("mongodb://localhost:27017/anunirva",function(err,dbconn){

if(!err){

console.log("we are connected to db");

db=dbconn;

}

});

Now getting data from the database to server and sending that data to controller----Major step dude

db.collection('details',function(err,detailsCollection){ ------details is my collection name

detailsCollection.find().toArray(function(err,details){

// console.log(details);

return res.json(details); -----------send data in the form of json only just remember it

})

});

Sending data to controller is fine now will see how to store data in db from controller

db.collection('details',function(err,detailsCollection){

var newalert={

text:req.body.newalert, ----------saving the data inj the form of object

};

detailsCollection.insert(newalert,{w:1},function(err){

return res.send();

})

});

We will see how to automatically refresh the data when we add something.simple dude

After whenever u post the data u have to deal with the get data

Just use the refresh button

Ex:$http.post(){

Refresh();

}

Function refresh(){

$http.get()

}

Now how to remove the data from db with the use of web page just like clicking on the X button.lets see that mann

This code should be in controller

$scope.removeit=function(msg){ ---------msg is the item that u want to delete

$http.put('/detail/remove',{x:msg}.then(function(){ -------------x is the property

Refresh()

});

}

Now on server side

app.put('/detail/remove',function(req, res, next){

db.collection('details',function(err,detailsCollection){

var myId=req.body.x.\_id;

console.log(myId);

detailsCollection.remove({\_id:ObjectId(myId),user:user.\_id},{w:1},function(err){

return res.send();

})

}}

Now signup and login: more interestinggg..!!!!!

Now we will see about the signup function just by clicking on a button:

$scope.signupsubmit=function(){

console.log("this is signup page dude");

var newUser={

username:$scope.username, ------------------------now saving data in a object

password:$scope.password

};

console.log(newUser);

$http.post('/users',newUser).then(function(){ ---------------sending data to the server ayya

alert("newuser sended to the server"); -----------------this will alert once the server gets the data

});

}

Now receiving this data on the server side:

app.post('/users',function(req, res, next){

db.collection('users',function(err,usersCollection){ --------this users is one more collection in our databas

usersCollection.insert(newUser,{w:1},function(err){ -----newUser we got from the webp

console.log(req.body);

return res.send();

Now the best part macha encrypting the password with the help of bycryptjs:

We use bcryptjs

Npm install bcryptjs –save

var bcrypt=require('bcryptjs'); ----in server file

app.post('/users',function(req, res, next){

db.collection('users',function(err,usersCollection){

bcrypt.genSalt(10,function(err,salt){ -----------this is the code for bcrypt

bcrypt.hash(req.body.password,salt,function(err,hash){

var newUser={

username:req.body.username, --------just remember the code dude

password:hash

};

usersCollection.insert(newUser,{w:1},function(err){

console.log(req.body);

return res.send();

});

Now coming to the signin part with the registerd users:

We will sigin the users with the click of button:

Here we are posting the data to server to check,so for checking here we are using the put method

$scope.signin=function(){

$http.put('/users/signin',{username:$scope.username,password:$scope.password}).then(function(res){

alert("sign in sended"); ---here we are just sending the data to the server

} ,function(err){

alert("bad login"); ---------this will alert if there is any bad login

})

};

Now on server side:

app.put('/users/signin',function(req, res, next){

db.collection('users',function(err,usersCollection){

usersCollection.findOne({username:req.body.username},function(err,user){

bcrypt.compare(req.body.password,user.password,function(err,result){

if(result){

res.send() -----if correct this will get alerted

}

else{

return res.status(400).send();

}

})

})

Now one of the best part coming in saving the signed in user if u refresh the page also,we can do thgis part by saving the signed user with the help of token:

Storing a session on browser:

Npm install jwt-simple –save

var jwt=require('jwt-simple'); ----------to save tokens

var JWT\_SECRET='secret key'; ------------gives specific key to our token ie when we restart our server this is going to be same never change

now coming to the signin part if there is successful login we need to save it so lets do this

app.put('/users/signin',function(req, res, next){

db.collection('users',function(err,usersCollection){

usersCollection.findOne({username:req.body.username},function(err,user){

bcrypt.compare(req.body.password,user.password,function(err,result){

if(result){

var token=jwt.encode(user,JWT\_SECRET); ---just include this

res.send() -----if correct this will get alerted

}

else{

return res.status(400).send();

in controller part to se the token just use this

$scope.signin=function(){

$http.put('/users/signin',{username:$scope.username,password:$scope.password}).then(function(res){

console.log(res.data.token); ---when u click on signin buton we can see the token

now one of the crazy part,we need to save this token in a cookie ie on that particylar web page

Cookie: for this we need one more parrticulr ffile

Ng-Cookie is one more dependency so include it on the app controller

$scope.signin=function(){

$http.put('/users/signin',{username:$scope.username,password:$scope.password}).then(function(res){

console.log(res.data.token);

$cookies.put('token',res.data.token); (key,value)

Now here comes nthe rootscope which is used to share data between controllers.

$rootScope.token=res.data.token;

$rootScope.currentUser=$scope.username; -------------heree we are saving the botth token aand curentuser

Now we need to use app.run because Run blocks - get executed after the injector is created and are used to kickstart the application. Only instances and constants can be injected into run blocks. This is to prevent further system configuration during application run time.

app.run(function($rootScope,$cookies){

if($cookies.get('token') && $cookies.get('currentUser')){

$rootScope.token=$cookies.get('token'); ----------whenever u run the application u can se

$rootScope.currentUser=$cookies.get('currentUser');

}

})

Now both the cookie and this name can be seen on Application-cookeis

Hiding items-when loged in:

One more crazy item,if I login there is no need of showing login button and signup button right?Lets do the stuff of hiding some portions if LOGIN IN.This is going to be craxzy macha

So the portion which is going to be hide putthat in a div.for that we use ng-hide right? YES

SO in the div type ng-hide=”currentUser” this will hide once I loged in because thaats true.

Showing items-when loged in:

Wow one more stuff here to do.so when I logged in I need to show some items dude like signout like that yuppp…this is nice mannn.so for that put the items that are to be shown in a div

In that ng-show=”currentUser” so the user can see the items if he logged in

Here one more item if I logged in I can manipulate only my items not others like twitter where I can just tweet I cant delete others tweets that’s not good to if I delete..phewww just do that

And I when I logout I need to delete all my details right..my token and my username hmm do that then

$scope.logouting=function(){

$cookies.remove('token'); -----here im removing

$cookies.remove('currentUser');

$rootScope.token=null; -----------here im setting the details to null

$rootScope.currentUser=null;

}

Now whose items are whose: here we can delete or modify only our own files.for that wee need authorization.

This is in the controller part--

$scope.submit=function(){

$http.post('/detail',

{newalert:$scope.newalert},

{headers:{'authorization':$rootScope.token}}).then(function(){ ---here headers are the most important

Refresh()

$scope.newalert='';

})

};

Now in the server side in the same post method we have –

app.post('/detail',function(req, res, next){

var token=req.headers.authorization;

var user=jwt.decode(token,JWT\_SECRET); ----here both the collections are met details and users

db.collection('details',function(err,detailsCollection){

var newalert={

text:req.body.newalert,

user:user.\_id,

username:user.username

};

----so in simple words it is with the help of token we gone into details collections and add the data from the user collection to the detail collection

Now we cant modify others details for that—keep on

Headers are the identification cells that gives info on the current user different to each and other user

Now we have in the removeit fn from the controller

$scope.removeit=function(msg){

$http.put('/detail/remove',{x:msg},

{headers:{'authorization':$rootScope.token}}).then(function(){

getalerts();

});

}

Now go to the server part then we have to include the same there too..

app.put('/detail/remove',function(req, res, next){

var token=req.headers.authorization; ------these are identifiers

var user=jwt.decode(token,JWT\_SECRET);

db.collection('details',function(err,detailsCollection){

var myId=req.body.x.\_id;

console.log(myId);

detailsCollection.remove({\_id:ObjectId(myId),user:user.\_id},{w:1},function(err){

return res.send();

})

user:user.\_id —--------------it indicates only that user can remove it this is utmost important here

here also we can see we get through the details collection by only having the user collection this can be possible with the ghelp of headers only.