



Mahesh Bhanushali

Follow

Software Developer|Freelancer|JavaScript Learner|Laravel|PHP|HappyToHelp

Apr 29 · 9 min read

Building a Restful CRUD API with Node.js, JWT, Bcrypt, Express and MongoDB



Introduction

In this tutorial we will be developing simple REST API for movies collection with their released date. We gonna implement simple CRUD(Create, Read, Update and Delete) operations on movie collection data in our journey.

Our application will have public API as well as protected API when we say public that mean it can be accessed without any authentication whereas protected we need to authenticate our self to access them. For authentication, we will be using JWT(Json Web Token). Movie collection application will have the following workflow

- User register through the registration form. (Name, Email, Password)
- The user is authenticated by providing email and password
- On successful authentication, JsonWebToken(JWT) is returned in response to access protected routes.
- In subsequent access to protected route, the user sends JWT token in the header for authentication instead of email and password.

Prerequisite

Before we start with coding following list needs to be installed. I am working on windows environment.

- [nodejs](#)
- [npm](#)

- mongodb

For windows environment MongoDB community version is recommended for quick start

I assume above list has been installed successfully on your development environment

Implementation

Open terminal and navigate to directory where you want your application to stay and follow following steps

```
mkdir node-rest-api-jwt
cd node-rest-api-jwt
npm init
```

After **npm init** you will be asked few questions one by one, keep default options as it is and keep hitting enter button

```
package name: (node-rest-api-jwt)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
```

```
About to write to C:\NodejsTut\node-rest-api-  
jwt\package.json:  
{  
  "name": "node-rest-api-jwt",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

Is this OK? (yes)

Now install following dependencies

```
npm install bcrypt body-parser express jsonwebtoken mongoose  
morgan --save  
  
npm install nodemon -g
```

- bcrypt :- hash your plain password and store hashed password in database
- body-parser :- Parse incoming request bodies in a middleware before your handlers, available under the `req.body` property
- express :- Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

- jsonwebtoken :- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.

You can read more about JWT

- mongoose :- Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment.
- morgan :- HTTP request logger middleware for node.js
- nodemon :- nodemon will watch the files in the directory in which nodemon was started, and if any files change, nodemon will automatically restart your node application.

After installing dependencies your package.json file will have the list of all installed dependencies

```
"dependencies": {  
  "bcrypt": "2.0.1",  
  "body-parser": "1.18.2",  
  "express": "4.16.3",  
  "jsonwebtoken": "8.2.1",  
  "mongoose": "5.0.13",  
  "morgan": "1.9.0"  
},
```

. . .

Now we are good to start with development. Move to root folder of the application i.e node-rest-api-jwt and create following directory

```
cd node-rest-api-jwt
mkdir app
cd app
mkdir api
cd api
mkdir controllers
mkdir models
```

We will follow MVC pattern so our application controllers and models files will be stored in app/api/controllers, app/api/models respective directories.

create folder config inside application root directory.

```
mkdir config
```

We will store our database configuration file in config folder. i.e node-rest-api-jwt/config

Next we will create route folder inside our application root directory where all of our application related routes files will be created.

```
mkdir routes
```

Lets create node server, create a file server.js inside directory node-rest-api-jwt

```
const express = require('express');
const logger = require('morgan');
const bodyParser = require('body-parser');
const app = express();

app.use(logger('dev'));
app.use(bodyParser.urlencoded({extended: false}));

app.get('/', function(req, res){
  res.json({"tutorial" : "Build REST API with node.js"});
});

app.listen(3000, function(){ console.log('Node server
listening on port 3000');});
```

Now open terminal and from application root folder fire command

```
nodemon server.js
```

You will see in terminal


```
[nodemon] 1.11.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
Node server listening on port 3000
```

Now open browser and type localhost:3000/

```
{"tutorial": "Build REST API with node.js"}
```

http://localhost:3000

. . .

Now lets create model file for user in node-rest-api-jwt/app/api/models/users.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');
const saltRounds = 10;
```

```
//Define a schema
const Schema = mongoose.Schema;

const UserSchema = new Schema({
  name: {
    type: String,
    trim: true,
    required: true,
  },
  email: {
    type: String,
    trim: true,
    required: true
  },
  password: {
    type: String,
    trim: true,
    required: true
  }
});

// hash user password before saving into database
UserSchema.pre('save', function(next){
  this.password = bcrypt.hashSync(this.password, saltRounds);
  next();
});

module.exports = mongoose.model('User', UserSchema);
```

In model file we include mongoose for model schema, bcrypt to hash our password and also defined salt round which will be used for hashing plain text password. Mongoose provide middleware(pre/post hooks) which we can use to manipulate our data before/after inserting into database. We have used **pre hook save method** to hash our password before saving into database. In **pre hooks** callback function **this** keyword refer to UserSchema object and this.password is password

passed from registration form which we will be creating soon. You can read more about mongoose and its function at <https://github.com/Automattic/mongoose>

Now create controller for user in node-rest-api-jwt/app/api/controllers/users.js

```
const userModel = require('../models/users');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

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

    userModel.create({ name: req.body.name, email:
req.body.email, password: req.body.password }, function
(err, result) {
      if (err)
        next(err);
      else
        res.json({status: "success", message: "User added
successfully!!!", data: null});

    });
  },

  authenticate: function(req, res, next) {
    userModel.findOne({email:req.body.email}, function(err,
userInfo){
      if (err) {
        next(err);
      } else {

        if(bcrypt.compareSync(req.body.password, userInfo.password))
        {
```

```
const token = jwt.sign({id: userInfo._id},
  req.app.get('secretKey'), { expiresIn: '1h' });

res.json({status:"success", message: "user found!!!", data:
  {user: userInfo, token:token}});

}else{

res.json({status:"error", message: "Invalid
email/password!!!", data:null});

}

}
});
},
}

}
```

User controller include user model, jsonwebtoken module and bcrypt. And we have defined two methods

- create :- To create new users, this function has model query too create new user into database.
- authenticate :- In this function we search for user in database by email id and compare plain password passed through login form with database hashed password, if password match we generate jwt token by passing user id and secret key, and have set 1hr validity of the token. You can explore more option on jwt at <https://jwt.io/introduction/> & <https://github.com/auth0/node-jsonwebtoken>

Now time to create routes for above users controller methods, node-rest-api-jwt/routes/users.js

```
const express = require('express');
const router = express.Router();
const userController =
require('../app/api/controllers/users');

router.post('/register', userController.create);
router.post('/authenticate', userController.authenticate);

module.exports = router;
```

In route we included user controller and each post route is mapped with respective controller method.

. . .

Create model, controller and route file for protected movie routes as we created for user

node-rest-api-jwt/app/api/controllers/movies.js

```
const movieModel = require('../models/movies');

module.exports = {
  getById: function(req, res, next) {
    console.log(req.body);
    movieModel.findById(req.params.movieId, function(err,
```

```
movieInfo){
  if (err) {
    next(err);
  } else {
    res.json({status:"success", message: "Movie found!!!",
data:{movies: movieInfo}});
  }
});
},

getAll: function(req, res, next) {
  let moviesList = [];

  movieModel.find({}, function(err, movies){
    if (err){
      next(err);
    } else{
      for (let movie of movies) {
        moviesList.push({id: movie._id, name: movie.name,
released_on: movie.released_on});
      }
      res.json({status:"success", message: "Movies list
found!!!", data:{movies: moviesList}});
    }

  });
},

updateById: function(req, res, next) {
  movieModel.findByIdAndUpdate(req.params.movieId,
{name:req.body.name}, function(err, movieInfo){

if(err)
  next(err);
else {
  res.json({status:"success", message: "Movie updated
successfully!!!", data:null});
}
});
},
```

```
deleteById: function(req, res, next) {
  movieModel.findByIdAndRemove(req.params.movieId,
function(err, movieInfo){
  if(err)
    next(err);
  else {
    res.json({status:"success", message: "Movie deleted
successfully!!!", data:null});
  }
});
},

create: function(req, res, next) {
  movieModel.create({ name: req.body.name, released_on:
req.body.released_on }, function (err, result) {
    if (err)
      next(err);
    else
      res.json({status: "success", message: "Movie added
successfully!!!", data: null});

  });
},

}
```

node-rest-api-jwt/app/api/models/movies.js

```
const mongoose = require('mongoose');

//Define a schema
const Schema = mongoose.Schema;

const MovieSchema = new Schema({
  name: {
    type: String,
```

```
    trim: true,
    required: true,
  },
  released_on: {
    type: Date,
    trim: true,
    required: true
  }
});

module.exports = mongoose.model('Movie', MovieSchema)
```

node-rest-api-jwt/routes/movies.js

```
const express = require('express');
const router = express.Router();
const movieController =
require('../app/api/controllers/movies');

router.get('/', movieController.getAll);
router.post('/', movieController.create);
router.get('/:movieId', movieController.getById);
router.put('/:movieId', movieController.updateById);
router.delete('/:movieId', movieController.deleteById);

module.exports = router;
```

. . .

Open server.js file and update existing code with following


```
const express = require('express');
const logger = require('morgan');
const movies = require('./routes/movies') ;
const users = require('./routes/users');
const bodyParser = require('body-parser');
const mongoose = require('./config/database'); //database
configuration
var jwt = require('jsonwebtoken');
const app = express();

app.set('secretKey', 'nodeRestApi'); // jwt secret token

// connection to mongodb
mongoose.connection.on('error', console.error.bind(console,
'MongoDB connection error:'));

app.use(logger('dev'));
app.use(bodyParser.urlencoded({extended: false}));

app.get('/', function(req, res){
  res.json({"tutorial" : "Build REST API with node.js"});
});

// public route
app.use('/users', users);

// private route
app.use('/movies', validateUser, movies);

app.get('/favicon.ico', function(req, res) {
  res.sendStatus(204);
});

function validateUser(req, res, next) {
  jwt.verify(req.headers['x-access-token'],
req.app.get('secretKey'), function(err, decoded) {
    if (err) {
      res.json({status:"error", message: err.message,
```

```
data:null});
    }else{
      // add user id to request
      req.body.userId = decoded.id;
      next();
    }
  });

}

// express doesn't consider not found 404 as an error so we
// need to handle 404 explicitly
// handle 404 error
app.use(function(req, res, next) {
  let err = new Error('Not Found');
  err.status = 404;
  next(err);
});

// handle errors
app.use(function(err, req, res, next) {
  console.log(err);

  if(err.status === 404)
    res.status(404).json({message: "Not found"});
  else
    res.status(500).json({message: "Something looks wrong :(
!!!"});

});

app.listen(3000, function(){
  console.log('Node server listening on port 3000');
});
```

To access movie routes we have defined middleware to validate user

```
// private route
app.use('/movies', validateUser, movies);

function validateUser(req, res, next) {
  jwt.verify(req.headers['x-access-token'],
    req.app.get('secretKey'), function(err, decoded) {
    if (err) {
      res.json({status:"error", message: err.message,
data:null});
    }else{
      // add user id to request
      req.body.userId = decoded.id;
      next();
    }
  });
}
```

Database Connection

Create database connection configuration file in node-rest-api-jwt/config/database.js

```
//Set up mongoose connection
const mongoose = require('mongoose');
const mongoDB = 'mongodb://localhost/node_rest_api';
mongoose.connect(mongoDB);
mongoose.Promise = global.Promise;

module.exports = mongoose;
```

and in server.js file we have setup connection to MongoDB

```
// connection to mongodb
mongoose.connection.on('error', console.error.bind(console,
'MongoDB connection error:'));
```

For windows users to start MongoDB server if you have installed community version on your system

Create a data folder in node-rest-api-jwt.

```
mkdir data
```

Now open terminal navigate to mongo folder where you have installed mostly it will be in c:\Program Files\MongoDB\Server\3.4\3.2\bin

```
mongod --dbpath c:\path to node-rest-api-jwt\data
```

This command will start MongoDB server and store all your collection in your application data folder, now open another terminal and hit command

```
nodemon server.js
```

| You can view your MongoDB collection by installing Robo 3T application

One thing I want to add here is that you have noticed we haven't created any collection in MongoDB, so creating collection on fly is handled by mongoose. You can read more on this by exploring mongoose document.

Testing

For testing we will use http client postman

route list

- GET http://localhost:3000
- POST http://localhost:3000/users/register

The screenshot shows the Postman application interface. The top bar includes a 'NEW' button, a 'Runner' tab, an 'Import' button, and a 'Builder' tab. A notification banner at the top states: 'Chrome apps are being deprecated. Download our free native apps for continued support and better performance. Learn more'. The left sidebar shows a 'Collections' view with a folder named 'node-rest-api-jwt' containing 4 requests: 'index', 'create new user', 'authenticate', and 'create movie'. The 'create new user' request is selected. The main panel shows the details of this POST request to 'http://localhost:3000/users/register'. The 'Body' tab is active, showing 'x-www-form-urlencoded' data with the following fields:

Key	Value	Description
name	Maresh Bhanushali	
email	maresh@gmail.com	
password	123456	

The response is shown in the 'Body' tab, displaying a JSON object:

```
{
  "status": "success",
  "message": "User added successfully!!!",
  "data": null
}
```

The status is '200 OK' and the time is '199 ms'. The bottom of the image shows a Windows taskbar with the time '03:08 AM' and date '30-04-2018'.

POST http://localhost:3000/users/register

- POST <http://localhost:3000/users/authenticate>

The screenshot shows the Postman application interface. On the left, the 'Collections' tab is active, showing a collection named 'node-rest-api-jwt' with 4 requests. The 'authenticate' request is selected. The main panel displays the details of this POST request to 'http://localhost:3000/users/authenticate'. The request body is set to 'x-www-form-urlencoded' with the following data:

Key	Value	Description
email	maresh@gmail.com	
password	123456	

The response is shown in the 'Body' tab, formatted as JSON. The status is 200 OK and the time taken is 217 ms. The response body is:

```

1 {
2   "status": "success",
3   "message": "user found!!!",
4   "data": {
5     "user": {
6       "_id": "5ae6379427bbe422cce696a0",
7       "name": "Mahesh Bhanushali",
8       "email": "maresh@gmail.com",
9       "password": "$2b$10$JAehr6o1sdyWJf9QtfrWeUcvhpFXHCDgPhnWPhF6vN5vc6zDTmfa",
10      "_v": 0
11    },
12    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVhZTYyNzk0MjYyYmU0MjYyY2U2OTZlMCI6Im1hdCI6MTUyNTA5Nzk3OSwiZXN1IjozNTI1MDQxNTc5fQ.~R4Sjf2-aAXI14f1Dp0CX4kopJCvKtTavcSSGr6ejE0"
13  }
14 }

```

The Windows taskbar at the bottom shows the time as 03:09 AM on 30-04-2018.

POST http://localhost:3000/users/authenticate

JWT token received in above request will be used to access protected movie routes. We will pass token in header with key=x-access-token and value=jwt token

- POST <http://localhost:3000/movies>

The screenshot shows the Postman application interface. The top bar includes buttons for 'NEW', 'Runner', 'Import', and 'Builder'. A notification banner at the top states: 'Chrome apps are being deprecated. Download our free native apps for continued support and better performance. Learn more'. The left sidebar shows a 'Collections' view with two collections: 'node-rest-api-jwt' (4 requests) and 'Postman Echo' (21 requests). The main panel displays a 'create movie' request. The request is a POST to 'http://localhost:3000/movies/'. The 'Headers' tab is selected, showing two headers: 'x-access-token' with value 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVhZTYz...' and 'Content-Type' with value 'application/x-www-form-urlencoded'. The 'Body' tab is also visible. The response is shown in the 'Body' tab, displaying a JSON object:

```
{  "status": "success",  "message": "Movie added successfully!!!",  "data": null}
```

. The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 265 ms'. The Windows taskbar at the bottom shows the system clock as 03:05 AM on 30-04-2018.

POST http://localhost:3000/movies

The screenshot shows the Postman application interface. On the left, the 'Collections' tab is active, showing a collection named 'node-rest-api-jwt' with 4 requests. The 'create movie' request is selected. The main panel displays the details of this POST request to 'http://localhost:3000/movies/'. The request body is set to 'x-www-form-urlencoded' with two parameters: 'name' with value 'Iron man' and 'released_on' with value '2018-05-12'. The response tab shows a successful status of 200 OK with a time of 265 ms. The response body is displayed in JSON format:

```
{  "status": "success",  "message": "Movie added successfully!!!",  "data": null}
```

POST <http://localhost:3000/movies>

- GET <http://localhost:3000/movies>

The screenshot shows the Postman application interface. On the left, the 'Collections' tab is active, showing a collection named 'node-rest-api-jwt' with 4 requests. The 'create movie' request is selected. The main panel shows the details of the 'create movie' request, which is a GET request to 'http://localhost:3000/movies/'. The 'Headers' tab is active, showing two headers: 'x-access-token' and 'Content-Type'. The 'Body' tab is also visible, showing a JSON response. The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 179 ms'.

Headers (2)

Key	Value	Description
<input checked="" type="checkbox"/> x-access-token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjVhZTYz...	
<input checked="" type="checkbox"/> Content-Type	application/x-www-form-urlencoded	
New key	Value	Description

Body

Pretty Raw Preview JSON

```

1 {
2   "status": "success",
3   "message": "Movies list found!!!",
4   "data": {
5     "movies": [
6       {
7         "id": "5ae6398627bbe422cce696a1",
8         "name": "Iron man",
9         "released_on": "2018-05-12T00:00:00.000Z"
10      }
11     ]
12   }
13 }

```

Status: 200 OK Time: 179 ms

Save Response

http://localhost:3000/movies

- GET <http://localhost:3000/movies/5ae63c5027bbe422cce696a3>

The screenshot shows the Postman application interface. On the left, the 'Collections' tab is active, showing a collection named 'node-rest-api-jwt' with 4 requests. The 'create movie' request is selected. The main panel displays the details of this request, which is a GET request to the URL `http://localhost:3000/movies/5ae6398627bbe422cce696a1`. The 'Headers' tab is selected, showing two headers: 'x-access-token' with a long JWT token and 'Content-Type' set to 'application/x-www-form-urlencoded'. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format. The response status is '200 OK' and the time taken is '98 ms'. The JSON response is as follows:

```

1 {
2   "status": "success",
3   "message": "Movie found!!!",
4   "data": {
5     "movies": {
6       "_id": "5ae6398627bbe422cce696a1",
7       "name": "Iron man",
8       "released_on": "2018-05-12T00:00:00.000Z",
9       "_v": 0
10    }
11  }
12 }

```

At the bottom of the window, the Windows taskbar is visible, showing the system clock as 03:13 AM on 30-04-2018.

`http://localhost:3000/movies/5ae63c5027bbe422cce696a3`

- PUT <http://localhost:3000/movies/5ae63c5027bbe422cce696a3>

The screenshot shows the Postman application interface. The top bar includes buttons for 'NEW', 'Runner', 'Import', and 'Builder'. A notification banner at the top states: 'Chrome apps are being deprecated. Download our free native apps for continued support and better performance. Learn more'. The left sidebar shows a 'Collections' view with a folder named 'node-rest-api-jwt' containing 4 requests. The main panel displays a REST client request for a PUT method to the URL 'http://localhost:3000/movies/5ae6398627bbe422cce696a1'. The 'Headers' tab is selected, showing two headers: 'x-access-token' with a long JWT token and 'Content-Type' with the value 'application/x-www-form-urlencoded'. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format:

```
{  "status": "success",  "message": "Movie updated successfully!!!",  "data": null}
```

. The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 236 ms'. The Windows taskbar at the very bottom shows the system clock as 03:14 AM on 30-04-2018.

http://localhost:3000/movies/5ae63c5027bbe422cce696a3

The screenshot shows the Postman application interface. The top bar includes a 'NEW' button, a 'Runner' button, an 'Import' button, and a 'Builder' button. A notification banner at the top states: 'Chrome apps are being deprecated. Download our free native apps for continued support and better performance. Learn more'. The left sidebar shows a 'Collections' tab with a search filter and a list of collections: 'node-rest-api-jwt' (4 requests) and 'Postman Echo' (21 requests). The 'node-rest-api-jwt' collection is expanded, showing four POST requests: 'index', 'create new user', 'authenticate', and 'create movie'. The 'create movie' request is selected. The main panel shows the details of the 'create movie' request. The request method is 'PUT' and the URL is 'http://localhost:3000/movies/5ae6398627bbe422cce696a1'. The request body is set to 'x-www-form-urlencoded' and contains two key-value pairs: 'name' with value 'Iron man123' and 'released_on' with value '2018-05-12'. The response status is '200 OK' and the time taken is '236 ms'. The response body is displayed in JSON format:

```
{
  "status": "success",
  "message": "Movie updated successfully!!!",
  "data": null
}
```

authenticate create movie create new user index + ... No Environment

Filter History Collections

All Me Team

node-rest-api-jwt 4 requests

POST index

POST create new user

POST authenticate

POST create movie

Postman Echo 21 requests

create movie Examples (0)

PUT http://localhost:3000/movies/5ae6398627bbe422cce696a1 Params Send Save

Authorization Headers (2) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary

Key	Value	Description
name	Iron man123	
released_on	2018-05-12	

New key Value Description

Body Cookies Headers (6) Test Results Status: 200 OK Time: 236 ms

Pretty Raw Preview JSON Save Response

```
1 {
2   "status": "success",
3   "message": "Movie updated successfully!!!",
4   "data": null
5 }
```

http://localhost:3000/movies/5ae63c5027bbe422cce696a3

- DELETE

<http://localhost:3000/movies/5ae63c5027bbe422cce696a3>

The screenshot shows the Postman application interface. On the left, the 'Collections' tab is active, showing a collection named 'node-rest-api-jwt' with 4 requests. The 'create movie' request is selected. The main panel shows the details of the 'create movie' request, which is a DELETE request to the URL 'http://localhost:3000/movies/5ae63c5027bbe422cce696a1'. The 'Body' tab is selected, showing a table with two rows: 'name' with value 'Iron man123' and 'released_on' with value '2018-05-12'. The 'Send' button is visible. Below the request details, the 'Test Results' tab is active, showing a JSON response:

```
{  "status": "success",  "message": "Movie deleted successfully!!!",  "data": null}
```

. The status is '200 OK' and the time is '205 ms'. The Windows taskbar is visible at the bottom.

authenticate create movie create new user index + ... No Environment

Filter History Collections

All Me Team

node-rest-api-jwt 4 requests

POST index

POST create new user

POST authenticate

POST create movie

Postman Echo 21 requests

create movie Examples (0)

DELETE http://localhost:3000/movies/5ae63c5027bbe422cce696a1 Params Send Save

Authorization Headers (2) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary

Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/> name	Iron man123			
<input checked="" type="checkbox"/> released_on	2018-05-12			
New key	Value	Description		

Body Cookies Headers (6) Test Results Status: 200 OK Time: 205 ms

Pretty Raw Preview JSON Save Response

```
1 {
2   "status": "success",
3   "message": "Movie deleted successfully!!!",
4   "data": null
5 }
```

Type here to search

03:15 AM 30-04-2018

Conclusion

In this tutorial we learned to develop REST API with simple CRUD operation, hashing user plain password using bcrypt and JWT token. If have any queries/suggestion please leave a comment and keep following for more such tuts

You can find complete code on github

