APIs

Introduction:: APIs:: Why and What

- We will study the workings of API from the server-side. We will also understand the process of making our API.
- Whenever we want two devices to communicate with minimalist resources, we use API { Application Programming Interface }.
- When we use an API, two devices communicate using a particular format. The most popular is the JSON.
- We are not sending HTML and receiving it back because we want our communication to be fast.
- Frameworks like Angular, VueJs, React are examples where we are using API to interact between server and client-side { framework }.

The Postman

- Though API is used when we are interacting using a frontend framework, we will be using **Postman** which will act as our frontend framework.
- Download **Postman** using the given link below according to your OS.
- Create an account on **Postman.**
- Just like a form being submitted, we can use the form data to be sent over through Postman.
- We will be able to submit the form assuming the device we are using is an Android device and how the data would be communicated in an API when the form is submitted in the key and value that will be simulated over here.

EXTRA:
You can check out the link below to understand more about ajax -
https://www.postman.com/

Setting Up The Directory Structure

- The first part of creating an API is to set up the directory structure so that the code which sends the response and accepts the request in an API format is there in separate files.
- The controller folder will have an API folder that contains all the controllers that are interacting in JSON format.
- Routes will also have an API folder.
- V1 { old version } and V2 { latest version } - Let's say the users are using the android application, we suddenly make some changes to the database. We would want that whenever a new set of users install the application they install the latest version. But we also do not want to bug the old users to change their version of the application. This is where we can use the concept of versions of API.
- Create one folder inside the controllers folder and routes folder named **{ API }.**
- Create a new **V1** folder for each **API** folder inside the controllers and route folder.
- API folders will have their **index** files. Each version also has its index files.

{ routes/index.js }

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

const router = express.Router();
const homeController = require('../controllers/home_controller');

console.log('router loaded');


router.get('/', homeController.home);
router.use('/users', require('./users'));
router.use('/posts', require('./posts'));
router.use('/comments', require('./comments'));


router.use('/api', require('./api'));

// for any further routes, access from here
// router.use('/routerName', require('./routerfile'));


module.exports = router;
```

{ routes/API/index.js }

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

const router = express.Router();



module.exports = router;
```

Rendering an API

- We have to make our first API. For that, we need to define routes and controller actions for the routes.

{ routes/API/index.js }

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

const router = express.Router();

router.use('/v1', require('./v1'));

module.exports = router;
```

{ routes/API/V1/index.js }

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

const router = express.Router();

router.use('/posts', require('./posts'));

module.exports = router;
```

- Inside controllers API V1, create a new file { **posts_api.js** }

{ **controllers/API/V1/posts_api.js** }

```
module.exports.index = function(req, res){
    return res.json(200, {
        message: "List of posts",
        posts: []
    })
}
```

.

- Inside V1 in the routes folder, create a new file { **posts.js** }.

{ **routes/API/V1/posts.js** }

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

const router = express.Router();
const postsApi = require("../../../controllers/api/v1/posts_api");


router.get('/', postsApi.index);



module.exports = router;
```

Playing With APIs

- We will be sending data from the database in the API and try to delete the post without authentication and authorization. We will render the list of posts that are inside the database. In the following step, we will delete one of the posts.
- The places where we have returned a flash message or the places we were redirecting the user to some other page that will be replaced by returning JSON.
- Right now we are using unauthorized and unauthenticated requests.

{ **controllers/API/V1/posts_api.js** }

```
const Post = require('../../../models/post');
const Comment = require('../../../models/comment');
module.exports.index = async function(req, res){
    let posts = await Post.find({})
        .sort('-createdAt')
        .populate('user')
        .populate({
            path: 'comments',
            populate: {
                path: 'user'
            }
        });
    return res.json(200, {
        message: "List of posts",
        posts: posts
    })
}
module.exports.destroy = async function(req, res){
    try{
        let post = await Post.findById(req.params.id);
        // if (post.user == req.user.id){
            post.remove();
            await Comment.deleteMany({post: req.params.id});
            return res.json(200, {
                message: "Post and associated comments deleted successfully!"
            });
        // }else{
        //     req.flash('error', 'You cannot delete this post!');
        //     return res.redirect('back');
        // }
    }catch(err){
        console.log('********', err);
        return res.json(500, {
            message: "Internal Server Error"
        });
    }
}
```

{ **routes/API/V1/posts.js** }

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

const router = express.Router();
const postsApi = require("../../../controllers/api/v1/posts_api");


router.get('/', postsApi.index);
router.delete('/:id', postsApi.destroy);



module.exports = router;
```

Authentication With APIs:: JWT { JSON Web Tokens }

- Whenever we are sending some data as a user to the server in the form of a request, the server checks whether that data matches a user in the database.
    - If it matches a user, then the server generates a token or an encrypted key, stores it somewhere, and sends it back to the browser in the cookie.
- For every request that the browser makes, the cookies have the generated authentication token and the server establishes the identity of the user using that token.
- APIs do not have cookies. For that, we have to store the authentication token somewhere else.
- While using JWT, we don't need to store anything in the database.
- Whenever we are sending in the details of the user, his identity is verified. The token generated has three parts separated by dots that are JWT JSON web tokens. { **Header** - What algorithm is used to encrypt JSON web tokens, payload - It contains the encrypted information, **Signature** - It is composed by some algorithm applied on the combination of header and payload}.
- These tokens are passed to the applications on android or mobile devices or any other frontend framework.
- Once this token is sent to the frontend framework, it stores the token in the local storage.
- The token contains all the necessary information. Hence, we will send that token in the header whenever we are sending back some requests The server in return establishes the identity.

Setting Up Passport JWT

- We will use **Passport** here as it is the core library for authentication.
- We need to install the passport jwt strategy using the command **{ npm install passport-jwt }** in the command line**.**
- We need to mention every strategy that is being used in the **{ index.js }** file of the application.
- Import passport jwt strategy in the **{ index.js }** file.
- Create a new file inside the **config** folder { **passport-jwt-strategy.js** }.
- Import passport and passport-jwt strategy inside the { **passport-jwt-strategy.js** } file.
- We will be storing complete user information inside the payload information in an encrypted format.

{ **passport-jwt-strategy.js** }

```
const passport = require('passport');
const JWTStrategy = require('passport-jwt').Strategy;
const ExtractJWT = require('passport-jwt').ExtractJwt;

const User = require('../models/user');


let opts = {
    jwtFromRequest: ExtractJWT.fromAuthHeaderAsBearerToken,
    secretOrKey: 'codeial'
}


passport.use(new JWTStrategy(opts, function(jwtPayLoad, done){

    User.findById(jwtPayLoad._id, function(err, user){
        if (err){console.log('Error in finding user from JWT'); return;}

        if (user){
            return done(null, user);
        }else{
            return done(null, false);
        }
    })

}));

module.exports = passport;
```

EXTRA:
You can check out the link below to understand more about ajax -
http://www.passportjs.org/packages/passport-jwt/

Creating a Token

- Earlier we were signing in and some cookies were set in the background by passport.
- Now we will explicitly create JSON web tokens and send them using a library called JSON web tokens.
- Install JSON web tokens using the command { **npm install jsonwebtoken** } in the terminal.
- This is used to generate tokens and then the **passport** is capable of decrypting those encrypted token using the library.
- Create a new file inside the API folder inside the controllers folder named as **{ users_api.js }.**
- Whenever a username and password are received we need to find that user and generate the JSON web token corresponding to that user using **async-await** style.

{ controllers/API/V1/users_api.js }.

```
const User = require('../../../models/user');
const jwt = require('jsonwebtoken');


module.exports.createSession = async function(req, res){

    try{
        let user = await User.findOne({email: req.body.email});

        if (!user || user.password != req.body.password){
            return res.json(422, {
                message: "Invalid username or password"
            });
        }

        return res.json(200, {
            message: 'Sign in successful, here is your token, please keep it safe!',
            data:  {
                token: jwt.sign(user.toJSON(), 'codeial', {expiresIn:  '10000'})
            }
        })

    }catch(err){
        console.log('********', err);
        return res.json(500, {
            message: "Internal Server Error"
        });
    }
}
```

{ routes/API/V1/users.js }.

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

const router = express.Router();
const usersApi = require('../../../controllers/api/v1/users_api');


router.post('/create-session', usersApi.createSession);

module.exports = router;
```

{ routes/API/V1/index.js }

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

const router = express.Router();

router.use('/posts', require('./posts'));
router.use('/users', require('./users'));

module.exports = router;
```

- We have defined that we will be finding JWT from the header and it would be decrypted using "**codeial**" because when we were generating the token we are using "**codeial**" as the key we were finding the user and passport will automatically set it as req. user in the request.

Authentication And Authorization

- We are setting the session to be false because we do not want the session cookie to be generated.

{ **controllers/API/V1/users_api.js** }

```
const User = require('../../../models/user');
const jwt = require('jsonwebtoken');


module.exports.createSession = async function(req, res){

    try{
        let user = await User.findOne({email: req.body.email});

        if (!user || user.password != req.body.password){
            return res.json(422, {
                message: "Invalid username or password"
            });
        }

        return res.json(200, {
            message: 'Sign in successful, here is your token, please keep it safe!',
            data:  {
                token: jwt.sign(user.toJSON(), 'codeial', {expiresIn:  '100000'})
            }
        })

    }catch(err){
        console.log('********', err);
        return res.json(500, {
            message: "Internal Server Error"
        });
    }
}
```

{ **controllers/API/V1/posts_api.js** }

```
module.exports.destroy = async function(req, res){

    try{
        let post = await Post.findById(req.params.id);

        if (post.user == req.user.id){
            post.remove();

            await Comment.deleteMany({post: req.params.id});



            return res.json(200, {
                message: "Post and associated comments deleted successfully!"
            });
        }else{
            return res.json(401, {
                message: "You cannot delete this post!"
            });
        }

    }catch(err){
        console.log('********', err);
        return res.json(500, {
            message: "Internal Server Error"
        });
    }

}
```

- We have set up the passport-jwt-strategy. Now we have to check whether the authentication is working or not. For that we need - { authentication token for the user, Id of the posts which need to be deleted }.
- We need to find the id of the post to be deleted.

{ **routes/API/V1/posts.js** }

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

const router = express.Router();
const passport = require('passport');
const postsApi = require("../../../controllers/api/v1/posts_api");


router.get('/', postsApi.index);
router.delete('/:id', passport.authenticate('jwt', {session: false}), postsApi.destroy);



module.exports = router;
```

Summarizing APIs

- We have studied how authentication with API works.
- We have an understanding of how the API works { get, post, delete request } and simulated using **postman**.