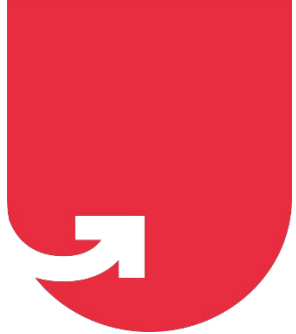
upGrad

#RahoAmbitious



Full Stack Software Development

upGrad

Course: Server-Side Development Using Node.js, Express.js and MongoDB

Lecture on: Express.js Modules

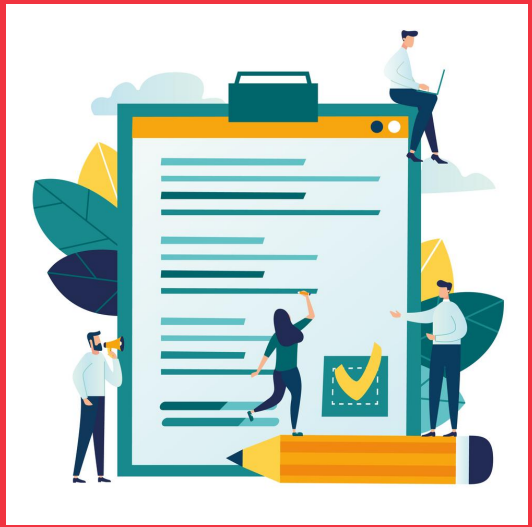
Instructor: Rocky Jagtiani

In the previous class, we covered...

- We got introduced to the concept of Express.js
- We understood the application of Express.js as middleware

Today's Agenda

- Handling Form Data
- Serving Static Files
- Cookies
- Sessions
- Error Handling



upGrad

Handling Form Data

Handling Form Data

upGrad

- The web's main function is communication
- Express.js provides us with the tools to understand the client's request and how to respond appropriately
- Basically, Express.js stores client data in two places:
  - ***request.query*** (for GET request)
  - request.body (for POST requests)
- On the client side, it is ideal to use the POST method for form submission because most browsers place limits on the length of the query and additional data is lost
- The difference between coding via GET and POST is that POST requires ***body-parser*** module. Body-parser is a middleware that makes POST data available to the request.body

Handling Form Data

upGrad

request.body will not work without the body-parser middleware.

Refer To: 3_form_data

To handle HTTP POST request in Express.js version 4 and above, you need to install a middleware module called body-parser.

Earlier, the middleware was a part of Express.js, but now you have to install it separately.

This body-parser module parses the JSON, the buffer, the string and the url-encoded data submitted using HTTP POST request.

Install body-parser using NPM as shown below.

npm install body-parser --save

Hands-on Exercise 1 (20 mins)

upGrad

- Create a project folder named express_practice_2
- Initialise the project
  npm init
- Install required packages
  npm install express --save
  npm install body-parser --save
- Code a simple index.html file to accept the first and the last name of a candidate. Keep the method as post
- On submission, fetch the first name and the last name of the candidate and print them to the browser

Hands on Exercise 1 (20 mins)

upGrad

Optional:
Instead of listening on port 3000, hear the request on port 5000

When sending HTML files as a response to the browser, one may get the following error: **TypeError**: *path must be absolute or specify root to res.sendFile*

This is when you code res.sendFile('index.html');

Soln: res.sendFile(__dirname +'/index.html');

Refer To: express_practice_2

Serving Static Files

Serving Static Files

upGrad

- Static files, like images, are files that clients download from the server
- Create a new directory, 'public' in your project structure
- Express.js, by default, does not allow you to serve static files. You need to enable it using the following built-in middleware:

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

Serving Static Files

upGrad

Express looks up the files relative to the static directory. So, the name of the static directory is not a part of the URL.

The root route is now set to your public directory. So, all the static files that you load will consider 'public' as root.

Refer To: 5_display Images

upGrad

Cookies

Cookies

upGrad

- Cookies are small text files that are sent by the server to the client as a piggy-backed attachment to the response

- Cookies are stored on the client side in the browser memory. Every time the user loads the website back, that is, hits the URL, this cookie is sent with the request. This helps keep track of user's actions

- Every cookie object has an expiry time. It could range from as low as session* time to a few years

**Note: Session time means that the cookie would live till you work on the site. As soon as you close the browser, the cookie is destroyed and cleared from the browser memory.*

Cookies

upGrad

- The numerous uses of HTTP cookies include:
  - Session management
  - Personalisation (Recommendation systems)
  - User tracking
- To use cookies with Express, you need ***cookie-parser*** middleware
- To install this middleware, use the following code:

```
npm install cookie-parser --S
```

- ***cookie-parser*** is a middleware that parses cookies attached to the client request object

Cookies

upGrad

```
// Syntax to create a cookie object
// cookies hold everything as key:value pairs
app.get('/', (req, res) => {
   res.cookie('edtech', 'upgrad').send('cookie set');   //Sets edtech = upgrad
});
```

```
// To check if the above cookie is set on the client side, that is, browser, run the following code:
console.log(document.cookie);
```

Creating cookies with expiration time
Just add property ‘expire’ along with the expiry time to the cookie

For example,

```
//Expires after 360000 ms from the time it is set.
res.cookie(name, 'upgrad', {expire: 360000 + Date.now()});
```

Another way to set expiration time is using ‘maxAge’ property. Using this property, you can provide relative time instead of absolute time

```
//This cookie also expires after 360000 ms from the time it is set.
res.cookie(name, 'value', {maxAge: 360000});
```

Deleting Existing Cookies

```
// To delete a cookie, use the clearCookie function
app.get('/clear_cookie_upgrad', function(req, res){
   res.clearCookie('upgrad');
   res.send('cookie upgrad cleared');
});
```

Refer To: cookie_practice

Signed and Unsigned Cookies

upGrad

- A signed cookie is simply a signature attached to the cookie.
- So, the cookie will still be visible to the client. However, it has a signature. So, the server side code can detect if the client modified the cookie or not.
- If the signature does not match, then it will give an error.

Signed and Unsigned Cookies

upGrad

// To create a signed cookie code syntax, use the following code:
res.cookie('name', 'value', {signed: true})

Note: Instead of app.use(cookieParser()), use app.use(express.cookieParser('SomePrivateKey'));

// To access a signed cookie, use the signedCookies object of req (that is, request) object:
req.signedCookies['name']

Browsers are supposed to support at least **4096 bytes per cookie**. To ensure that you do not exceed the limit, don't exceed the size of 4093 bytes per domain.

Poll 1 (15 sec)

All the static files, such as images, audio or video, that you serve are stored under _____ dir and all static files that you load will consider public as ___.

1. public
2. root
3. resources
4. package.json

Poll 1 (Answer)

All the static files, such as images, audio or video, that you serve are stored under _____ dir and all static files that you load will consider public as ___.

1. **public**
2. **root**
3. resources
4. package.json

Sessions

Sessions

upGrad

- HTTP is stateless; so, every request is unique. Although two or more consecutive requests could come in from the same client browser.

- So, in order to associate a request to any other request, you need a way to store user data between HTTP requests.

- Cookies and URL parameters are both suitable ways to exchange data between the client and the server. But, they are both readable and stored on the client side. You want something that can be stored and managed on the server side only.

- Sessions solve exactly this problem. You assign the client an ID and it makes all further requests using that ID. The information associated with the client is stored on the server linked to this ID.

Sessions

upGrad

- To use sessions, you need express-session module:
  npm install express-session --S

- The session middleware handles all the things, that is, creating the session, setting the session cookie and creating the session object in request object.

- Whenever you make a request from the same client again, you will have their session information stored with you (given that the server was not restarted).

***Note**: You can add more properties to the session object.*

Refer To: session_practice

Poll 1 (15 sec)

The maximum cookie size supported by a browser per domain is:

1. 4093 bytes
2. 1024 bytes
3. 2048 bytes
4. 4096 bytes

Poll 1 (Answer)

Maximum cookie size supported by a browser per domain is:

1. 4093 bytes
2. 1024 bytes
3. 2048 bytes
4. **4096 bytes**

Error Handling

Error Handling

upGrad

- Error handling in Express.js is done by writing a regular middleware function.
- Error handling middleware is defined in the same way as other middleware functions, except that error-handling functions MUST have four arguments instead of three – err, req, res and next.

// For example, to send a response on any error, you can use the following code:

```
app.use(function(err, req, res, next) {
   console.error(err.stack);
   res.status(500).send('Something broke!');
});
```

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   //Create an error object and pass it to the next function
   var err = new Error("dash-dash went wrong");
   next(err);
});

/*
 * other route handlers and middleware here
*/

//An error handling function or middleware
app.use(function(err, req, res, next) {
   res.status(500);
   res.send("dash-dash went wrong.")
});

app.listen(3000);
```

Error Handling

upGrad

- For error handling, you use the **next(err)** function.
- A call to **next(err)** function skips all middlewares and matches you to the next error handler for that route. Let's understand this through an example.

```
var express = require('express');
var app = express();

app.get('/', function(req, res){
   //Create an error object and pass it to the next function
   var err = new Error("dash-dash went wrong");
   next(err);
});

/*
 * other route handlers and middleware here
*/

//An error handling function or middleware
app.use(function(err, req, res, next) {
   res.status(500);
   res.send("dash-dash went wrong.")
});

app.listen(3000);
```

Refer To: errorhandling_practice

upGrad

Project Work - Checkpoint 6

Checkpoint 6

upGrad

- Code the routes file **app/routes/tutorial.routes.js.**
- Follow the routes and methods as discussed and decided in Project checkpoint 2.

Methods	URLs	Actions
GET	api/tutorials	Get all online courses
GET	api/tutorials/:id	Get online course by id (Each online course has a unique ID)
POST	api/tutorials	Add a new online course (one at a time) - **Only by ADMIN**
PUT	api/tutorials/:id	Update an online course by id - **Only by ADMIN**
DELETE	api/tutorials/:id	Delete an online course by id - **Only by ADMIN**
DELETE	api/tutorials	Remove all online courses - **Only by ADMIN**
GET	api/tutorials/published	Find all published online courses
GET	api/tutorials?title=[kw]	Find all online courses the title of which contains ‘kw’
GET	api/tutorials?category=[kw]	Find all online courses based on category

Checkpoint 6

upGrad

- Code the routes file **app/routes/user.routes.js.** Follow the routes and methods as discussed and decided in Project checkpoint 2.

Methods	URLs	Actions
POST	api/signup	Create a USER object and save it to USER collection
POST	api/login	Verify login credentials (email_id and password). If verified, set ***IsLoggedIn*** to true
POST	api/logout	Logouts the user based on their **id.** On logging out, *IsLoggedIn* is set to false

Refer To: app/routes/user.routes.js

Checkpoint 6

upGrad

Code the routes file **app/routes/enrollment.routes.js.** Follow the routes and methods as discussed and decided in Project checkpoint.

Methods	URLs	Actions
POST	api/enroll	Create a ENROLLMENT object and save it to ENROLLMENT collection

Refer To: app/routes/enrollment.routes.js

Checkpoint 6

upGrad

- Install Chrome extension for swagger inspector (https://swagger.io/tools/swagger-inspector/). This extension is required to test the APIs.

- To use POSTMAN or Swagger UI, follow the following steps:
  - Choose the HTTP method: GET, POST, PUT or DELETE
  - Type the URL of the API
  - Provide parameters in one of the two following ways:
    - For GET request, provide a query parameter, that is, concatenated to the URL. For example. *http://localhost:3000/api/tutorials?title=js*

upGrad

- For POST request, provide body parameters in JSON format.

  For example.
  http://localhost:3000/api/tutorials/ and the JSON inputs would be supplied through the body parameters

```
{
   "title" : "DevOps and javascript",
   "description" : "Table of contents - Primary Topics",
   "published" : true,
   "category" : "DevOps"
}
```

Checkpoint 6

upGrad

- All GET HTTP methods are supposed to fetch data. Make sure that you check their o/p, so that when you integrate the backend with the frontend, you are sure that your APIs are not at fault.

  For example,
  GET HTTP Request
  http://localhost:3000/api/tutorials?title=js
  would return all Courses having **js** anywhere in the Course title.

Homework

In this project, you are supposed to scan the index.js file code and answer the following questions, rather than code.

Q1. Explain the components of authHeader, an object of request.headers.authorization? (Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication)

Q2. When request.session.user is not set, then what does the code do?

Installation required for running the project

npm install express --S
npm install cookie-parser --S
npm install express-session --S
npm install session-file-store --S

Refer To: session_management_project

Doubt Clearance (5 mins)

Key Takeaways

- We learnt to handle form data, cookies, session data and errors in Express.js

The following tasks are to be completed after today's session:

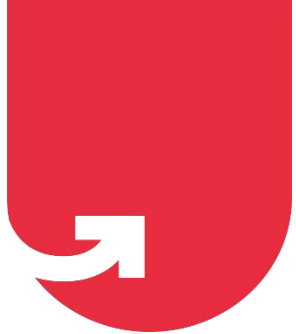
Homework
MCQs
Coding Questions
Course Project - Checkpoint 7

In the next class, we will discuss...

- We will learn to develop an end-to-end application

upGrad

#RahoAmbitious



Thank you!