ExpressJS

Lesson 1 - Introduction

What is Express.js

Express.js is a web application framework for Node.js. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node.js.

Why Express.js

Express.js is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Features of Express.js

- Allows to set up middlewares to respond to HTTP Requests.
- Defines a routing table which is used to perform different actions based on HTTP Method and URL.
- Allows to dynamically render HTML Pages based on passing arguments to templates.
- Allows to create a RESTful API.
- Allows to add your own custom logic to the middleware, creating a robust API.
- Allows to integrate with a database of your choice.
- Allows to use your own view engine.
- Allows to use your own template engine.

What is a RESTful API

REST stands for REpresentational State Transfer. It is an architectural style for designing networked applications. REST is an approach to communications that is often used in the development of Web services.

Why RESTful API

RESTful APIs are designed around resources, which are any kind of object, data, or service that can be accessed by the client. A resource has an identifier, which is a URI that uniquely identifies that resource.

Features of RESTful API

- RESTful APIs are designed around resources, which are any kind of object, data, or service that can be accessed by the client.
- A resource has an identifier, which is a URI that uniquely identifies that resource.
- RESTful APIs use a set of HTTP verbs to implement the concept of CRUD (create, read, update, and delete) operations.
- RESTful APIs are stateless. This means that all the information necessary to handle the request must be contained within the request itself. Session state is therefore kept entirely on the client.
- RESTful APIs are cacheable. This means that the Web server generating the response can indicate whether the response should be cached by the client.

Installation

To install Express.js, open a terminal and run the following command:

```
npm install express --save
```

Hello World

To create a simple Express.js application, create a file named app.js and add the following code:

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

app.get('/', (req, res) => {
  res.send('Hello World!');
});

app.listen(1377,()=>{
  console.log('Server started on port 1377');
})
```

Referenties

- Express
- Express - Routing
- Express - Middleware
- Express - Error handling
- Express - Built-in middleware
- Express - API reference
- Express - API reference - Router
- Express - API reference - Request
- Express - API reference - Response
- Express - API reference - Application
- Express - API reference - Error handling
- Express - API reference - Middleware

```

```

ExpressJS

Lesson 2 - Hello world

Create a simple Express.js application

To create a simple Express.js application, create a project folder and enter the following command in the terminal:

```
npm init -y
npm install express --save
npm install body-parser --save
npm install cookie-parser --save
```

what is package.json in node.js

The package.json file is a manifest file for Node.js projects. It is used to store metadata about the project. This file is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

modify script in package.json

```
"scripts": {
    "start": "node app.js", // production
    "dev":"nodemon app.js" // development
  },
```

what is body-parser

Body-parser is a piece of express middleware that reads a form's input and stores it as a javascript object accessible through `req.body`.

what is cookie-parser

Cookie-parser is a piece of express middleware that reads a form's input and stores it as a javascript object accessible through `req.cookies`.

Create a simple Express.js application

create a file named app.js and add the following code:

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const cookieParser = require('cookie-parser');

app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser())

app.get('/',(req,res)=>{
  res.send('hello world')
})

app.listen(1337, () => console.log('Example app listening on port 1337!'))
```

Run the application

To run the application, enter the following command in the terminal:

```
- For development
npm run dev

- For production
npm start
```

What is nodemon

Nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

Referencies Expressjs Installation

ExpressJS

Lesson 3 - Request and Response

What is a request

A request is an HTTP request sent by a client to a server. It consists of a request line, request header, and an optional message body.

What is a response

A response is an HTTP response sent by a server to a client. It consists of a status line, response header, and an optional message body.

Request object properties

- req.app - returns the instance of the express application that is using the middleware
- req.baseUrl - returns the URL path on which a router instance was mounted
- req.body - returns the body of the request
- req.cookies - returns the cookies sent by the request
- req.fresh - returns true if the response has not been modified
- req.hostname - returns the hostname of the request
- req.ip - returns the remote IP address of the request
- req.ips - returns an array of IP addresses specified in the X-Forwarded-For header
- req.method - returns the HTTP method of the request
- req.originalUrl - returns the original URL of the request
- req.params - returns an object containing properties mapped to the named route "parameters"
- req.path - returns the path part of the request URL
- req.protocol - returns the protocol string of the request, either http or (for TLS requests) https
- req.query - returns an object containing a property for each query string parameter in the route

Response object properties

- res.app - returns the instance of the express application that is using the middleware
- res.append - appends the specified HTTP response header field with value to the response
- res.attachment - sets the Content-Disposition header to "attachment"
- res.cookie - sets the cookie name to value
- res.clearCookie - clears the cookie name
- res.download - prompts a file to be downloaded
- res.end - ends the response process
- res.format - performs content-negotiation on the Accept HTTP header on the request object, when present
- res.get - gets the specified HTTP response header field
- res.json - sends a JSON response
- res.jsonp - sends a JSON response with JSONP support
- res.send - sends a response of various types
- res.sendFile - transfers the file at the given path

- res.set - sets the specified HTTP response header field to value
- res.status - sets the HTTP status for the response
- res.type - sets the Content-Type HTTP header to the MIME type as determined by mime.lookup()

Request object methods

- req.accepts() - checks if the specified content types are acceptable, based on the request's Accept HTTP header field
- req.get() - gets the specified HTTP request header field
- req.is() - checks if the incoming request contains the "Content-Type" HTTP header field, and it contains any of the MIME types specified by the type argument
- req.param() - returns the value of parameter name when present or value when not

Response object methods

- res.append() - appends the specified HTTP response header field with value to the response
- res.attachment() - sets the Content-Disposition header to "attachment"
- res.cookie() - sets the cookie name to value
- res.clearCookie() - clears the cookie name
- res.download() - prompts a file to be downloaded
- res.format() - performs content-negotiation on the Accept HTTP header on the request object, when present
- res.get() - gets the specified HTTP response header field

Request params vs query examles

- http://localhost:1337/user/1?name=John
- http://localhost:1337/user/1?name=John&age=20

```
req.params = { id: 1 }
req.query = { name: 'John', age: '20' }
```

Response examples

```
res.send('hello world')
res.json({ name: 'John' })
res.status(404).send('Not found')
res.status(500).send({ error: 'something blew up' })
```

Referencies ExpressJS MDN

ExpressJS

Lesson 4 - Http requests and response in Express.js

GET request

A GET request is used to request data from a specified resource.

POST request

A POST request is used to send data to a server to create/update a resource.

PUT request

A PUT request is used to send data to a server to create/update a resource.

DELETE request

A DELETE request is used to delete a resource.

PATCH request

A PATCH request is used to update a resource.

Example of GET request

```
app.get('/',(req,res)=>{
  res.send('hello world')
})
```

Example of POST request

```
Send an post request:
curl -X POST -H "Content-Type: application/json" -d
'{"username":"root",password:"123123"}' http://localhost:1337
```

```
app.post('/',(req,res)=>{

const {username,password} = req.body
  res.json({username,password})

})
```

Example of PUT request

```
Send an put request:
curl -X PUT -H "Content-Type: application/json" -d
'{"username":"root",password:"123123"}' http://localhost:1337/1
```

```
app.put('/:id',(req,res)=>{
    const {username,password} = req.body
    res.json({username,password})
})
```

Example of DELETE request

```
Send an delete request:
curl -X DELETE http://localhost:1337/1
```

```
app.delete('/:id',(req,res)=>{
    res.json({message:'deleted'})
})
```

Example of PATCH request

```
Send an patch request:
curl -X PATCH -H "Content-Type: application/json" -d
'{"username":"root",password:"123123"}' http://localhost:1337/1
```

```
app.patch('/:id',(req,res)=>{
    const {username,password} = req.body
    res.json({username,password})
})
```

Referencies HTTP request methods

ExpressJS

Lesson 5 - Routing & middleware in Express.js

What is routing

Routing is the process of selecting a path for traffic in a network or between or across multiple networks. Routing is performed by specialized hardware and software, and may occur in a router, a layer 3 switch, or a host.

```
app.get('/')
app.get('/login')
app.get('/register')
app.get('/profile/:id')
app.get('/profile/dashboard')
app.get('*') // when no route is matched
```

What is middleware

Middleware is software that provides common services and capabilities to applications outside of what's offered by the operating system. Middleware is frequently used for integration of different applications and systems.

```
function isAuth(req,res,next){
  if(req.isAuthenticated()){
    return next() // if user is authenticated, continue to the next
middleware
  }
  res.redirect('/login') // redirect to login page
}
```

What is next() function

The next() function is a method available on the response object (res) of an Express.js app. It is used to pass control to the next middleware function after the current middleware function has completed its job.

```
app.get('/profile/:id',isAuth,(req,res)=>{
  res.send('profile')
})
```

What is express.Router()

The express.Router class is a small wrapper around the app object that allows us to create modular, mountable route handlers. A Router instance is a complete middleware and routing system; for this reason, it is often referred to as a “mini-app”.

```
const router = express.Router()

router.get('/',(req,res)=>{
  res.send('home')
})

router.get('/login',(req,res)=>{
  res.send('login')
})

router.get('/register',(req,res)=>{
  res.send('register')
})

router.get('/profile/:id',(req,res)=>{
  res.send('profile')
})

router.get('/profile/dashboard',(req,res)=>{
  res.send('dashboard')
})

router.get('*',(req,res)=>{
  res.send('404')
})

module.exports = router
```

Referencies Routing Middleware Express.Router

ExpressJS

Lesson 6 - File upload using multer

File upload is the process of sending files from a client to a server. In this lesson we will learn how to upload files using Express.js.

Multer is a middleware for handling multipart/form-data, which is primarily used for uploading files. It is written on top of busboy for maximum efficiency.

Install multer

```
npm install multer
```

Create a simple Express.js application

```
var express =   require("express");
var multer  =   require('multer');
var app =   express();
var storage =   multer.diskStorage({
  destination: function (req, file, callback) {
    callback(null, './uploads');
  },
  filename: function (req, file, callback) {
    callback(null, file.originalname);
  }
});
var upload = multer({ storage : storage}).single('myfile');


app.post('/upload',function(req,res){
    upload(req,res,function(err) {
        if(err) {
            return res.end("Error uploading file.");
        }
        res.end("File is uploaded successfully!");
    });
});

app.listen(2000,function(){
    console.log("Server is running on port 2000");
});
```

Referencies https://www.tutorialspoint.com/nodejs/nodejs_uploading_files.htm https://www.npmjs.com/package/multer https://www.npmjs.com/package/multer#diskstorage

ExpressJS

Lesson 7- JWT Authentication

What is JWT?

JWT stands for JSON Web Token. It is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is digitally signed using JSON Web Signature (JWS) and/or encrypted using JSON Web Encryption (JWE). JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA.

What is JWT Authentication?

JWT authentication is a method for an HTTP server to verify that a user is who they say they are. It is a stateless authentication mechanism, meaning that the user state is not stored on the server. Instead, every request to the server must be accompanied by a token, which the server uses to validate the authenticity of the request. The token is usually sent in the Authorization header using the Bearer schema. The token itself is a cryptographically signed JSON Web Token (JWT).

How JWT Authentication works

1. The user provides their credentials to the server.
2. The server verifies the credentials and creates a JWT token.
3. The server sends the token to the client.
4. The client stores the token and sends it to the server in the Authorization header with every request.
5. The server verifies the token and processes the request.
6. The server sends a response to the client.
7. The client receives the response.
8. The client can now access protected routes, services, and resources.
9. The client can also refresh the token before it expires.
10. The client can also log out by removing the token.

How to use JWT Authentication with ExpressJS

1. Install the following packages:
    - `npm install jsonwebtoken`
    - `npm install express-jwt`
    - `npm install bcryptjs`

1. Create a file named `auth.js` and add the following code:

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs'); // for password hashing
const jwtSecret = 'your_jwt_secret'; // This has to be the same key used in
the JWTStrategy
const jwtExpirySeconds = 300; // 5 minutes

const users = []; // array of users

// Login endpoint
app.post('/login',(req,res)=>{

  const {username, password} = req.body;

  // Ideally you'll fetch this from the db
  // Idea here was to show how jwt works with simplicity
  const user = users.find(u => { return u.username === username &&
u.password === password });

  if (user) {
    // from now on we'll identify the user by the id and the id is the only
personalized value that goes into our token
    const token = jwt.sign({ id: user.id }, jwtSecret, {
      algorithm: "HS256",
      expiresIn: jwtExpirySeconds
    });
    console.log('token:', token);
    res.cookie('token', token, { maxAge: jwtExpirySeconds * 1000 });
    res.status(200).send();
  } else {
    res.status(401).send();
  }

})
```

```
// Register endpoint

app.post('/register',(req,res)=>{
  const {username, password} = req.body;

  // Ideally you'll fetch this from the db
  // Idea here was to show how jwt works with simplicity
  const user = users.find(u => { return u.username === username });

  if (user) {
    res.status(401).send();
  } else {
    // Password hashing
    const salt = bcrypt.genSaltSync(10);
    const hash = bcrypt.hashSync(password, salt);

    users.push({
      id: users.length,
      username,
      password: hash
    })
    res.status(200).send();
  }

})
})

// Logout endpoint

app.post('/logout',(req,res)=>{
  res.clearCookie('token');
  res.status(200).send();
})
```

```
// Protected endpoint

function requireAuth(req,res,next){
  const token = req.cookies.token;

  if (token) {
    jwt.verify(token, jwtSecret, (err, decodedToken) => {
      if (err) {
        console.log(err.message);
        res.status(401).send();
      } else {
        console.log(decodedToken);
        next();
      }
    });
  } else {
    res.status(401).send();
  }
}

app.get('/protected', requireAuth, (req, res) => {
  res.status(200).send("Access granted: Protected content");
});
```

References JWT JWT Authentication JWT Authentication with ExpressJS JWT Authentication with NodeJS JWT Authentication with NodeJS and MongoDB