REST api with Express & MySQL

Joost Klaver
joost@pxlwidgets.com
Team lead development

Guido Carucci
guido.carucci@pxlwidgets.com

https://endeavourgroup.co
https://pixelindustries.com

Endeavour-group

10 companies - 130 employees - 2 offices

- DBOM - online marketing, SEO/SEA
- 3sixtyfive - influencer marketing with Youtube vloggers
- Brewwwers - wordpress en shopify websites
- Superlab - online brand strategy & communication
- HelloDialog - e-mail marketing & marketing automation
- PXLWidgets - apps, websites, e-commerce systems
- Online Koplopers - online marketing detachering
- Coster Copy - online copy-writing
- Hollandsch Welvaren - online campaigns, communication & applications for Dutch municipalities
- Suepar Video's - video creation, strategy and production

1 of the fastest growing companies in the region, stagebedrijf van het jaar, regular meetups about all internet related subjects

Endeavour-group

Example Github repository and workshopper

A few things to install or clone before we start:

- expressworks workshopper:

```
> npm install -g expressworks
```

(see the rest of the very interesting workshoppers at: https://nodeschool.io/#workshoppers)
After install you run it with:

```
> expressworks
```

- Example express mysql app:

```
> git clone https://github.com/quadrofolio/techgrounds-api.git
```

Agenda

- Introduction to REST, CRUD and request - response cycle
- HTTP methods/verbs
- Setting up Express
- Testing API routes
- Path & query parameters
- Retrieving data with your API
- Getting Hands-On with expressworks workshopper and example git repository

REST

Representational State Transfer (REST)

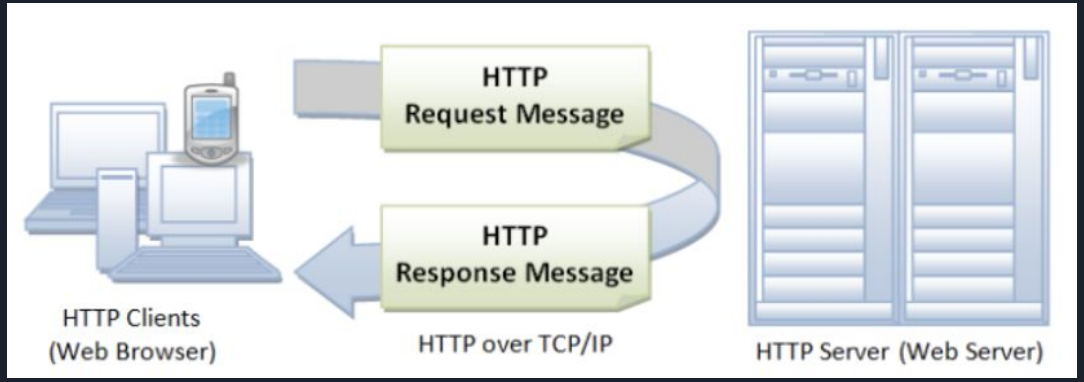
HTTP verbs: GET, HEAD, POST, PUT, PATCH, DELETE, CONNECT, OPTIONS and TRACE

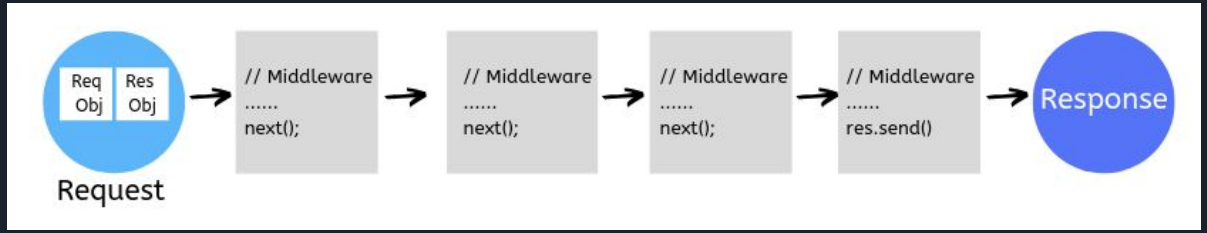
CRUD = CREATE READ UPDATE DELETE

GET = READ retrieve a resource
POST = CREATE insert new resource
PUT = UPDATE update by replacing complete resource
DELETE = DELETE remove a resource

(PATCH = update by patching certain properties of the resource)

Request → Middleware → Response cycle





REST in express: Setting up 1/2

You can use 2 methods:

1. create your own files, we'll do this one together
2. generate express app:

```
npx express-generator
express --view=pug express-api
```

REST in express: Setting up 2/2

```
> mkdir express-api && cd express-api
> npm init && npm install -S express body-parser && npm install nodemon -D
> code .

const express = require('express');
const bodyParser = require('body-parser');
const app = express();
// parse requests of content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));
// parse requests of content-type - application/json
app.use(bodyParser.json());

app.get('/', (req, res) => {
  return res.send('Received a GET HTTP method');
});
app.post('/', (req, res) => {
  return res.send('Received a POST HTTP method');
});
app.put('/', (req, res) => {
  return res.send('Received a PUT HTTP method');
});
app.delete('/', (req, res) => {
  return res.send('Received a DELETE HTTP method');
});
app.listen(3000, () =>
  console.log(`Example app listening on port 3000!`),
);
```

REST in express: testing with CURL

```
app.get('/', (req, res) => {
  return res.send('Received a GET HTTP method');
});
app.post('/', (req, res) => {
  return res.send('Received a POST HTTP method');
});
app.put('/', (req, res) => {
  return res.send('Received a PUT HTTP method');
});
app.delete('/', (req, res) => {
  return res.send('Received a DELETE HTTP method');
});
app.listen(3000, () =>
  console.log(`Example app listening on port 3000!`),
);

In terminal:
> curl http://localhost:3000 -> Received a GET HTTP method
> curl -X POST http://localhost:3000 -> Received a POST HTTP method
> curl -X PUT http://localhost:3000 -> Received a PUT HTTP method
> curl -X DELETE http://localhost:3000 -> Received a DELETE HTTP method
```

REST in express: URI’s & Resources

```
app.get('/users', (req, res) => {
  return res.send('GET HTTP method on user resource');
});
app.post('/users', (req, res) => {
  return res.send('POST HTTP method on user resource');
});
app.put('/users', (req, res) => {
  return res.send('PUT HTTP method on user resource');
});
app.delete('/users', (req, res) => {
  return res.send('DELETE HTTP method on user resource');
});
```

REST in express: path/route & query parameters

```
Path/Route parameters:

app.put('/users/:userId', (req, res) => {
  return res.send(
    `PUT HTTP method on user/${req.params.userId} resource`,
  );
});

Query parameters:

app.delete('/users?userId=1', (req, res) => {
  return res.send(
    `DELETE HTTP method on user/${req.query.userId} resource`,
  );
});
```

REST in express: retrieving data

There are several ways to store data:

1. in memory JS objects (array, object), we'll do this one together
2. json or JS files with data (jsondata = require('jsonfile.json' don't do this when your data needs to change)
3. databases:
    a. SQLite
    b. MySQL
    c. Mongo
    d. Postgress
    e. etc.

REST in express: data from JS objects

```
let users = {
  1: {
    id: '1',
    username: 'Joost Klaver',
  },
  2: {
    id: '2',
    username: 'Guido Carucci',
  },
};
let messages = {
  1: {
    id: '1',
    text: 'Hello World',
    userId: '1',
  },
  2: {
    id: '2',
    text: 'Ciao World',
    userId: '2',
  },
};
```

```
let users = { ... };
let messages = { ... };

app.get('/users', (req, res) => {
  return res.send(Object.values(users));
});
app.get('/users/:userId', (req, res) => {
  return res.send(users[req.params.userId]);
});
```

REST in express: data from JS objects

To get a specific object you need to be able to find it.

So the Objects need a unique identifier = ID to be found on

In our data:

```
let messages = {
  1: {
    id: '1',
    text: 'Hello World',
    userId: '1',
  },
```

```
const message = messages[messageId];
```

REST in express: data from JS objects

```
> npm install uuid

// In server.js:
import uuidv4 from 'uuid/v4';

app.post('/messages', (req, res) => {
  const id = uuidv4();
  const message = {
    id,
    text: req.body.text,
    userId: req.body.userId
  };
  messages[id] = message;
  return res.send(message);
});
```

But how do we create new ID’s (POST http verb) ?

REST in express: data from JS objects

```
curl -X POST -H "Content-Type:application/json" http://localhost:3000/messages -d
'{"text":"Hi again, World", "userId": 1}'
```

```
let messages = {
  1: {
    id: '1',
    text: 'Hello World',
    userId: '1',
  },
  2: {
    id: '2',
    text: 'By World',
    userId: '2',
  },
  'df3f785c-57d2-11ea-82b4-0242ac130003': {
    id: 'df3f785c-57d2-11ea-82b4-0242ac130003',
    text: 'Hello again, World',
    userId: '1',
  },
};
```

REST in express: data from JS objects

Remove a message from the messages object:

```
app.delete('/messages/:messageId', (req, res) => {
  const {
    [req.params.messageId]: message,
    ...otherMessages
  } = messages;
  messages = otherMessages;
  return res.send(message);
});
```

```
curl -X DELETE http://localhost:3000/messages/1
```

```
let messages = {
  2: {
    id: '2',
    text: 'By World',
    userId: '2',
  },
  'df3f785c-57d2-11ea-82b4-0242ac130003': {
    id: 'df3f785c-57d2-11ea-82b4-0242ac130003',
    text: 'Hi again, World'
  },
};
```

REST in express: better solution is data from MySQL

```
> npm i mysql
```

```
var mysql = require('mysql')
var connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root',
  database: 'techgrounds'
})

// message model:
const Message = function(message) {
  this.text = message.text;
  this.userId = message.userId;
};
```

Create a database: techgrounds

Create 2 tables:

- users:
  - id,
  - username
- messages:
  - id,
  - text,
  - userId

(See: techgrounds.sql included in repository)

REST in express: better solution is data from MySQL

```
// inside GET route:
...
var messageId = req.param.messageId;
connection.connect()
connection.query(`SELECT * FROM messages WHERE id = ${messageId}`, function (err,res) {
  if (err) throw err;
  console.log('The found message: ',  res[0]);
})
connection.end()

// inside POST route:
...
var newMessage = new Message({
  text: req.body.text,
  userId: req.body.userId
});
connection.connect()
connection.query('INSERT INTO messages SET ?', newMessage, function (err,res) {
  if (err) throw err;
  console.log('The new message: ', { id: res.insertId,  ...newMessage });
})
connection.end()
```

REST in express: better solution is data from MySQL

Types of queries:

```
GET:    'SELECT * FROM messages WHERE id = :id'

POST:   'INSERT INTO messages SET ?'

UPDATE: 'UPDATE messages SET ? WHERE id = :id'

DELETE: 'DELETE FROM messages WHERE id = :id'
```

REST in express: further reading

- Express: https://expressjs.com/
- Node: https://nodejs.org/
- Nodeschool workshoppers: https://nodeschool.io/#workshoppers
- Express MySQL tutorial: https://bezkoder.com/node-js-rest-api-express-mysql/
- Express MySQL tutorial async: https://time2hack.com/creating-rest-api-in-node-js-with-express-and-mysql/
- Understanding express middleware by example: https://developer.okta.com/blog/2018/09/13/build-and-understand-express-middleware-through-examples

Now let’s code!

Open terminal and type:

> expressworks