Web 3

Express

HO
GENT

Express

- Snel, minimalistisch framework voor Node.js
- Maakt het ons makkelijk om robuuste webapplicatie te maken met Node.js
- Maakt het ons ook makkelijk om een **REST API** te schrijven
- Focus ligt op performantie
- Vele populaire frameworken zijn gebaseerd op Express bvb. Feathers, LoopBack, Sails, ...

https://expressjs.com

HO GENT

Express

Installatie

- Het Express pakket wordt lokaal geïnstalleerd in een project via NPM
- We kunnen ook gebruik maken van de express generator.
- In elk project waar je Express wilt gebruiken lokaal (zonder generator) dien je dit commando uit te voeren

```
$ npm install express     OF     yarn add express
```

HO GENT

Express

Express generator

- Express generator zal een projectstructuur aanbrengen in ons project
- Maakt wel nog gebruik van de oude JavaScript syntax

```
$ npx express-generator --no-view --git <MAP NAAM>
```

- Probeer na te gaan wat er in iedere file gebeurt die aangemaakt is geweest

HO GENT

Express

Hello World

- Onze eerste Express applicatie, natuurlijk beginnend met Hello World
- Nieuw Node.js project dus met het Express package als een dependency

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

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

app.listen(3000);
```

HO GENT

Express

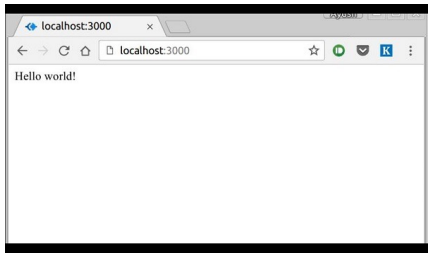
Hello World (2)

- Onze eerste Express applicatie, natuurlijk beginnend met Hello World
- Nieuw Node.js project dus met het Express package als een dependency

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

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

app.listen(3000);
```



HO GENT

Express

Routing

- Web frameworks leveren resources zoals HTML pagina's, scripts, afbeeldingen, etc. op verschillende routes
- In Express kunnen we een route definiëren door middel van volgende methode

```
app.method(path, handler);
```

HO GENT

Express

Routing (2)

```
app.method(path, handler);
```

- De `method` property wordt niet expliciet gebruikt maar wel 1 van de HTTP verbs (GET, POST, PUT, …)
- De `path` parameter, is de route waar de request naartoe gestuurd wordt
- De `handler` is de callback functie die opgeroepen wordt wanneer de juiste methode overeen komt met de route
  - De `handler` callback functie heeft twee parameters de `req` (request) en de `res` (response)

HO GENT

Express

Routing (3)

- We kunnen dus meerdere methodes gebruiken voor dezelfde route zoals in het voorbeeld hieronder

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

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

app.post('/hello', (req, res) => {
   res.send("POST methode op /hello");
});

app.listen(3000);
```

HO GENT

Express

Routing (4)

- Express beschikt ook over een speciale all methode om alle types van de HTTP verbs op te vangen op een bepaalde route
- Wordt meestal gebruikt om middleware te definiëren (Later besproken)

```
app.all('/hello', (req, res) => {
  res.send("HTTP methodes maken geen verschil");
});
```

HO GENT

Express

Routing (5)

- Deze methodes zouden moeilijk te onderhouden zijn als we heel veel routes gaan definiëren in onze index file
- Om de routes te definiëren buiten onze index file om gaan we gebruik maken van **`Express.Router`**

routers.js

```
const router = express.Router();

router.get('/', (req, res) => {
  res.send('GET route');
});

module.exports = router;
```

HO GENT

Express

Routing (6)

- Om de routes te definiëren buiten onze index file om gaan we gebruik maken van **`Express.Router`**

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

const routes = require('./routes.js');

app.use('/', routes);

app.listen(3000);
```

index.js

HO GENT

Express

Routing (7)

- De meest gebruikte en tevens ook de ondersteunende HTTP methodes met Express
- **GET**
  - Requests gebruik makend van GET kunnen enkel maar data ontvangen
- **POST**
  - Requests gebruik makend van POST kunnen nieuwe data toevoegen
- **PUT**
  - Requests gebruik makend van PUT kunnen bestaande data aanpassen, als de data nog niet bestaat dan kan deze nieuwe aanmaken
- **DELETE**
  - Requests gebruik makend van DELETE kunnen bestaande data verwijderen

HO GENT

Express

Request Object

- In de callback van een `get`, `post`, `delete` methode hebben we steeds toegang tot een request object
- Stelt de HTTP request voor en heeft toegang tot properties zoals de parameters, de body, HTTP headers, ...
- Toegang tot verschillende properties:
  - `req.body`
  - `req.params`
  - `req.query`
  - `req.ip`
  - …

https://expressjs.com/en/api.html#req

HO GENT

Express

Request Object Body

- De `body` property bevat key-value paren van de data die meegestuurd is met bvb. een POST request
- Standaard is dit `undefined` en wordt maar pas ingevuld als we gebruik maken van body parser middleware (later meer hierover - `express.json() / express.urlencoded()`)

```
…

app.post('/profile', (req, res) => {
   console.log(req.body);
   res.json(req.body);
});

…
```

HO GENT

Express

Request Object Params

- De `params` property bevat de data die toegewezen is aan de benoemde route parameters
- Bijvoorbeeld de route `/user/:name` hebt, dan is name als een property beschikbaar als `req.params.name`

…

```
app.get('/profile/:name', (req, res) => {
   console.log(req.params.name);
   res.send(req.params.name);
});
```

…

HO GENT

Express

Request Object Query

- De `query` property bevat de data die toegewezen is voor elke query tekenreeksparameter voor de route
- Bijvoorbeeld de route `/search?color=blue` hebt, dan is color als een property beschikbaar als `req.query.color`

```
…

app.get('/', (req, res) => {
   console.log(req.query.color);
   res.send(req.query.color]);
});

…
```

HO GENT

Express

Request Object IP

- De `ip` property bevat het ip adres van de host die de request verstuurd heeft

```
…

app.get('/', (req, res) => {
   console.log(req.ip);
   res.send(req.ip);
});

…
```

HO GENT

Express

Response Object

- In de callback van een `get`, `post`, `delete` methode hebben we steeds toegang tot een response object
- Stelt de HTTP response voor dat Express terugstuurt bij het ontvangen van een HTTP request
- Toegang tot verschillende methodes:
  - `res.status()`
  - `res.send()`
  - `res.json()`
  - `res.render()`
  - …

https://expressjs.com/en/api.html#res

HO GENT

Express

Response Object status()

- De `status()` methode definieert de HTTP status voor de response die teruggestuurd wordt
- Voorbeeld:

```
res.status(200);          // OK
res.status(404);          // Not Found
res.status(500);          // Internal Server Error
```

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

HO GENT

Express

Response Object send()

- De `send()` methode stuurt de response terug
- De parameter in deze methode is de body en kan van volgende types zijn: `Buffer` object, `String`, `Boolean`, `Array`, `Object`

```
res.send(Buffer.from('whoop'));
res.send({ some: 'json'});
res.send('<p>some html</p>');

res.status(404).send('Sorry, niet gevonden');
```

HO GENT

Express

Response Object json()

- De `json()` methode stuurt een JSON response terug
- De parameter in deze kan elke type zijn dat JSON ondersteund; `Object`, `Array`, `String`, `Boolean`, `Number`, `null`

```
res.json(null);
res.json({ user: 'tobi'});

res.status(500).json({ error: 'message'});
```

HO GENT

Express

Response Object render()

- De `render()` methode wordt gebruikt om een view te renderen (aangemaakt met een template engine)
- De parameter die meegegeven wordt in de render methode is de view, waarvan de HTML string teruggestuurd wordt naar de client.

```
res.render('index');
```

HO GENT

Express

URL Building

- Routes die we nu gedefinieerd hadden waren statische routes of fixed routes
- Bedoeling dat we ook dynamische routes gaan ondersteunen bvb. een dynamische id om een object op te vragen
- Het laat ons toe om parameters kunnen mee te geven aan de URL en onze data te baseren op deze parameters

HO GENT

Express

URL Building (2)

- Voorbeeld van een dynamische route met een ID als parameter:

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

app.get('/:id', (req, res) => {
  res.send('De id is ' + req.params.id);
});

app.listen(3000);
```

HO GENT

Express

URL Building (3)

- Voorbeeld (2) van een dynamische route met een ID en een name als parameter

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

app.get('/things/:name/:id', (req, res) => {
  res.send('De ID is ' + req.params.id +
             'en naam: ' + req.params.name);
});

app.listen(3000);
```

HO GENT

Express

URL Building (4)

- Om de mogelijkheid van URL parameters te beperken kunnen we gebruik maken van Pattern Matched Routes door gebruik te maken van een REGEX
- Voorbeeld de ID parameter moet een getal zijn van 5 cijfers

```
app.get('/things/:id([0-9]{5})', (req, res) => {
  res.send('ID: ' + req.params.id);
});
```

HO GENT

Express

URL Building (5)

- Als geen enkele request matcht met de routes, krijg je de volgende foutmelding “Cannot GET <your-request-route>”
- Oplossing door een fallback route met een wildcard *****
- OPGELET: Dit moet wel de laatste route zijn, anders wordt deze telkens aangeroepen

```
app.get('*', (req, res) => {
  res.status(404).send('Niets gevonden');
});
```

HO GENT

Express

Middleware

- Middleware functies zijn functies die toegang hebben tot het request (req) object, het response (res) object, en de next() middleware methode in de request-reponse cyclus van de applicatie
- Middleware wordt vooral gebruikt om
  - Code uit te voeren
  - Wijzigingen aan te brengen aan het request en het response object
  - Beëindigen van de request-response cyclus
  - De volgende middleware functie aanroepen in de stack

HO GENT

Express

Middleware
Application level

- Application level middleware is middleware die over de hele applicatie gebruikt wordt
- De middleware functie wordt dus aan de `app` instantie gebonden door middel van de `use()` methode

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

app.use((req, res, next) => {
   console.log("Time: ", Date.now());
   next();
});
```

HO GENT

Express

Middleware
Application level (2)

- Kan ook beperkt worden tot een specifiek pad of alle onderliggende paden

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

app.use('/things', (req, res, next) => {
   console.log("Time: ", Date.now());
   next();
});

app.get('/things', (req, res) => {
   res.send('Things');
});
```

HO GENT

Express

Middleware
Application level (3)

- Kan ook gebruikt worden om het request object te wijzigen

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

app.use((req, res, next) => {
   req.requestTime = Date.now();
   next();
});

app.get('/things', (req, res) => {
   res.send('Requested at: ' + req.requestTime);
});
```

HO GENT

Express

Middleware
Router level

- Router level middleware is middleware die gebonden is aan een instantie van express.Router()
- De middleware functie wordt dus aan de `express.Router()` instantie gebonden door middel van de `use()` methode

```
const express = require('express');
const app = express();
const router = express.Router();

router.use((req, res, next) => {
   console.log("Time: ", Date.now());
   next();
});
```

HO GENT

Express

Middleware
Third-party

- Third party middleware kan ook gebruikt worden om middleware functies toe te voegen aan onze applicatie
- Kan gebruikt worden om
  - De body van requests te parsen die een payload hebben zoals een POST request met data in, door middel van body-parser
  - Cookies te parsen, door middel van cookie-parser
  - …
- Deze third party middleware wordt geïnstalleerd met NPM

HO
GENT

Express

Middleware
Third-party (2)

- Voorbeeld body-parser middleware die de payload van bijvoorbeeld een POST request met data in zal parsen.

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

app.use(express.urlencoded({ extended: false}));
app.use(express.json());
```

https://github.com/expressjs/body-parser#readme

HO GENT

Express

Templating

- Een template engine kan je statische bestanden in de express applicatie gebruiken.
- Tijdens de runtime zal de template engine de variabelen in de template vervangen door actuele waarden, en transformeert de template naar een HTML bestand dat dan op zijn beurt verzonden wordt naar de client.
- Maakt het makkelijker om een webpagina te maken via een template engine
- Express ondersteunt volgende template engines
  - Pug, Mustache, EJS.

https://expressjs.com/en/guide/using-template-engines.html

HO GENT

Express

Project Structuur

- Express heeft geen vaste project structuur. Dit betekent dus dat je zelf een structuur in je code moet brengen
- We spreken ook niet van ”één correcte manier” om dit te doen
- Meestal is het wel een goede structuur om uw routes, en middlewares apart in een map te zetten (routes, middlewares)

HO GENT

Opdracht

- **Express Web Server**
  - Maak een simpele webserver met Express. Maak gebruik van een apart route bestand waarin de routes staan van de applicatie.
  - De bedoeling is dat je verschillende **GET** routes aanmaakt:
    - **‘/’** => Geeft een p element terug met de tekst “home page”
    - **‘/test’** => Geeft een p element terug met de tekst “test page”
    - **‘/test/123’** => Geeft een p element terug met de tekst “ID: 123” OPGELET GA NA DAT DE ID EEN GETAL IS MET 3 CIJFERS (REGEX)
    - **‘/search?q=blue’** => Geeft een p element terug met de tekst “SEARCHED: blue“

HO GENT

Opdracht (2)

- De bedoeling is ook dat je deze **POST** route aanmaakt
  - **‘/test’** => Stuurt data via JSON op naar de route (firstName, lastName). En Express stuurt deze data terug als een json response naar de client met een extra property server: true. (Je kan dit uittesten met Postman). LET OP: Je zal gebruik moeten maken van de body-parser middleware hiervoor, anders zal je niets terug krijgen als resultaat
- Als laatste dien je ook nog een fallback route te moeten hebben om alles op te vangen als het niet juist is
  - **‘*’** => Geeft een p element terug met tekst ”NIET GEVONDEN” en ook een status 404

HO GENT

Opdracht (3)

- Maak ook een middleware functie aan die bij elke request een console.log zet met de huidige tijd en het ip adres van de client
- Maak nog een laatste middleware aan waarbij je een requestTime property toevoegt aan het request object. In deze property zet je de huidige tijd. Je maakt dan ook een nieuwe route aan met ‘/time’ waarbij je de property requestTime uit het request object gaat terugsturen

HO GENT

Opdracht (4)

- **Express ES Syntax**
  - Maak een nieuw project aan via de express generator. Vorm de express-generator bestanden om naar ES syntax. Dus verander alle oude syntax, hoe we gezien hebben in de tweede les.
  - LET DUS OP in alle volgende opdrachten, als je van de express generator gebruik wilt maken dat je de syntax steeds aanpast (variabelen, functies, …)
- **!!! Bij het indienen van de opdracht mag je eerst de node_modules map verwijderen in het project, dit wordt toch steeds opnieuw aangemaakt bij npm install**

HO GENT