

# NodeJS – Webapps met express



# Wat is Express?

- Express is ook bekend als een 'middleware-framework'
  - Onderliggend worden er een groot aantal zaken gereld en aangeboden als één functie

Bijvoorbeeld: in kale node.js moet een `http.createServer()` en `.listen()` opgeroepen worden. Binnen express gebruiken we enkel `.listen()`
- De globale opbouw van Express:
  - Express ontvangt een http-request
  - De request wordt langs middleware-functies gestuurd waarmee data bekeken en verwerkt kan worden voordat het de app binnenkomt of verlaat

# Wat is Express

- Meest gebruikte library voor web projecten (standaard)
- Standaard uitwisselingsformaat in Express is JSON
- Beschikt over ingebouwde functie om JSON in de body van een request te lezen en verwerken
  - Geen handmatige parsing meer nodig

# Een express-app maken

- Express installeren
- Routes definiëren
- JSON retour zenden

Download de voorbeelden uit Resources/CH11\_Voorbeelden

# Een express-app maken

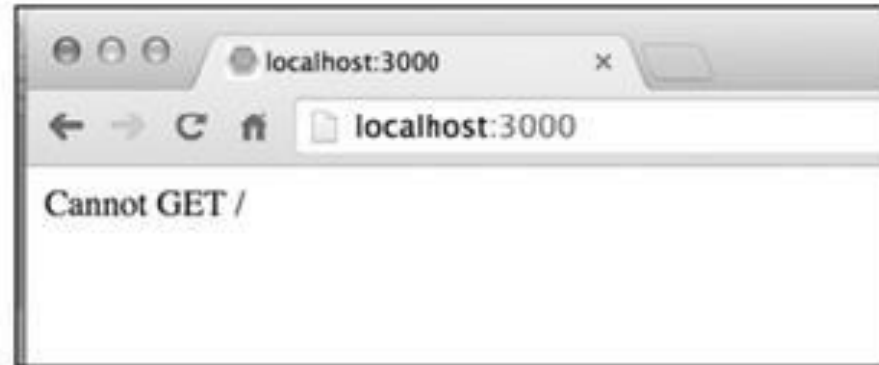
- Maak een nieuwe directory en installeer Express via NPM  
`npm install express`
- Maak in de directory een bestand `server.js` met volgende inhoud:

```
server.js
1  var express = require('express');
2  var app = express();
3
4  app.listen(3000);
5  console.log('Express server gestart op localhost:3000...');
```

# Een express-app maken

- Start de applicatie  
`node server.js`
- Probeer de homepage op te vragen

Omdat nog geen routes of inhoud is gedefinieerd, wordt een eenvoudige foutmelding getoond (zie afbeelding).



Afbeelding 5.3 Met enkele regels code is een Express-webserver gemaakt. Er is echter nog geen inhoud gedefinieerd.

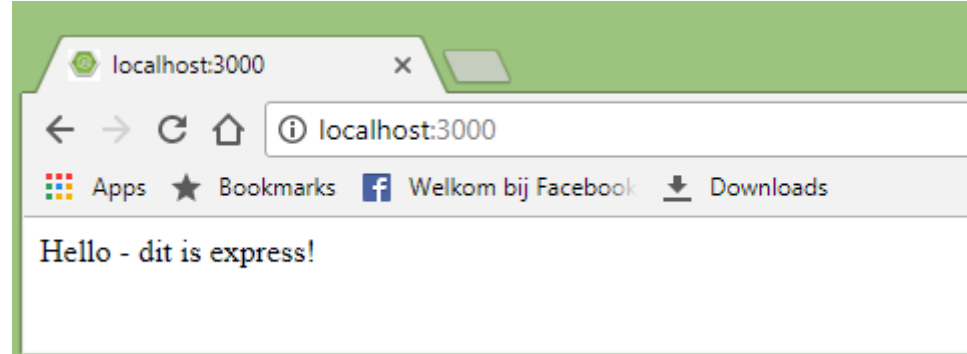
# Een express-app maken

- Bekijk het bestand /1101/server.js
- Routes definiëren
  - Routes moeten gedefinieerd worden om te bepalen hoe een inkomend verzoek moet worden afgehandeld.
- Breid de server uit als volgt:

```
server.js
1  var express = require('express');
2  var app = express();
3
4  app.get('/', function(req, res){
5      res.send('Hello - dit is express!');
6  });
7
8  app.listen(3000);
9  console.log('Express server gestart op localhost:3000...');
```

# Een express-app maken

- Vraag de homepage opnieuw op
  - Let ook op de Response headers in de developer tools. Express heeft automatisch de headers opgesteld





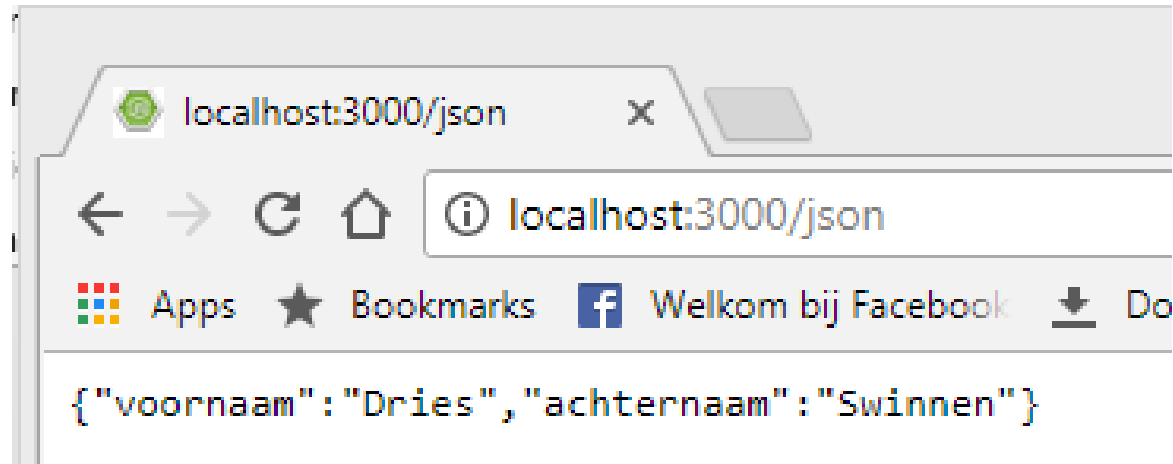
# Een express-app maken

- JSON retour zenden
  - Het response object uit de callback methode heeft meer mogelijkheden als enkel .send
  - Methodes zoals .json zijn mogelijk
- Voorzie volgende route:

```
var persoon = {  
  voornaam: 'Dries',  
  achternaam: 'Swinen'  
};  
app.get('/json', function(req, res){  
  res.json(persoon);  
});
```

# Een express-app maken

- Start de applicatie en test de nieuwe route
  - Je merkt dat het opnieuw uitvoeren van node server.js nodig is om wijzigingen te verwerken. Dit kan automatisch met de module [nodemon](#)
- De conversie naar JSON is automatisch toegepast door Express (dubbele aanhalingstekens)

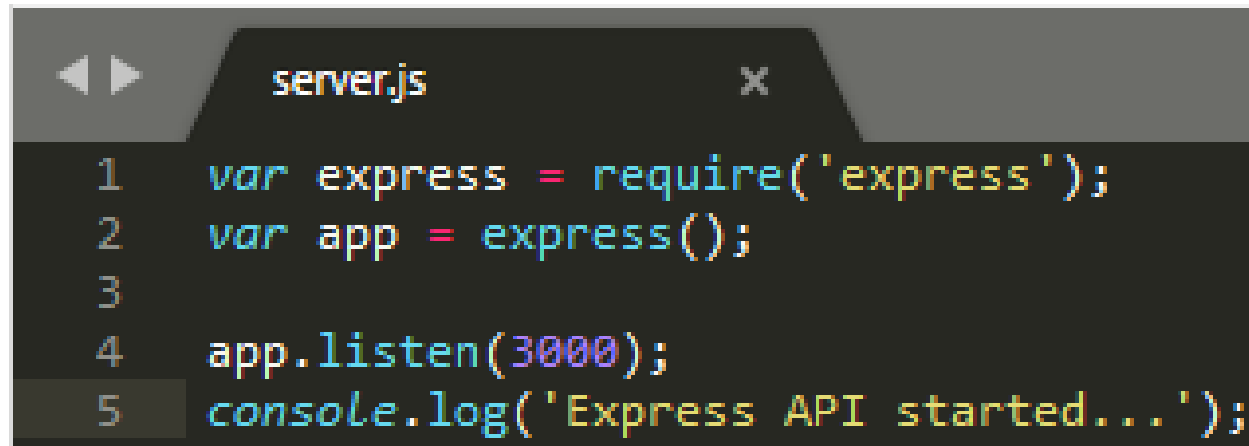


# Een express-API maken

- Bekijk de bestanden in de map /1102\_API
- Aan de hand van routes en het teruggeven van JSON kunnen we een heel eenvoudige API opzetten.
- Volgende routes worden voorzien in onze app:
  - De route /api moet korte instructies voor het gebruik van de API teruggeven
  - De route /api/auteurs moet een JSON-tabel met auteurs retourneren
  - De route /api/boeken moet een JSON-tabel met boeken retourneren

# Een express-API maken

- Maak een minimale express app in een nieuwe folder en installeer express:



```
server.js
1  var express = require('express');
2  var app = express();
3
4  app.listen(3000);
5  console.log('Express API started...');
```

# Een express-API maken

- Voorzie een route voor de instructie van de API:

```
app.get('/', function(req, res){  
  var msg = '<h1>Express API</h1>';  
  msg += '<p>Gebruik \\api\\auteurs voor een lijst met auteurs.</p>';  
  msg += '<p>Gebruik \\api\\boeken voor een lijst met boeken.</p>';  
  res.send(msg);  
});
```

- De uitleg wordt nu getoond bij het laden van de app op <http://localhost:3000>

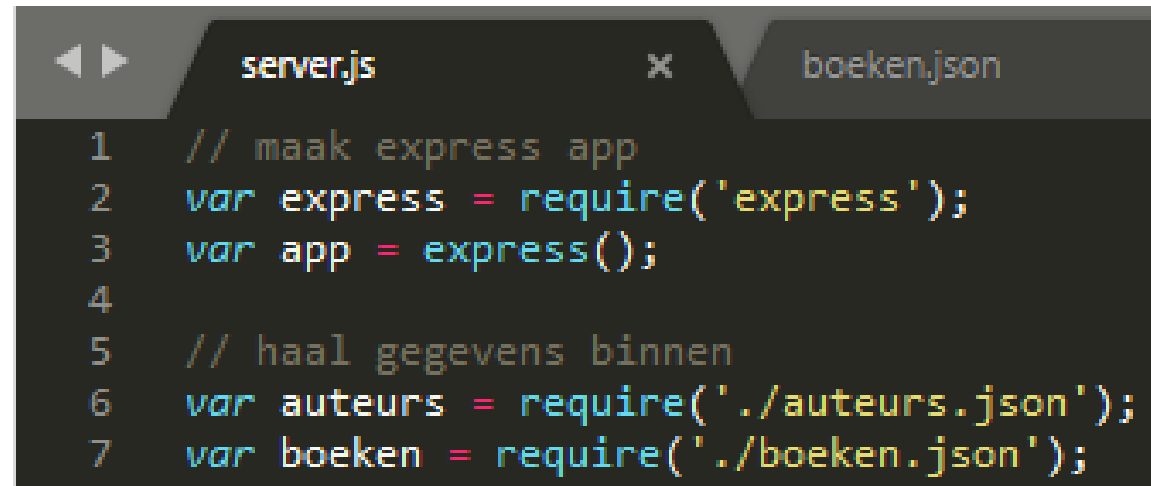
# Een express-API maken

- Data invoegen met require()
  - In onze site zijn 2 JSON bestanden voorzien (zie H12/1202\_API/)
    - Auteurs.json
    - Boeken.json

```
[
  {
    "id": 1,
    "titel": "Web Development Library - AngularJS",
    "auteur": "Peter Kassenaar",
    "ISBN": "9789059407879"
  },
  {
    "id": 2,
    "titel": "Focus op Fotografie: Portretfotografie",
    "auteur": "Pieter Dhaeze",
    "ISBN": "9789059408128"
  },
  {
    "id": 3,
    "titel": "App Development Library - Apps maken voor Android",
    "auteur": "Michiel de Rond",
    "ISBN": "9789059408395"
  },
  {
    "id": 4,
    "titel": "Ontdek de Windows-tablet",
    "auteur": "Bob van Duuren",
    "ISBN": "9789059406186"
  }
]
```

# Een express-API maken

- Data invoegen met require()
  - Voorzie volgende regels boven de declaratie van de route:



```
1 // maak express app
2 var express = require('express');
3 var app = express();
4
5 // haal gegevens binnen
6 var auteurs = require('./auteurs.json');
7 var boeken = require('./boeken.json');
```

# Een express-API maken

- Vervolgens voegen we de routes toe aan onze applicatie

```
app.get('/api/auteurs', function(req, res){  
  res.json(auteurs);  
});  
app.get('/api/boeken', function(req, res){  
  res.json(boeken);  
});
```

- Start de applicatie opnieuw (node server.js of via nodemon)



# Een express-API maken

- Routeparameters gebruiken kan via de routes
  - Opbouw lijkt op die van angular:

```
app.get('/api/boeken/:idx', function(req, res){  
  });
```

# Een express-API maken

- De parameter opvragen kan met `req.params.<paramnaam>`
- Gebruiken de javascript functie `forEach` om door alle boeken te lopen

```
app.get('/api/boeken/:idx', function(req, res){
  var id = req.params.idx;
  var gezochtBoek;
  boeken.forEach(function(boek){
    if(boek.id === parseInt(id)){
      gezochtBoek = boek;
    }
  });
  res.json(gezochtBoek);
});
```

# Een express-API maken

- Routeparameters gebruiken

Dit werkt prima, zoals de afbeelding laat zien. Hier is het boek met id 4 opgevraagd.



Afbeelding 5.8 Met routeparameters zijn dynamische URL's te creëren.

# Statische bestanden serveren

- Bekijk de bestanden in de map /1103\_static
- De opdracht `app.use()` en `express.static()`
  - Statische bestanden worden in de webserver in een public-map gezet en worden aangegeven door één regel code
  - `app.use` betekent dat Express een middleware functie moet uitvoeren op binnenkomende request(s)

```
var auteurs = require('./auteurs.json');
var boeken = require('./boeken.json');

// 3. Stel middleware in voor serveren van statische bestanden (HTML, CSS, images)
app.use(express.static(__dirname + '/public'));

// 4. Start de server.
app.listen(3000);
console.log('Express app gestart op localhost:3000');
```

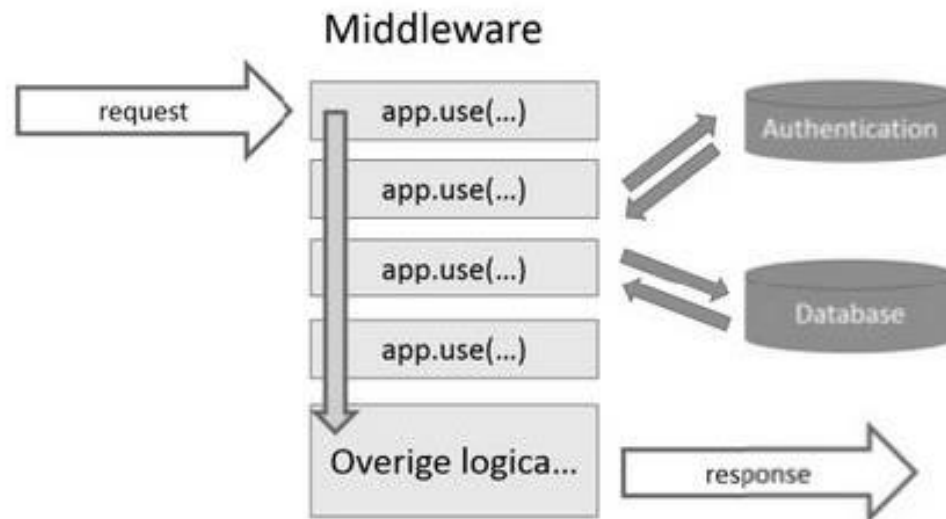
# Statische bestanden serveren

- De HTML site kan nu geladen worden via onze server



# Werken met middleware

- Bekijk de bestanden in de map /1103\_static
- Middleware is een functie die wordt uitgevoerd op het binnenkomende request
  - Resultaat wordt doorgegeven aan de volgende functie in de stack tot er een functie meer is
  - Daarna wordt de response teruggegeven



# Werken met middleware

- `app.use()` wordt gebruikt om middleware functies te definiëren
- Parameters voor `app.use()`
  - Req: het request object
  - Res: het result object
  - Next: de volgende functie in de middleware-keten

```
app.use(function(req, res, next){  
  console.log('Requested file: ', req.url);  
  next();  
});
```

- De methode `next()` moet opgeroepen worden, anders blijft de request hangen

# Werken met middleware

- Verschillende soorten middleware beschikbaar:
  - Application-level, router-level, error-handling, builtin, third-party
- De volgorde van de functies is belangrijk
- Meer informatie over middleware

<http://expressjs.com/guide/using-middleware.html>

- In het voorbeeld wordt de middleware gebruikt als logging:

```
$ node server.js
Express app gestart op localhost:3000
Requested file: /
Requested file: /js/lib/bootstrap/dist/css/bootstrap.min.css
Requested file: /boeken.html
Requested file: /js/lib/bootstrap/dist/css/bootstrap.min.css
Requested file: /index.html
Homepage opgevraagd: 2018-01-13T14:21:10.691Z
Requested file: /js/lib/bootstrap/dist/css/bootstrap.min.css
.....
```



# POST-requests

- Bekijk de bestanden in /1105\_POST
  - Voor GET-requests maakte we gebruik van `app.get()`
  - Voor POST-requests bestaat de functie `app.post()` met dezelfde structuur als `app.get()`
    - Methodes bestaan voor elk type request in Express
  - Extra middleware / [body-parser](#) is vereist voor een vlotte werking
    - Parsed de formulier velden uit de body van een request
- ```
npm install body-parser
```

# POST-requests

- Body-parser toevoegen en configureren
  - Eerst require
  - Daarna via middleware (app.use()) configureren voor gebruik
    - bodyParser.urlencoded of bodyParser.json() afhankelijk van de soort request

```
// Middleware laden voor het parsen van JSON in het request
var bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({
  extended: true
}));
```

# POST-requests

- Vervolgens kan een post request opgevangen worden via `app.post()` zoals hieronder:

```
// Een POST-request verwerken
var user = {};
app.post('/', function (req, res) {
  // verwerk binnenkomende request. We gaan er van uit
  // dat de parameter 'username' en 'email' aanwezig zijn.
  // TODO: error checking!
  console.log(req.body);
  user.username = req.body.username;
  user.email = req.body.email;

  // Echo het user-object naar de client
  res.json(user);
});
```

# POST-requests

- Bij het opstarten van server2.js uit het voorbeeld kan de post ook uitgevoerd worden via <http://localhost:3000/login.html>
- Simuleren van http requests kan via:
  - Via browser plugins/add-ons zoals postman / ARC
  - Via tools zoals Fiddler
  - Via curl

# De functie Router()

- Bekijk de bestanden server.js en routes.js in de map /1106/Router
- Als de applicatie wordt gestart, is de uitvoer naar verwachting
- De hoofdmodule server.js is nu een stuk compacter



# De functie Router()

```
server.js
// Express en externe routes laden
var express = require('express');
var routes = require('./routes');
var app = express();

app.use('/', routes);
app.listen(3000, function () {
  console.log('Express-server gestart op http://localhost:3000');
});
```

```
routes.js
// routes.js - middleware via express.Router()
var express = require('express');
var router = express.Router();

// Dit mag ook in één regel:
// var router = require('express').Router();

// middleware specifiek voor deze router - merk op dat
router.use(function(req, res, next){
  console.log('Router aangeroepen: ', new Date());
  next();
});

// homepage voor deze router.
router.get('/', function(req, res){
  res.send('Homepage van de router');
});

router.get('/about', function(req, res){
  res.send('Over ons - vanuit de router');
});

router.get('*', function(req, res){
  res.send('onbekende route...');
});
module.exports = router;
```

# De functie Router()

- Abstracte routes
  - In voorgaand voorbeeld gebruiken we `app.use('/', routes)` voor de koppeling van de routes aan onze root directory
  - Kunnen dit specifiek maken voor verschillende entry-points:
    - Bv aparte route `/team` waarop de routes van toepassing zijn
    - Enkel `app.use()` aanpassen (zie `1106_router/server2.js`)

```
// De router is nu abstract gemaakt en werkt alleen voor de route \team in de  
// request. Elke andere route is nu ongeldig (of kan via een andere router worden afgehandeld).  
app.use('/team', routes);
```

# De functie Router()

- Abstracte routes

Nu is de 'gewone' homepage niet meer bereikbaar, maar worden de routes alleen toegepast op de route /team in de URL. Dit alles zonder dat u routes.js hebt hoeven aanpassen.





# De functie Router()

- Router gebruiken in de applicatie
  - Bekijk het voorbeeld 1106\_router/server4.js
  - De routes staan in een eigen bestand (./router/index.js)
  - De locatie van de data is aangepast (boeken.json en auteurs.json)

```
// server4.js
var express = require('express');
var bodyParser = require('body-parser');
var routes = require('./router');

var app = express();
app.use(bodyParser.urlencoded({
  extended: true
}));
app.use(express.static(__dirname + '/public'));

app.use('/', routes);
```

# De functie Router()

- Router gebruiken in de applicatie
  - Routes zijn als volgt gedefinieerd

```
// .\router\index.js - bestand met routes voor boeken, auteurs en login.  
var router = require('express').Router();  
var auteurs = require('../data/auteurs.json');  
var boeken = require('../data/boeken.json');  
  
router.get('/api', function (req, res) {  
    //... stuur data.  
});  
  
//... overige routes.  
  
module.exports = router;
```