Programmation Web – Avancé

JavaScript & Node.js

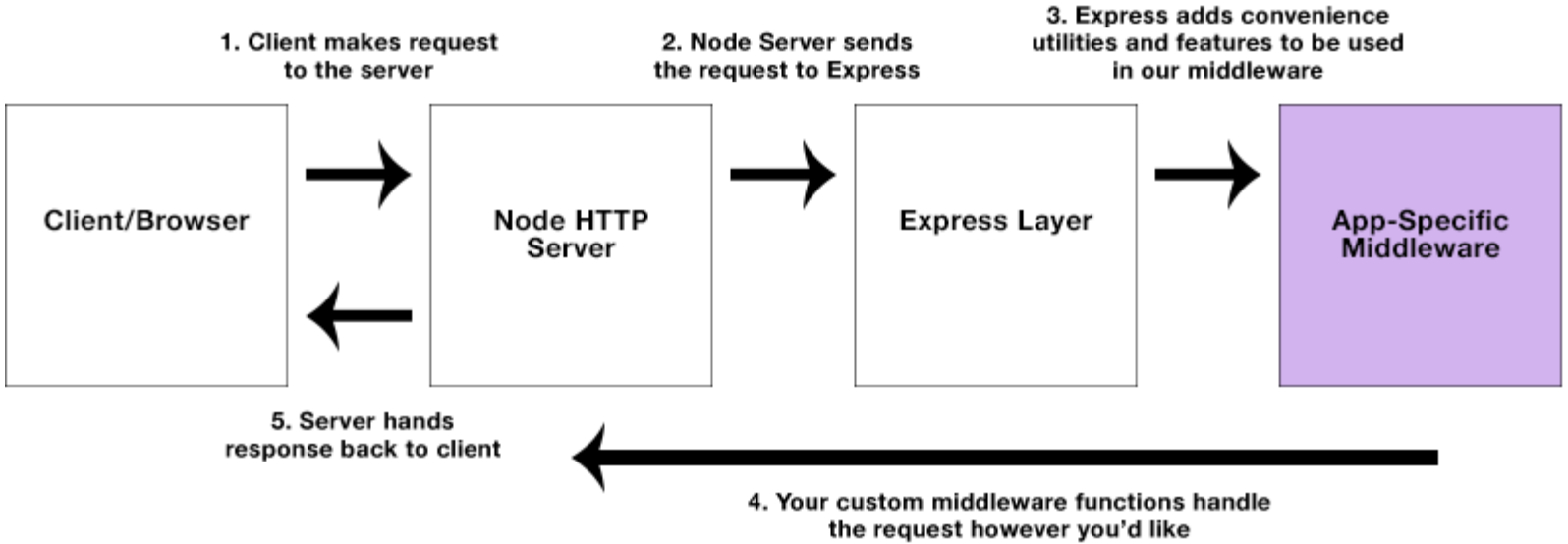
Framework Express pour RESTful APIs

Presentation template by SlidesCarnival

RESTful API via le Framework Express

Accélérer le développement de son backend : le framework Express...

Introduction au framework Express



Introduction à Express [58.]

Concepts principaux associés à une application Express fournissant une API

- Configuration et démarrage d'une application
- Serveur dynamique
- Middlewares
- Routing

Configuration et démarrage d'une application

```
{
  "name": "more-than-hello-world-hbs",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "dev": "nodemon ./bin/www",
    "start": "node ./bin/www"
  },
  "dependencies": {
    "express": "~4.16.1",
    "hbs": "~4.0.4",
    …
  }
}
```

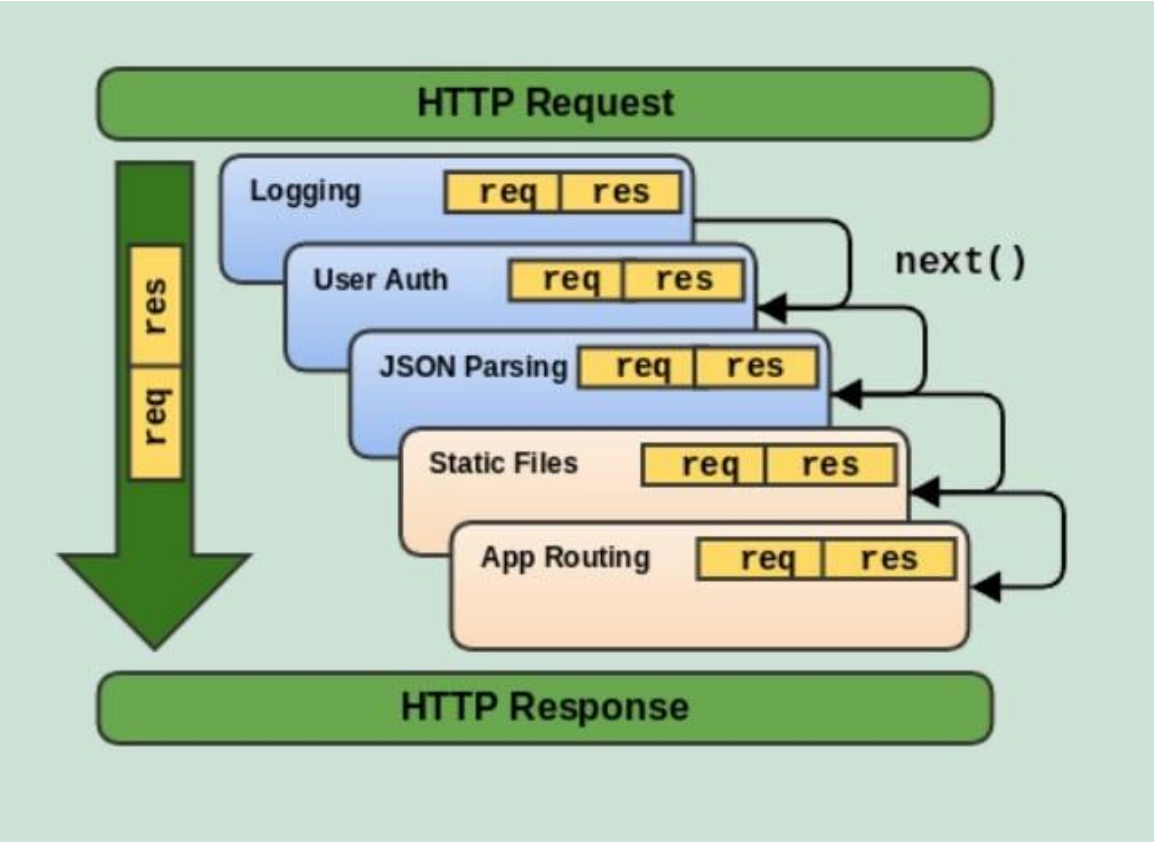
- Configuration
  - package.json
  - Détails [59.]
- Démarrage
  - npm start
  - npm run dev

Serveur dynamique

bin/www

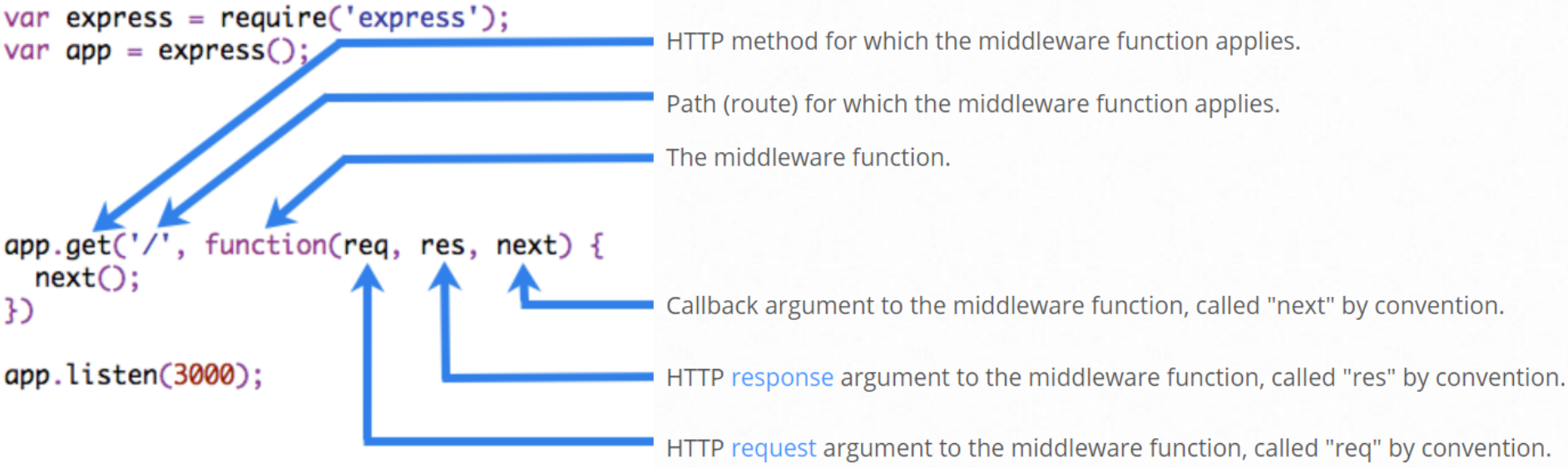
```
var app = require('../app');
var http = require('http');
var port = normalizePort(process.env.PORT || '80');
app.set('port', port);
var server = http.createServer(app);
server.listen(port);
```

Middlewares



Comprendre les Middlewares sous Express [60.]

Middlewares



Ecrire un middleware [61.]

Middlewares

- Application-level middleware
- Router-level middleware
- Error-handling middleware
- Built-in middleware
- Third-party middleware
- Details [62.]

Application-level middleware

- Utilisation de middleware [62.]

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

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

Router-level middleware

- Utilisation de middleware [62.]

```
var router = express.Router();
// a middleware function with no mount path. This code is executed for every r
equest to the router
router.use(function (req, res, next) {
  console.log("Time:", Date.now());
  next();
});
/* GET /pizzas : list all the pizzas from the MENU */
router.get("/", function (req, res, next) {
  return res.json(menu);
});
```

Error-handling middleware

- Utilisation de middleware [62.]

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

- 4 arguments au lieu de trois
- A définir après tous les middlewares pouvant générer une erreur via **next(err)**

Built-in middleware & third-party middleware

```
var createError = require("http-errors");
var express = require("express");
var path = require("path");
var cookieParser = require("cookie-parser");
var logger = require("morgan");
app.use(logger("dev")); // HTTP request logger
app.use(express.json()); // Parse requests with JSON payloads
app.use(express.urlencoded({ extended: false })); // Parse requests with URL-
                                                  encoded payload
app.use(cookieParser()); // Parse cookie header (req.cookies)
app.use(express.static(path.join(__dirname, "public"))); // Serve static assets
```

Routing

- Contrôle de la réponse à une requête client pour un **endpoint**/**URI**/**PATH** et une méthode HTTP
- Définition d'une route [63.] : **app.METHOD(PATH, HANDLER)**

```
/* GET user resource identified by its username */
router.get("/:username", function (req, res, next) {
  console.log("GET users/:username", req.params.username);
  const userFound = User.getUserFromList(req.params.username);
  if (!userFound) return res.status(404).send("ressource not found");
  return res.json(userFound);
});
```

Routing

- Définition d'un router (mini-app) :

/app.js

```
var usersRouter = require('./routes/users');
app.use('/users', usersRouter);
```

/routes/users.js

```
var express = require('express');
var router = express.Router();
var User = require("../models/User.js");
/* GET /users/  */
router.get('/', function(req, res, next) {
  return res.json(User.list);
});
module.exports = router;
```

Routing

- Chemin et paramètres d'une route [63.] :
  - Paramètres : **req.params**

```
/* GET user resource identified by its username */
router.get("/:username", function (req, res, next) {
  console.log("GET users/:username", req.params.username);
  const userFound = User.getUserFromList(req.params.username);
  if (!userFound) return res.status(404).send("ressource not found");
  return res.json(userFound);
});
```

Routing

- Chemin et paramètres de la requête (« query string ») [114.] :
  - Paramètres : **req.query**

```
/**
 * GET /pizzas : read all the pizzas from the menu
 * /pizzas?order=title : order by title : ascending
 * /pizzas?order=-title : order by title : descending
 */
router.get("/", function (req, res) {
  console.log("GET /pizzas");
  return res.json(pizzaModel.getAll({ order: req.query.order }));
});
```

Routing

- Parser le body d'une requête JSON :
  - Représentation par le client de la ressource à créer : format JSON (Media type)
  - => Content-Type header = application/json

```
POST http://localhost:3000/api/users/
Content-Type: application/json

{
    "email":"student@vinci.be",
    "password":"Student"
}
```

Routing

- Parser le body d'une requête JSON :
  - Paramètres : **req.body** grâce à **express.json()**

```
/* POST new user */
app.post("/users", function (req, res, next) {
  if (User.isUser(req.body.username)) return res.status(409).end();
  let newUser = new User(req.body.username, req.body.password);
  newUser.save();
  return res.json({ username: req.body.email });
});
```

Routing

- Méthodes associées aux réponses [63.] :
  - **res.json()** : renvoi d'une réponse au format JSON
  - **res.send()** : renvoi d'une réponse (types variés)
  - **res.end()** : fin du processus de réponse
  - **res.render()** : render d'un template de view
  - **res.redirect()** : redirection d'une requête
  - **...**

Gestion de la réponse

- Gestion des codes de status HTTP [76.]
  - Réponses informatives (100–199),
  - Réponses en cas de succès (200–299),
  - Redirections (300–399),
  - Erreurs du client (400–499),
  - Erreurs du serveur (500–599).

Gestion de la réponse

- Gestion des codes de status HTTP [76.]

```
router.post("/login", function (req, res, next) {
  let user = new User(req.body.username, req.body.password);
  if (!user.checkCredentials(req.body.username, req.body.password))
      return res.status(401).send("bad email/password");

  return res.json({ username: req.body.username });
});
```

En résumé : RESTful API

- Initialisation d'un boilerplate
- Gestion d'opérations sur des ressources

GET : lecture de ressource(s)

```
router.get("/", function (req, res) {
    return res.json(menu);
});

router.get("/:id", function (req, res) {
  const foundIndex = menu.findIndex((pizza) => pizza.id == req.params.id);
  // Send an error code '404 Not Found' if the pizza was not found
  if (foundIndex < 0) return res.status(404).end();
  return res.json(menu[foundIndex]);
});
```

- Mise en cache possible

POST : création d'une ressource

```
router.post("/", function (req, res) {
  if ( // invalid parameter : 'Bad Request' sent to the client
    return res.status(400).end();
  // add pizza to the menu with new id… : not all the code is given here!
  const newPizza = { id: nextId,
  title: req.body.title,
  content: req.body.content };
  // return the new pizza to the client (with generated id)
  return res.json(newPizza);  });
```

- Paramètres dans le body de la requête
- Généralement non mis en cache

PUT : modification de ressource(s)

```
router.put("/:id", function (req, res) {
  if ( // invalid parameter : 'Bad Request’ sent to the client
    return res.status(400).end();
  // check if pizza exists : 'Not Found' sent to the client if not found
  const foundIndex = menu.findIndex((pizza) => pizza.id == req.params.id);
    if (foundIndex < 0) return res.status(404).end();
  // update the pizza found in the menu and send it back to the client…
  const updatedPizza = { ...menu[foundIndex], ...req.body };
  menu[foundIndex] = updatedPizza;
  return res.json(updatedPizza); });
```

- Param. à mettre à jour = body de la requête
- ID de la ressource à mettre à jour dans l’URL

DELETE : suppression de ressource(s)

```
router.delete("/:id", function (req, res) {
  const foundIndex = menu.findIndex((pizza) => pizza.id == req.params.id);
  // Send an error code '404 Not Found' if the pizza was not found
  if (foundIndex < 0) return res.status(404).end();
  const itemRemoved = menu.splice(foundIndex, 1);

  return res.json(itemRemoved[0]);
});
```

- ID de la ressource à supprimer dans l'URL

Création d'une RESTful API sous Express

- DEMO : Création d'une RESTfull API pour une pizzeria : Part 1 – CRUD pizzas : gestion de données non persistantes, code peu structuré
  - Quelles opérations ?
  - Quelle(s) ressource(s) ?

Création d'une RESTful API sous Express

- DEMO : Création d'une RESTfull API pour une pizzeria : Part 1 – CRUD pizzas : gestion de données non persistantes, code peu structuré
  - GET /pizzas
  - GET /pizzas/{id}
  - POST /pizzas
  - PUT /pizza/{id}
  - DELETE /pizza/{id}