FORMATION

NODE JS

Allan BERNIER - 2024

Présentations

Allan BERNIER

1 an développeur full stack
Adesin (Laravel - VueJS)

3 ans Freelance

Formateur
Développeur full stack (Laravel - VueJS)

1 an développeur back
Banque populaire caisse d’épargne
(Spring - Angular)

Participants

- Avoir une bonne connaissance du protocole HTTP et de JavaScript.
- Quelles sont vos attentes par rapport à cette formation ?
- Avez-vous déjà eu une expérience professionnelle sur VueJS ?

Déroulement de la formation

Horaires, temps de pause

- Début 9h
- Pause 10h30 - 10h45
- Déjeuner 12h15 - 13h30
- Pause 15h15 - 15h30
- TP 15h30 - 17h30

Objectifs de formation

push par chapitre sur github :
www.github.com/allanBernier/

1. Anticiper et développer asynchrone dans un environnement multi-utilisateurs
2. Identifier les API fondamentales fournies par Node.js
3. Décrire NPM et la modularité, de façon approfondie
4. Accéder aux données depuis Node.js
5. Utiliser les modules Express
6. Réaliser des API REST et les tester
7. Identifier les dernières fonctionnalités de Node.js
8. Tester une application Node.js.

Programme

- Rappels JavaScript en ECMAScript 6
- Présentation rapide de NPM (Node Package Manager)
- Les modules dans Node.js
- Apports de N-API dans les dernières versions
- Présentation de Node.js
- Intérêts de JavaScript côté serveur
- Asynchronisme, programmation évènementielle
- Logique asynchrone
- Des callbacks partout
- Patterns asynchrones
- Event et EventEmitter
- Gestion des processus fils
- Timers
- Les Worker Threads
- Rappels de la programmation asynchrone
- Modules HTTP
- Module FS (File System)
- Ressources statiques
- Moteurs de templates
- Templates avec EJS
- Modes de connexion
- Accès MongoDB
- Accès MySQL
- ORM pour Node.js
- Commandes NPM
- Installation locale ou globale
- Configuration
- API Requêtes et Réponses
- API Routes
- Création de l'application cliente avec le framework Vue.js
- Test

Sommaire

1. **Rappels ES6**
2. **Express**
3. **CRUD SQL**
4. **CRUD NoSQL**
5. **Templates EJS**
6. **CRUD express/vueJS**
7. **BONUS : Event Emitter, Threads, ChildsProcess**

ES6

Express

Créer une application express

Create express app

```
const express = require('express')
const app = express()
const port = 3000

app.use(express.json())
app.listen(port, () => {
  console.log(`Serveur Express.js écoutant sur le port ${port}`)
})
```

Créer une route via express

Hello world

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

Récupérer les Url params

Url Params

```
app.get('/url-params/:name', (req, res) => {
  const name = req.params.name
  res.send(`Hello ${name}`)
})
```

Récupérer les Url params

Url Params

```
app.get('/path', function(req, res) {  // url: /path?name='test'
  const name = req.query['name'];
  res.send(`Hello ${name}`);
});
```

Récupérer les Url params

Url Params

```
app.get('/path', function(req, res) {  // url: /path?name='test'
  const name = req.query['name'];
  res.send(`Hello ${name}`);
});
```

Code retour

status code

```
app.get('/error', function(req, res) {
  res.status(400);
  res.send('None shall pass');

  // res.status(400).send('None shall pass');
});
```

Les types de réponses

response

```
// send html
app.get('/html', function(req, res) {
  res.send('<h1>Hello World!</h1>');
});

// send json
app.get('/json', function(req, res) {
  res.json({ message: 'Hello World!' });
});

// send file
app.get('/file', function(req, res) {
  res.sendFile('path/to/file');
});
```

Séparer en package

app.js

```
const express = require("express")
const app = express()
const productRouter = require("./router/productRouter")
app.use('/product', productRouter)
app.listen(process.env.SERVER_PORT, () => {
    console.log("Serveur démarré !")
})
```

productController

```
let methods = {}

methods.getAll = (req, res) => {};
methods.findById = (req, res) => {};
methods.store = (req, res) => {};
methods.update = (req, res) => {};
methods.destroy = (req, res) => {};

module.exports = methods
```

productRouter.js

```
const express = require("express")
const router = express.Router();
const controller = require("../controller/productController")

router.get("", controller.getAll)
router.get("/:id", controller.findById)
router.post("", productRequest, controller.store)
router.put("/:id", productRequest, controller.update)
router.delete("/:id", controller.destroy)

module.exports = router
```

Separing files

```
// app.js
const app = express()
app.listen(3000, () => {
  console.log('Server running on port 3000')
})
// router.js
const express = require("express")
const router = express.Router();

router.get("", controller.getAll)
router.get("/:id", controller.findById)
router.post("", controller.store)
router.put("/:id",  controller.update)
router.delete("/:id", controller.destroy)


// controller.js
const controller = {
  getAll: (req, res) => {
    res.send("Get all resources")
  },
  findById: (req, res) => {
    res.send("Get resource by id")
  },
  store: (req, res) => {
    res.send("Create new resource")
  },
  update: (req, res) => {
    res.send("Update resource")
  },
  destroy: (req, res) => {
    res.send("Delete resource")
  }
}
```

Middleware

middleware

```
const productRequest = (req, res, next) => {
    if ( req.body.name != undefined && req.body.price != undefined){
        return next();
    }
    res.status(403).send("Price or Name not fill")
}

router.post("/store", productRequest, controller.store)
```

CRUD SQL

Sequelize configuration

DB config

```
const sequelize = require("sequelize")

const db = new sequelize(
    "database_name",
    "database_user",
    "database_secret_password",
    {
        dialect : "mysql",
        host: "localhost" // "database_url"
    }
)

module.exports = db;
```

Definir un modèle

Uploaded using RayThis Extension

```
const sequelize = require("sequelize")
const db = require("../config/database")


const productSchema = db.define(
    "product",
    {
        id : {type : sequelize.INTEGER, primaryKey: true, autoIncrement: true},
        name : {type : sequelize.STRING},
        price : {type : sequelize.FLOAT}
    }
)

module.exports = productSchema
```

Faire des requêtes

get de model

```
let Product = require("../model/product")
```

Uploaded using RayThis Extension

```
findById = async (req, res) => {
    let product = await Product.findByPk(req.params.id)

    if (product != null){
        return res.json(product)
    }
    return res.status(404).send("Product not found")
}
```

Uploaded using RayThis Extension

```
getAll = (req, res) => {
    Product.findAll()
    .then((queryResult) => { res.json(queryResult) } )
}
```

Uploaded using RayThis Extension

```
store = (req, res) => {
    Product.create(req.body)
    .then( (queryResult) => { res.status(201).json(queryResult)})
}
```

Uploaded using RayThis Extension

```
update = (req, res) => {
    Product.update(req.body, { where : {id : req.params.id }})
    .then( (queryResult) => {
        if (queryResult[0] == 1){
            return res.send("Product updated")
        } else {
            return res.status(403).send("Unknown id")
        }
    })
    .catch( (err) => {
        return res.status(403).send("Unknown id")
    })
}
```

Uploaded using RayThis Extension

```
destroy = async (req,res) => {
    await Product.destroy({ where : {id : req.params.id}})
    res.send("Product deleted")
}
```

CRUD NOSQL

mongoose configuration

app.js

```
const mongoose = require("mongoose")
mongoose.connect("mongodb+srv://user:password@cluster0.w7evgzl.mongodb.net/?
retryWrites=true&w=majority")
.then( (res) => {
    console.log("Connected to db")
})
```

Definir un schéma

product.js

```
const mongoose = require("mongoose")
const productSchema = mongoose.Schema({
    name : String,
    price : Number
})
module.exports = mongoose.model("product", productSchema);
```

Faire des requêtes

get de model

```
let Product = require("../model/product")
```

Uploaded using RayThis Extension

```
const getAll = (req, res) => {
    Product.find()
    .then((queryResult) => { res.json(queryResult) } )
}
```

Uploaded using RayThis Extension

```
const findById = (req, res) => {
    Product.findById(req.params.id)
    .then( (queryResult) => { res.json(queryResult) })
    .catch( (err) => { res.status(404).send("Product not found")})
}
```

Uploaded using RayThis Extension

```
const store = (req, res) => {
    Product.create(req.body)
    .then( (queryResult) => { res.status(201).json(queryResult)})
}
```

Faire des requêtes

get de model

```
let Product = require("../model/product")
```

Uploaded using RayThis Extension

```
const update = (req, res) => {
    Product.updateOne({_id : req.params.id  }, req.body)
    .then( (queryResult) => {
        if (queryResult.modifiedCount == 1){
            return res.send("Product updated")
        } else if (queryResult.matchedCount == 0) {
            return res.status(404).send("Product not found")
        } else {
            return res.status(403).send("Product not updated")
        }
    })
    .catch( (err) => {
        return res.status(403).send("Unknown id")
    })
```

Uploaded using RayThis Extension

```
const destroy = (req,res) => {
    let id = req.params.id
    Product.deleteOne({_id : id  })
    .then( (queryResult) => {
        console.log(queryResult)
        res.send("Product deleted")
    })
}
```

Template EJS

Configuration

app.js

```
app.set("view engine", "ejs")
app.set("views", path.join(__dirname, "views")) // "./views"
app.get("/", (req, res) => {
    res.render('helloworld', {name : "Toto"})
})

app.use(express.static(path.join(__dirname, "public")))
```

Page EJS

helloworld.ejs

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>Je suis une page html</h1>

    <h2> <%= name %> </h2>

</body>
</html>
```

Layout EJS

Uploaded using RayThis Extension

```
<%- include("_header") %>
<body>
    <%- include("_product_form") %>
</body>
<%- include("_footer") %>
```

EJS Tags

Uploaded using RayThis Extension

```
// <% 'Scriptlet' tag, for control-flow, no output
// <%_ ‘Whitespace Slurping’ Scriptlet tag, strips all whitespace before it
// <%= Outputs the value into the template (HTML escaped)
// <%- Outputs the unescaped value into the template
// <%# Comment tag, no execution, no output
// <%% Outputs a literal '<%'
// %> Plain ending tag
// -%> Trim-mode ('newline slurp') tag, trims following newline
// _%> ‘Whitespace Slurping’ ending tag, removes all whitespace after it
```

VueJS

Event Emitter, Threads, ChildsProcess

Exercice : CRUD Statique

Créer une API RESTful qui permet de gérer une liste de tâches. Les utilisateurs peuvent ajouter, mettre à jour, récupérer et supprimer des tâches à l'aide des méthodes HTTP GET, POST, PUT et DELETE.

Une tâche est composé de :
titre string required,
description string,
date_debut : date,
date_fin : date,
done : boolean

Exercice : CRUD SQL

Créer une API RESTful qui permet de gérer une liste de tâches. Les utilisateurs peuvent ajouter, mettre à jour, récupérer et supprimer des tâches à l'aide des méthodes HTTP GET, POST, PUT et DELETE.

Une tâche est composé de :
titre string required,
description string,
date_debut : date,
date_fin : date,
done : boolean

Exercice : CRUD NOSQL

Créer une API RESTful qui permet de gérer une liste de tâches. Les utilisateurs peuvent ajouter, mettre à jour, récupérer et supprimer des tâches à l'aide des méthodes HTTP GET, POST, PUT et DELETE.

Une tâche est composé de :
titre string required,
description string,
date_debut : date,
date_fin : date,
done : boolean

Exercice : EJB Application tâches

Page Index : Cette page affiche un tableau listant toutes les tâches. Pour chaque tâche, les actions suivantes sont possibles :

Modifier : Accéder à la page de modification de la tâche sélectionnée.
Supprimer : Effacer la tâche de la liste.
Marquer comme accomplie (Done) : Indiquer que la tâche est terminée.
Marquer comme non accomplie (Undone) : Revenir sur le statut accompli d'une tâche pour la marquer comme non terminée.

Page Update (Mise à jour) : Permet de modifier une tâche existante. Tous les champs de la tâche à modifier sont disponibles pour être édités. Cette page utilise le composant _taskForm pour la saisie des informations.

Page Create (Création) : Permet de créer une nouvelle tâche. Cette fonctionnalité utilise également le composant _taskForm, offrant un formulaire pour saisir les détails de la nouvelle tâche.

Composant _TaskForm : C'est le formulaire utilisé à la fois pour créer et modifier les tâches. Il contient tous les champs nécessaires à la saisie des informations relatives à une tâche.

Composant _Header : C'est le layout qui est affiché sur toutes les pages relatives aux tâches. Il inclut un bouton permettant de naviguer vers la page Index et un autre vers la page Create, facilitant l'accès rapide aux fonctionnalités principales de l'application.

Exercice : Vue JS

Reprendre toutes les fonctionnalités de l’application tâches mais cette fois ci avec une application vuejs qui va communiquer avec une application express pour l’accès à la base de donnée

Exercice : Event Emitter

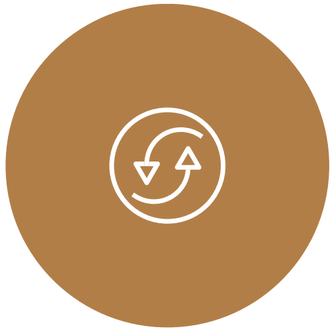
Créer un event get-random

génère un nombre aléatoire puis envoie un emit sur random-result

Créer un event random-result

Récupère un nombre et l'affiche dans la console

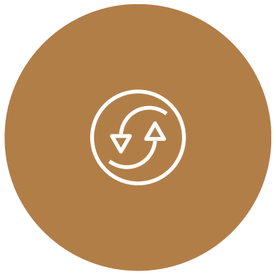
Exercice : WorkerThread



Créer un worker qui affiche la liste des nombres premier à partir d'un million

Le thread principal doit afficher une log toute les 2 seconde pendant que le prime worker tourne

Exercice : Child Process



Créer un child process qui affiche la version envoyé en paramètre au lancement du ficher node js

```
node .\version.js node
v16.13.0
```

```
node .\version.js php
PHP 7.4.1 (cli) (built: Dec 17 2019 19:23:59) ( NTS Visual C++ 2017 x64 )
Copyright (c) The PHP Group
Zend Engine v3.4.0, Copyright (c) Zend Technologies
```