Documenation techniques API

Identification du document

Nature du document	Nom du fichier	Format
Documentation technique	MegaCastingAPI	Pdf et md

Versionnage

Numéro	Date	responsable	descriptif
1.0.0	08/01/19	S. Le Cann	Création du document

Sommaires

Technologies

Nodejs

Pour la réalisation des API notre choix s'est porté sur la plateforme logiciel libre et événementielle Nodejs écrite en Javascript.

Nodejs intègre un serveur HTTP. Son fonctionnement est basé sur une boucle événementielle lui permettant de supporter de fortes montées en charge.

L’utilisation de Node.js en tant que serveur web permet de traiter un gros volume de requêtes simultanément de manière efficace. Cette performance élevée s’explique par une conception asynchrone (modèle non bloquant) permettant d’éviter les attentes. Ainsi, plusieurs requêtes peuvent être lancées en parallèle. Les résultats sont traités au fil de l'eau.

Node utilise le compilateur JavaScript V8 de Google focalisé sur les performances et la sécurité. A l'origine, la machine virtuelle V8 a été créée pour interpréter le JavaScript dans le navigateur Chrome. Merci aux développeurs de Google, le moteur V8 est et restera à la pointe de la technologie. Par ricochet, les progrès de V8 impactent directement Node.js.

Le gestionnaire de packages npm Initialement gestionnaire de packages de Node.js, npm est aujourd'hui le gestionnaire de packages du monde JavaScript. On y trouve aussi bien des modules pour le backend que pour le frontend.

Les développeurs sont très actifs. Ils contribuent constament à l'amélioration des librairies Node.js au point d'en faire un problème à part entière : la JavaScript fatigue.

Npm compte aujourd'hui plus de 500 000 packages et le nombre ne cesse d'augmenter, bien plus rapidement que pour les autres langages.

Une technologie stable et éprouvée Node.js n’est plus la petite dernière des technologies. C'est une techologie utilisée et éprouvée par les géants du web : Netflix, Trello, PayPal, LinkedIn, Walmart, Uber, Medium, Groupon, Ebay ou la NASA.

Ce support par les gros acteurs et cette politique de Long Term Support (LTS) donne à Node.js l'avantage d'être une technologie sûre et un bon choix sur l'avenir.

- Nodejs
    - packages
        - algoliasearch: 3.32.0 Implémentation d'algolia
        - bcryptjs: 2.4.3 Chiffrement
        - body-parser: 1.18.3 Récupération du body des requètes http
        - express: 4.16.4 Framework web
        - express-validator: 5.3.1 validation des données
        - jsonwebtoken: 8.4.0 Token d'authentification
        - sequelize: 5.0.0-beta.15 Object relation mapping
        - sequelize-auto: 0.4.29 Création des modèles à partir de la bd
        - tedious": "^3.0.1 Pilote SQL Server
    - Syntaxe
        - Ecmascript 6
        - Javascript
- Azure (Hébergement et build)

Cycle de vie

1. Chargement de l'API 1. Chargement des dépendances 2. Chargement des middlewares 3. Chargement des routes

2. Synchronisation à la base de données 1. Lancement de sequelize 2. Tentative de contact à la base de données 3. Chargement des modèles

3. Lancement du serveur web 1. Demarrage app.listen()

```
 Server runnning on Port:- 3000Started at :- Thu Jan 10 2019 02:33:31
GMT+0100 (GMT+01:00)
```

Reception d'une requète http

- Analyse de la requète

```
app.use(bodyParser.json());
```

- Paramétrage du Cross-origin resource sharing (CORS)

```
app.use((req, res, next) => {
res.setHeader('Acces-Control-Allow-Origin', '*');
res.setHeader('Acces-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE');
res.setHeader('Acces-Control-Allow-Headers', 'Content-Type, Authorization');
next();
});
```

- Interception de la requète par un router

```
app.use('/', defaultRoutes);
app.use('/auth', authRoutes);
app.use('/adresses', isAuth, adresseRoutes);
app.use('/contacts', isAuth, contactRoutes);
app.use('/contenus', isAuth, contenuRoutes);
app.use('/prospects', isAuth, prospectRoutes);
app.use('/clients', isAuth, clientRoutes);
app.use('/codePostaux', isAuth, codePostalRoutes);
app.use('/partenaires', isAuth, partenaireRoutes);
app.use('/typeContenus', isAuth, typeContenuRoutes);
app.use('/domaineMetiers', isAuth, domaineMetierRoutes);
app.use('/localisations' ,isAuth, localisationRoutes)
app.use('/metiers' ,isAuth, metierRoutes);
app.use('/offreCastings', isAuth, offreRoutes);
app.use('/statutJuridiques' ,isAuth, statutJuridiqueRoutes);
app.use('/utilisateurs' ,isAuth, utilisateurRoutes);
app.use('/contrats' ,isAuth, contratRoutes);
```

- Envoi vers une route

```
router.get('/', prospectController.getProspects);
router.get('/formated', prospectController.getProspectFormated);
```

```
router.get('/:prospectId', prospectController.getProspect);
router.post('/', prospectController.createProspect);
router.delete('/:prospectId', prospectController.deleteProspect);
router.put('/:prospectId', prospectController.updateProspect);
```

- Envoi vers la fonction correspondante dans le contrôleur

```
exports.getProspects = (req, res, next) => {
        Prospect.findAll()
        .then(prospects => {
                res.status(200).json({prospects: prospects});
        })
        .catch(err => {
                if (!err.statusCode) {
                err.statusCode = 500;
        }
        next(err);
        });
}
```

Décomposition d'une fonction

Déclaration de la fonction

```
getProspects = (req, res, next) => {}
```

Utilisation de l'API Sequelize, retourne une promesse

```
Prospect.findAll()
```

Appel de fonction asynchrone, imbrication de callback

```
.then(prospects => {
})
```

Envoi du callback dans une réponse http avec code de status 200(succès), formatage en JSON du callback

```
res.status(200).json({prospects: prospects});
```

On capture les erreurs

```
.catch(err => {
}
```

On test si l'erreur a un code de status. Si non on le passe en 500

```
if (!err.statusCode) {
        err.statusCode = 500;
}
```

On renvoi ensuite l'erreur au générateur

```
next(err);
```

Le générateur formate l'erreur et la renvoi en réponse http avec un message en json ainsi que le code erreur

```
app.use((error, req, res, next) => {
    console.log(error);
    const status = error.statusCode || 500;
    const message = error.message;
    const data = error.data;
    res.status(status).json({ message: message, data: data });
});
```

Cas d'utilisation

- Requète /prospects
    - GET Visualiser les prospects
    - GET /:id Recherche un prospect
        - Include: Saisie d'un id
    - POST Créer un prospect
        - Include: Saisie PRO_NAME
    - DEL supprimer un prospect
        - Include: Saisie d'un id

Configuration

Sequelize auto

Commande de création des modèles à partir de la bdd

```
sequelize-auto -o "./models" -d MEGACASTING -h megacasting.database.windows.net -u
adminmegacasting -p 1433 -x t4tX38CwrHQJbDWkl2qr -e mssql -c config.json
```

Fichier config.json. Importance du dialecOptions dans le cas ou on contacte une DB mssql sur Azure. La base est chiffrée

```
{
  "host": "megacasting.database.windows.net",
  "dialect": "mssql",
  "dialectOptions": {
    "encrypt": true
  }
}
```

Connection à la base de données

Création de l'objet sequelize pour la connection à la base de données

```
const sequelize = new Sequelize(dataBase, login, mdp, {
    host: 'megacasting.database.windows.net',
    dialect:'mssql',
    dialectOptions: {
      encrypt: true,
      },
    operatorsAliases: false,
    pool: {
      max: 5,
      min: 0,
      acquire: 30000,
      idle: 10000
    },
  });
```

CORS

Permet d'autoriser des requètes d'une origine différentes. Paramétrage des méthodes autorisées, et des paramètres du header

```
app.use((req, res, next) => {
    res.setHeader('Acces-Control-Allow-Origin', '*');
    res.setHeader('Acces-Control-Allow-Methods', 'GET, POST, PUT, PATCH, DELETE');
    res.setHeader('Acces-Control-Allow-Headers', 'Content-Type, Authorization');
    next();
});
```

Logique de contrôle de l'authentification

```
module.exports = (req, res, next) => {
    const autHeader = req.get('Authorization');
    const token = 'Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dpbiI6IkpHcmF5IiwiaWQiOiI1MiIsImlhdCI6M
TU0NzAzMjY5MiwiZXhwIjoxNTQ3MDc1ODkyfQ._6CsUUIznN5BflzT7nrVvye34FMzNvpNgLd10Aw0iUE'
    const secret =
'85B8EF807F51D09FC54288EE2539B81D1CCA149A1BE5C081C0937DCCB4D91DAC';

    console.log(autHeader);
    console.log(token);
    if (!autHeader) {
        const error = new Error('Non authentifié');
        error.statusCode = 401;
        throw error;
    }

    if (autHeader == token) {
        next();

    } else {
        const token = autHeader.split(' ')[1];
        let decodedToken;
        try {
            decodedToken = jwt.verify(token, secret);
        } catch (error) {
            error.statusCode = 500;
            throw error;
        }
        if (!decodedToken) {
            const error = new Error('Non authentifié');
            error.statusCode = 401;
            throw error;
        }
        req.userId = decodedToken.userId;
        next();
    }
};
```

Implémentation algolia

Intégration de la méthode addObjects lors de la création d'une offre pour l'envoyer vers le service de référencement AlgoliaSearch

```
.then(async (offre) => {
            const object = await getFormatedOffresByIdSvc(offre.CAST_ID);
            index.addObjects(object, function(err, content) {
              });
            res.status(201).json({offre: offre});
        })
```

Intégration continue

Utilisation du service Serveur de builds Kudu App Service pour le déploiement automatique des applications nodejs dans azure directement à partir de repository github

Librairies node

Sequelize-auto

Permet de créer automatiquement les models de données sequelize dans nodejs à partir d'une base de données

Tedious

Tedious est l'implémentation en javascript du Tabular Data Stream Protocol (TDS) de microsoft. C'est le protocole de comunication entre un client et une base de données relationel SQL Server

bcrypt

Pour l'utilisation de bcrypt. Installer en administrateur les outils de développement de microsoft, Visual Studio Build Tools avec la commande

```
npm install --global --production windows-build-tools

node-gyp --python C:/Python27/
npm config set python C:/Python27/python.exe
```

Ensuite installer l'addon Node.js node-gyp

```
npm install -g node-gyp
```

switch vers bcryptjs

```
https://www.npmjs.com/package/bcryptjs
```

Fonctionne en local Fonctionne sur Azure

Bcrypt a trop de dépendance système tel que python 2.7 windows build tools, qui ne sont pas intégré de base sur azure. Installation non réussi. Avec Bcryptjs, il n'y a pas de dépendance. Fonctionne parfaitement.

jsonwebtoken

JSON Web Token (JWT) est un standard ouvert défini dans la RFC 75191. Il permet l'échange sécurisé de jetons (tokens) entre plusieurs parties. Cette sécurité de l'échange se traduit par la vérification de l'intégrité des données à l'aide d'une signature numérique. Elle s'effectue par l'algorithme HMAC ou RSA.

```
npm install --save jsonwebtoken
```

Memory-cache

Cache mémoire pour Nodejs

```
npm install memory-cache --save
```