

TD Back-End

Introduction

Maintenant que vous êtes capable de développer un front comme il se doit, nous passons donc à la deuxième partie qui compose une application web : le back-end 🎉

Le starter à cloner se trouve ici:

<https://github.com/NablaT/backend-ps6-starter-quiz.git>

Pour faire des requêtes à votre serveur, utilisez Postman:

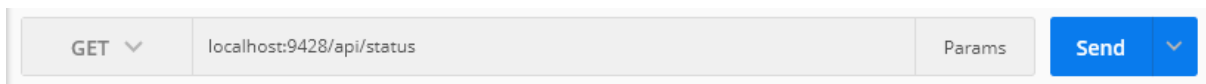
<https://www.postman.com/downloads/>

Note: Vous utilisez Webstorm et vous voyez pleins d'erreurs dans le starter ? Il faut alors que vous changiez la version de JavaScript utilisé par Webstorm. Pour cela, allez dans **File > Settings > Languages & Frameworks > JavaScript** et choisissez **ECMAScript 6** comme language version.

Exercice 0 : Prenez vos marques !

Petit exercice d'introduction des familles pour vous permettre de faire copain-copain avec le code existant. Dans la suite, vous allez avoir besoin de [Postman](#), téléchargez le en cliquant sur le lien et vous pourrez ensuite commencer le TD. Postman va vous permettre d'interagir avec votre serveur, comme le ferait n'importe quel site web.

- 1) Clonez le projet si ce n'est pas déjà fait, installez les dépendances (*npm install*), et lancer la commande : *npm run dev*
- 2) **Vérifiez le statut du serveur.** Pour cela, rien de plus simple, ouvrez un nouveau Tab dans Postman, sélectionnez la méthode HTTP GET et ajoutez le lien vers votre serveur (normalement : `http://localhost:9428/api/status`) et appuyer sur **Send**.



Vous devriez avoir reçu un status code 200 avec la réponse "ok". Si c'est pas le cas, vérifiez que vous avez bien lancé votre serveur 😊. D'ailleurs vous devriez voir votre requête dans les **logs** de votre serveur.

```
[2019-02-20T16:57:45.590Z] GET /api/status 200 0.381 ms - 4
```

- 3) **Récupérez la liste des quizzes.** Là encore rien de compliqué, il suffit de changer **status** par **quizzes** dans l'url précédent et de **Send**. Vous devriez avoir récupéré un beau tableau vide `[]` ! Nous le remplirons dans une question ultérieure.

- 4) **Faites n'importe quoi.** Certains d'entre vous ont sûrement déjà testé ce scénario au cours des précédentes étapes 😊 (peut-être même sans faire exprès).
Nous venons de faire deux appels avec succès, voyons ce qu'il se passe lorsqu'on met n'importe quoi à la fin de notre url.

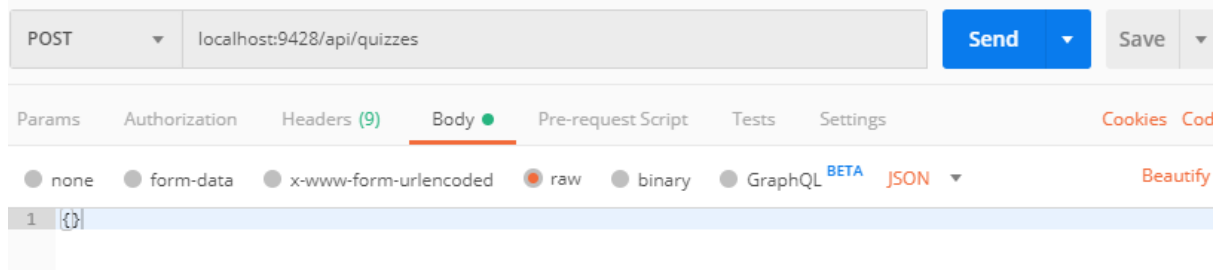
GET ▾	localhost:9428/api/cetdesttropchanme	Params	Send ▾
-------	--------------------------------------	--------	--------

Et vous allez voir dans les logs de votre serveur:

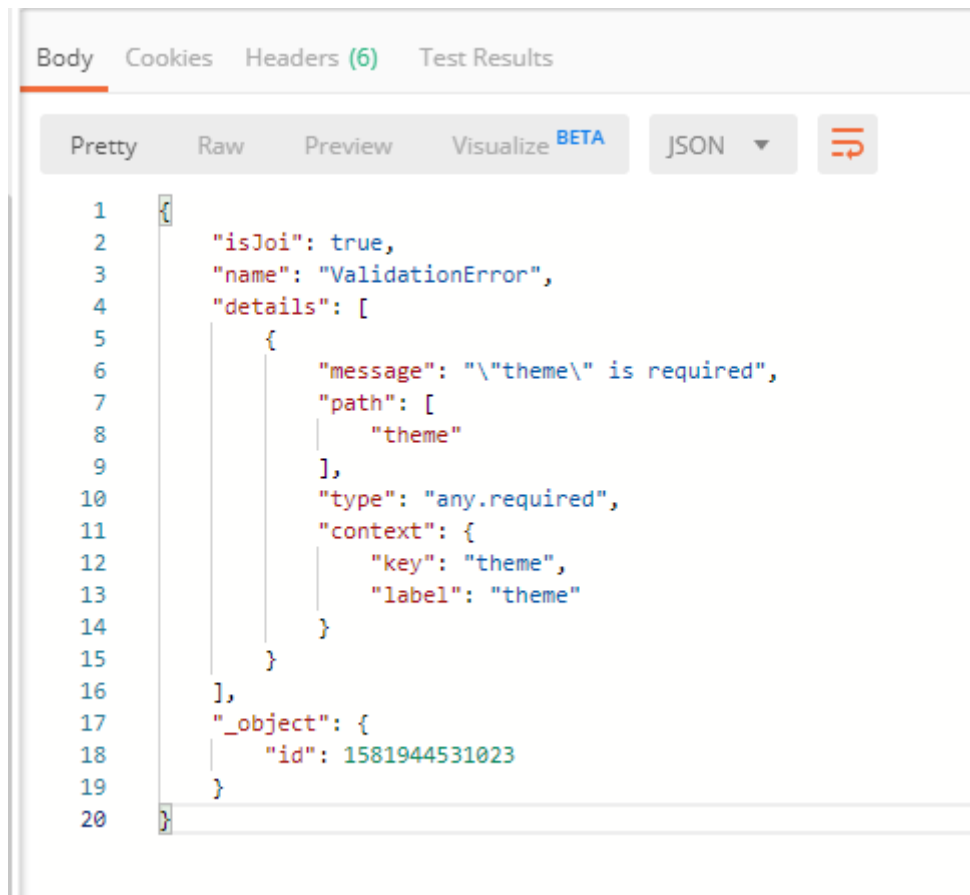
```
[2019-02-20T17:18:10.122Z] GET /api/cetdesttropchanme 404 0.109 ms - -
```

Le code renvoyé a changé ! Ce n'est plus **200** mais **404** 🚀. Votre serveur vous répond mais au lieu de vous renvoyer **200 -> succès de la requête**, il vous renvoie **404 -> not found**. Cela nous permet de comprendre que notre route n'existe pas. [Plus d'info ici !](#)


- 5) Voyons un peu comment marche ce serveur en explorant les fichiers présents dans le repository :
- allons voir le point d'entrée du serveur **app/index.js** : on y trouve un fichier simple qui appelle une fonction **buildServer**
 - dans **app/build-server.js** : on peut constater la construction du serveur avec diverses options et notamment la configuration de notre **api** à la ligne 13, allons donc voir cette api de plus près
 - dans **app/api/index.js** : on y voit la route GET /status ainsi que la définition des sous-routes **/quizzes**
 - quand on se rend dans **app/api/quizzes/index.js** on y voit 2 routes :
 - un GET pour récupérer la liste des quizzes (que vous avez normalement testé à la question 3) grâce à **Quiz.get()**
 - un POST pour créer un quiz grâce à **Quiz.create()**
 - la fonction **Quiz.get()** nous amène dans le fichier **app/models/index.js** puis **app/models/quiz.model.js** : on y voit la définition d'un quiz
- 6) **Ajoutez un quiz** : vous venez donc de constater qu'il y a une route POST permettant de créer un quiz. Utilisons là avec Postman !
- sélectionnez la méthode http POST
 - allez dans l'onglet Body
 - Sélectionnez le format "raw" puis à droite dans la select box, sélectionnez "JSON". Mettez-y un objet vide {}.
- Vous devriez avoir quelque chose comme ça :

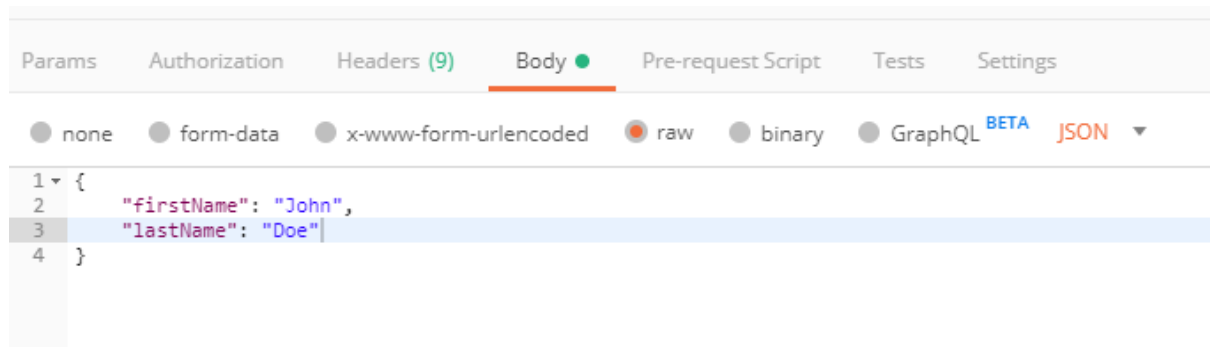


- Vous recevez un status code **400 Bad Request** et un message d'erreur vous expliquant que le champ theme est required:



- Mettez à jour l'objet envoyé (votre objet vide {}) afin de spécifier le theme). Vous pouvez aller voir le fichier **app/models/quiz.model.js** quels sont les champs required pour un Quiz. Continuez à mettre à jour votre objet envoyé en fonction des erreurs que vous recevez jusqu'à recevoir une réponse **201 Created** avec votre magnifique quiz retourné.

 **Tip:** Le format à envoyer est JSON, il vous faut rajouter des guillemets autour de vos clés pour que le format soit correct. Vous trouverez un exemple ci-dessous:



- Constatez maintenant que votre quiz a bien été sauvegardé dans **database/quiz.data.json**

Exercice 1 : Récupération d'un quiz par ID

Vous allez créer votre toute première route 😊 Ça va ? Pas trop la pression ? Allez c'est parti !

Étapes :

- dans **app/api/quizzes/index.js** : ajouter une route GET `/:quizId` (en vous inspirant du GET déjà présent). Il faut vraiment marquer `/:quizId`. Vous pourrez ensuite récupérer ce paramètre dans **req.params.quizId**. N'hésitez pas à mettre un `console.log` pour voir votre objet `req.params`.
- servez-vous de ce **req.params.quizId** pour trouver le bon quiz (il paraîtrait que le model Quiz aurait une méthode `getById`, je dis ça, je dis rien 😊)

Comme toujours, cela doit devenir une habitude, testez avec Postman que vous puissiez bien obtenir un quiz par ID en faisant un GET sur `/quizzes/IDduQuiz`

Exercice 2 : Supprimer un quiz

Pas très différent de la question précédente, vous allez devoir créer une route pour supprimer un quiz !

Étapes :

- dans **app/api/quizzes/index.js** : ajouter une route DELETE `/:quizId` (en vous inspirant du GET précédent).
- servez-vous de ce **req.params.quizId** pour supprimer le bon quiz (il paraîtrait que le model Quiz aurait une méthode `delete`, je dis ça, je dis rien 😊)

Exercice 3 : Mettre à jour un quiz

Bon, je pense que vous avez compris la musique donc on va la faire courte en donnant des indices pour voir si vous avez les bases :

- `PUT /:quizId`
- `req.body`

- Quiz.update()

Exercice 4 : On prend les mêmes et on recommence 😊 - gestion des Questions !

Maintenant que vous savez créer/updater/supprimer/récupérer des quizz, il serait judicieux de pouvoir faire la même chose avec les **questions**.

Nous souhaitons accéder aux questions d'un quizz spécifique. La route que nous souhaitons utiliser a la structure suivante:

/api/quizzes/:quizId/questions

Étapes :

- Création d'un nouveau model **question.model.js** dans le dossier **app/models**
 - Une question est composée des attributs suivant:
 - label: string, required
 - answers: array
 - quizId: number

👤 **Tip:** Le **quizId** n'est pas donné dans le body de la requête car il se trouve déjà dans l'url.

- Importer/exporter le nouveau model dans le fichier **app/models/index.js**
- Création d'un dossier **app/api/quizzes/questions** dans lequel vous devez créer un nouveau router.

Attention: Lorsque vous créez votre nouveau router dans votre fichier index.js dans le dossier questions, vous devez rajouter {mergeParams: true} en parametre de votre router:

const router = new Router({mergeParams: true})

Nous avons besoin de récupérer le quizId qui se trouve dans l'url et par défaut, les parametres de la route parent ne sont pas visibles dans la route enfant (ici la route parent est quizzes et la route enfant est questions)

- Utilisez ce nouveau Router dans **app/api/quizzes/index.js** grâce à la fonction use() du router.

👤 **Tip:** Le développeur malade du TD précédent a trouvé un bout de code qui montre l'utilisation d'un sous router de books pour gérer des chapitres:

```

JS index.js  X
app > api > books > JS index.js > ...
1  const { Router } = require('express')
2
3  const { Book } = require('../models')
4  const ChapterRouter = require('./chapters');
5
6  const router = new Router()
7
8  router.get('/', (req, res) => {
9    try {
10      res.status(200).json(Book.get())
11    } catch (err) {
12      res.status(500).json(err)
13    }
14  })
15
16  router.use('/:bookId/chapter', ChapterRouter)
17

```

- Ajoutez les mêmes routes que pour les quizz (get, get by id, post, put, delete) à l'intérieur de votre fichier.

Informations supplémentaires :

- GET: On ne veut récupérer que les questions associées à ce quizz.
- UPDATE & DELETE: On ne peut que modifier ou supprimer une question appartenant au quizz mentionné
- POST: Dans cette première version, voici à quoi devrait ressembler une question après création:

```

{
  "label": "Quelle est la capitale de la Finlande ?",
  "quizId": 1581945685209,
  "answers": [],
  "id": 1581958523422
}

```

 **Tip:** La fonction [parseInt](#) en JavaScript permet de transformer une string en un number.

- TESTEZ AVEC POSTMAN 

Exercice 5 : On prend les mêmes et on recommence encore 😏 - gestion des users !

Maintenant on aimerait pouvoir gérer des **users** comme les quizz. On souhaite créer/updater/supprimer/récupérer des **users**. Les utilisateurs sont indépendants des quizz pour le moment.

Pour cette question, vous êtes libres de définir le modèle de user que vous souhaitez.

Exercice 6 : Association des Questions et des Quizz

Notre API est pleine de fonctionnalités mais certaines nécessitent une petite optimisation. En effet, lorsque l'on récupère un quizz, l'objet ne contient pas les questions. On est obligé de faire manuellement un GET sur les questions de ce quizz pour les récupérer.

Le but est donc de mettre à jour notre get et get by id afin de récupérer un objet quizz qui contient l'ensemble de ses questions.