Node.js

API REST avec Express

API REST avec Express
Partie 1

RÈGLE D'OR EN DÉVELOPPEMENT :

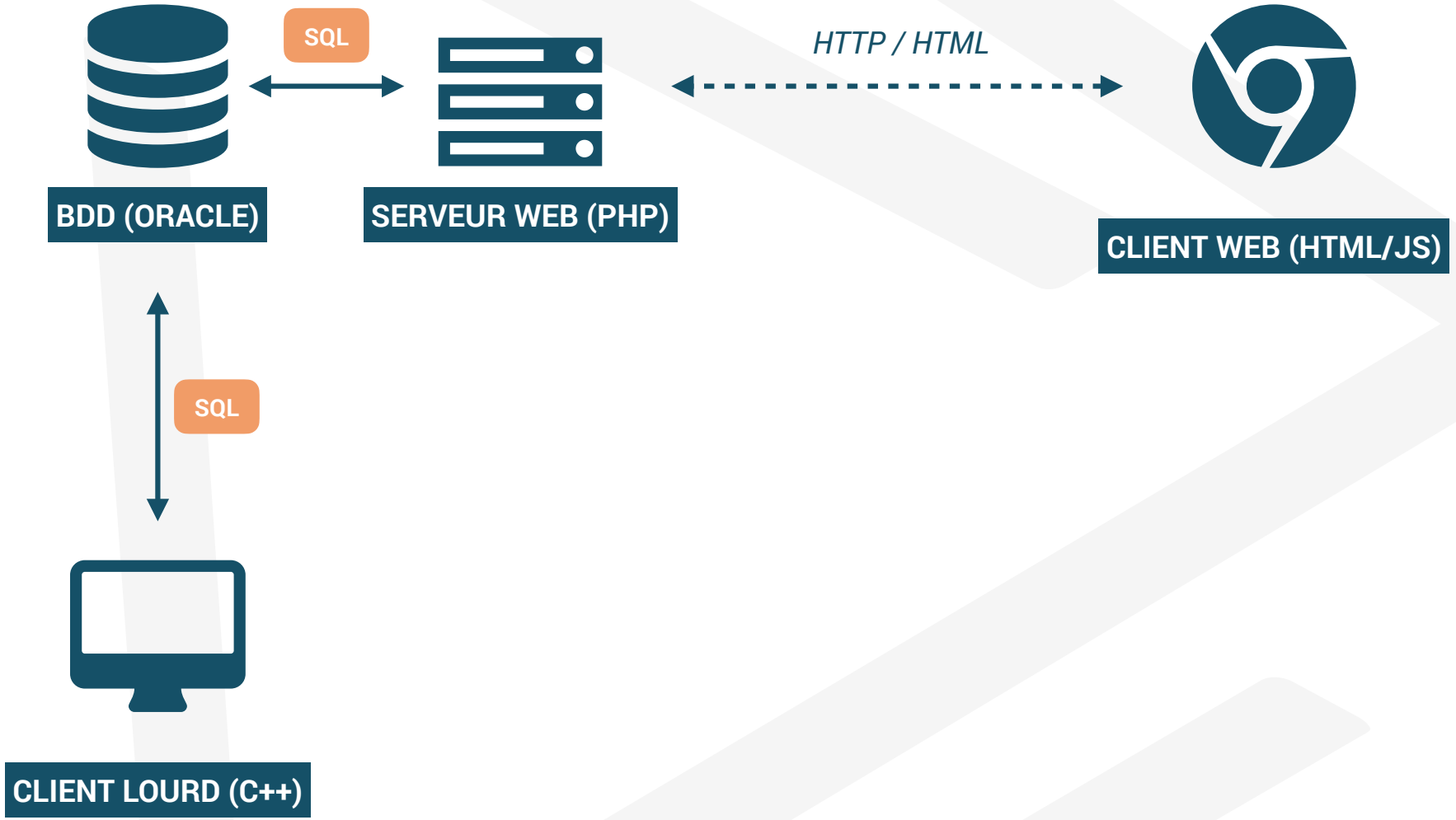
ÉVITER LES DOUBLONS

DÉVELOPPEMENT : ÉVITER LES DOUBLONS

- Au niveau du code
  - Répétition d'un morceau de code identique ou quasi-identique
  - Souvent le fruit d'un copier-coller avec ou sans adaptation
  - Nécessite de répéter les opérations de maintenance
- Au niveau d'un éco-système
  - Une même application en différentes déclinaisons (desktop, web, mobiles)
  - Doublons inévitables
  - Multiplication des coûts de développement et de maintenance
- Nécessité de mutualiser au maximum

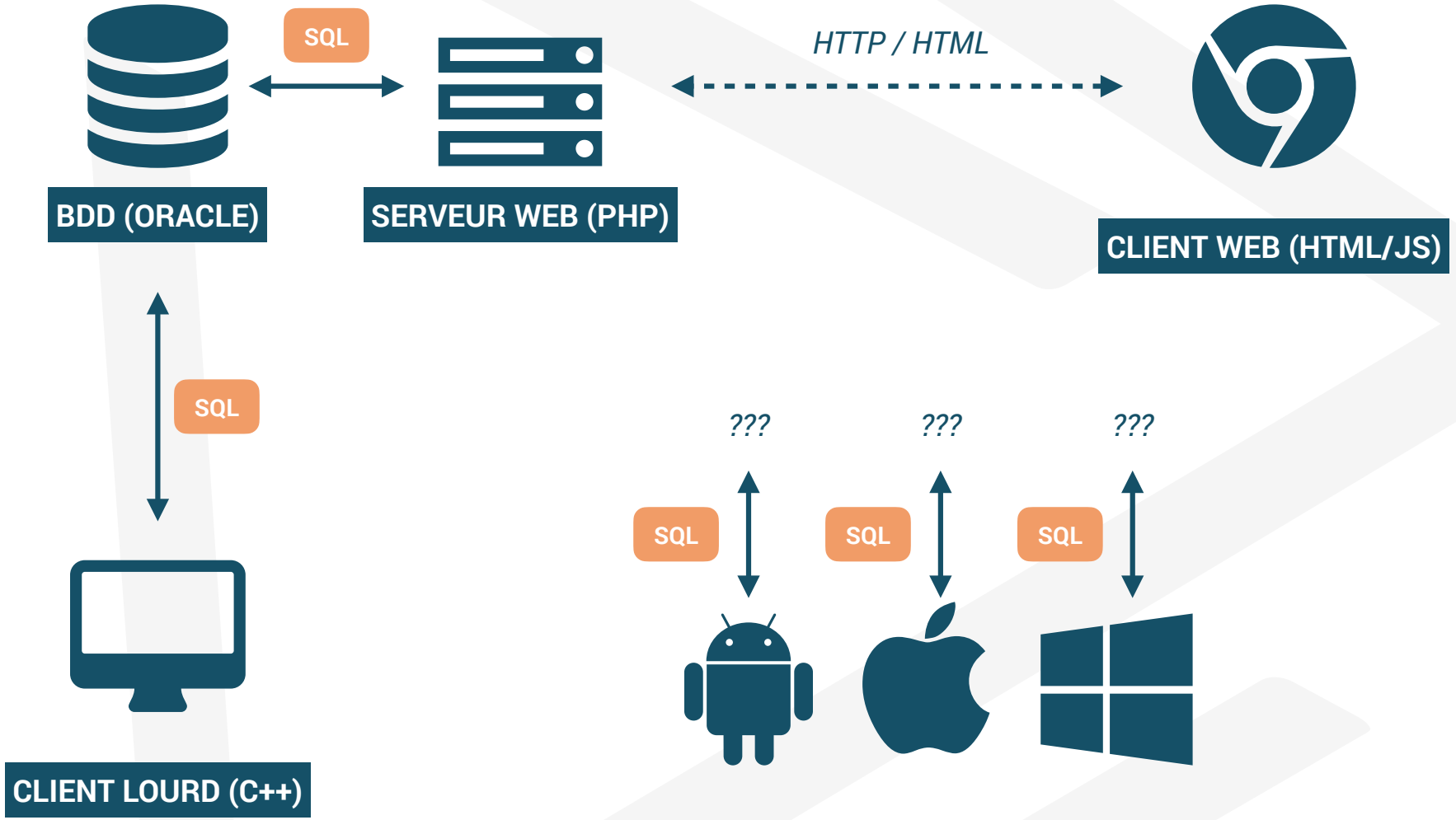
EXEMPLE TYPE ALLOCINE (FICTIF)

- Fin des années 90 : lancement de la plateforme
- Développement d'un client lourd Windows C++ interne à l'entreprise et du site Web



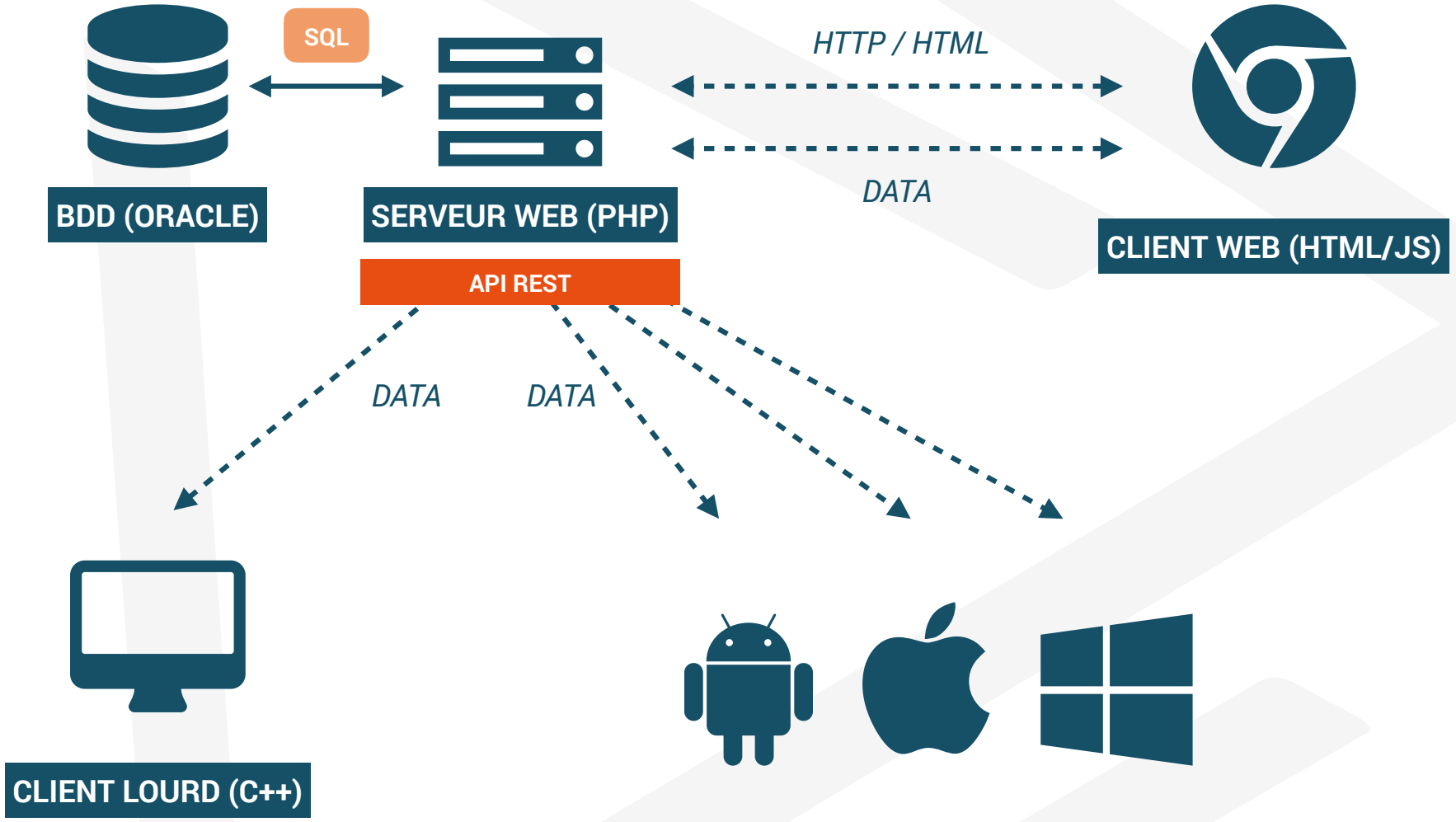
EXEMPLE TYPE ALLOCINE (FICTIF)

- Fin des années 2000 : ajout des clients mobiles
- iOS = Objective C, Android = Java, Windows = C#, chacun a son interface propre
- Problème : comment se connecter à la base de données ? (sécurité)



EXEMPLE TYPE ALLOCINE (FICTIF)

- Solution :
  - Mise en place d'un service Web comme moyen unique d'accéder aux données
  - Utilisation du protocole HTTP seul moyen universel (navigateur) : API REST



API REST

- Principe d'une API REST
  - S'appuyer sur le protocole HTTP
  - Utilisation du vocabulaire HTTP (GET, POST, PUT, DELETE)
  - Mise en place de routes ou endpoints via des URL
  - N'impose rien de plus

- Pourquoi HTTP ?
  - Protocole simple
  - Utilisé et éprouvé depuis les années 90
  - Protocole sans état (sans connexion permanente)
  - Intégré dans les navigateurs
  - Disponible dans la plupart des langages
  - Peut traverser les proxys et firewall

API REST

- REST = REpresentational State Transfert
  - Style d'architecture créé par Roy Fielding en 2000
  - Très vite devenu populaire
  - JSON est souvent associé pour le format d'échange, mais n'est pas imposé

API REST

- Principe : associer une URL et une ou plusieurs méthodes HTTP pour manipuler les données

Méthode HTTP	https://monapi/utilisateurs/	https://monapi/utilisateurs/id
GET	Retourne tous les utilisateurs	Retourne l'utilisateur "id"
PUT	Remplace tous les utilisateurs par un autre ensemble	Remplace les données de l'utilisateur
POST	Crée un nouvel utilisateur	*Généralement non utilisée*
DELETE	Supprime toute la collection	Supprime l'utilisateur "id"

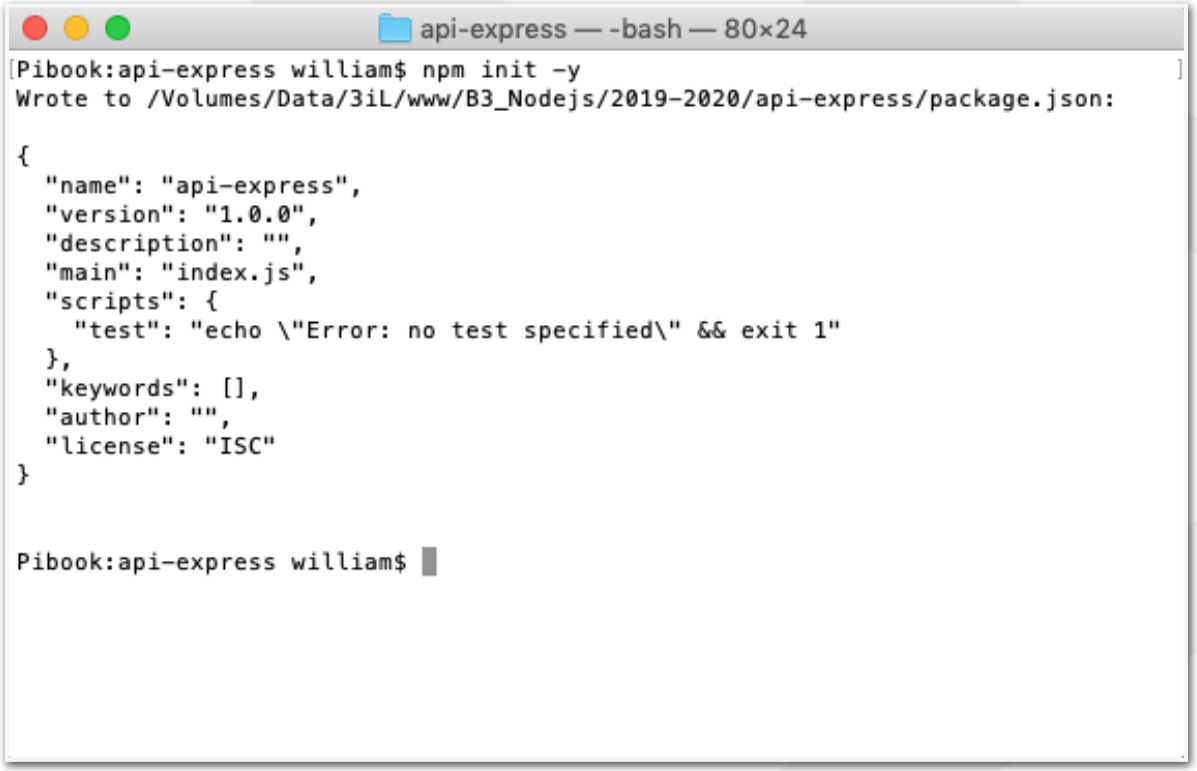
API REST

- Chaque URL est une terminaison ou endpoint
- Il n'y a aucune obligation à ce que chaque endpoint ait toutes les opérations, on adapte suivant les besoins et ce que l'on souhaite présenter aux utilisateurs

API REST avec Express
Partie 2

INITIALISATION DU PROJET

- Créer un dossier `api-express`
- Exécuter `npm init -y` pour répondre automatiquement oui à toutes les questions.



INITIALISATION DU PROJET

- Installer express et body-parser

```
npm install express body-parser
```

- Installer nodemon en dépendance de développement

```
npm install nodemon --save-dev
```

- Créer un fichier `index.js`
- Modifier le fichier `package.json` pour ajouter le script `start` comme suit :

/package.json (extrait) :

```
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1",
8     "start": "nodemon index.js"
9   },
```

- Lancer le projet

```
npm start
```

FOURNITURE DES DONNÉES

- Notre projet va travailler sur des données "Profs".
- Nous allons nous efforcer de simuler l'accès à une base de données au travers d'un modèle.
- Le modèle mettra en place les accès classiques sous la forme de fonctions :
  - `getAll()` : retourne tous les profs
  - `getById()` : retour un prof par son `id`
  - `add()` : ajoute un prof
  - `update()` : modifie les données d'un prof
  - `delete()` : supprime un prof

- Afin de ne pas alourdir/complexifier trop le code, nous travaillerons à partir d'un tableau d'objet.
- Pour simuler le fonctionnement d'une base de données, les `id` des profs ne seront pas des entiers consécutifs commençant à 1.

FOURNITURE DES DONNÉES

- Créer un dossier `/data` puis un fichier `/data/profs.js`
- Mettre en place le tableau suivant :

/data/profs.js :

```
let profs = [
    {
        id:187,
        nom:"Ruchaud",
        prenom: "William",
        bureau: "210"
    },
    {
        id:24,
        nom:"Oulad Moussa",
        prenom: "Mustapha",
        bureau: "213"
    },
    {
        id:23,
        nom:"Chervy",
        prenom: "Benjamin",
        bureau: "218"
    }
];
```

GETALL()

- Pour gérer les `id` des profs que nous créerons, nous ajoutons une variable maxId à laquelle nous donnons la valeur d'id la plus grande actuellement.
- Pour la fonction `getAll()` qui retourne tous les profs, nous allons rester très basique, elle n'assurera aucun tri, ni filtrage.

/data/profs.js (extrait) :

```
14      {
15          id:23,
16          nom:"Chervy",
17          prenom: "Benjamin",
18          bureau: "218"
19      }
20  ];
21  let maxId = 187;
22
23  function getAll () {
24      return profs;
25  }
```

EXPORT ET TEST

- Pour que notre fonction `getAll()` soit utilisable en dehors du fichier `profs.js`, il faut l'exporter.
- Il faudra réaliser cet export pour toutes les autres fonctions que nous rajouterons par la suite.

/data/profs.js (extrait) :

```
23  function getAll () {
24      return profs;
25  }
26
27  module.exports = {
28      getAll : getAll
29  }
```

EXPORT ET TEST

- Pour tester nous allons juste ajouter un `require` dans `index.js` et faire un affichage de ce que retourne `getAll()`.

/index.js :

```
1   const profs = require('./data/profs');
2
3   console.log(profs.getAll());
```

- Le contenu du tableau a dû apparaître dans la console.

```
[nodemon] starting `node index.js`
[
  { id: 187, nom: 'Ruchaud', prenom: 'William', bureau: '210' },
  { id: 24, nom: 'Oulad Moussa', prenom: 'Mustapha', bureau: '213' },
  { id: 23, nom: 'Chervy', prenom: 'Benjamin', bureau: '218' }
]
[nodemon] clean exit - waiting for changes before restart
```

GETBYID()

- La fonction `getById()` va nous permettre de récupérer les données d'un prof par son id.
- Elle renverra `null` si elle ne trouve pas l'id demandé.
- Nous utiliserons `array.filter()` de JavaScript pour effectuer la recherche.

/data/profs.js (extrait) :

```
23  function getAll () {
24      return profs;
25  }
26
27  function getById (id)  {
28      let res = profs.filter((p) => {
29          return p.id == id;
30      })
31      if(res.length == 1){
32          return res[0];
33      }
34      return null;
35  }
36
37  module.exports = {
38      getAll : getAll,
39      getById : getById
40  }
```

GETBYID() : LES TESTS

- Dans `/index.js` ajouter des tests avec un cas qui fonctionne et un autre qui doit renvoyer `null` et vérifier les résultats.

/index.js :

```
1  const profs = require('./data/profs');
2
3  console.log(profs.getAll());
4
5  // Doit retourner William Ruchaud
6  console.log(profs.getById(187));
7
8  // Doit retourner null
9  console.log(profs.getById(1));
```

```
[nodemon] starting `node index.js`
[
  { id: 187, nom: 'Ruchaud', prenom: 'William', bureau: '210' },
  { id: 24, nom: 'Oulad Moussa', prenom: 'Mustapha', bureau: '213' },
  { id: 23, nom: 'Chervy', prenom: 'Benjamin', bureau: '218' }
]
{ id: 187, nom: 'Ruchaud', prenom: 'William', bureau: '210' }
null
[nodemon] clean exit - waiting for changes before restart
```

ADD()

- La fonction `add()` rajoute un nouveau prof au plus grand id+1.
- Nous partirons du principe qu'elle reçoit des données complètes et correctes.
- Nous utiliserons `array.push()` de JavaScript.
- Elle retourne `true` et a peu de chances d'échouer à notre échelle.
- Bien penser à l'ajouter à l'export.

/data/profs.js (extrait) :

```
37  function add  (nom, prenom, bureau)  {
38      maxId++;
39      profs.push({
40          id:maxId,
41          nom:nom,
42          prenom:prenom,
43          bureau:bureau
44      })
45      return true;
46  }
47
48  module.exports = {
49      getAll : getAll,
50      getById : getById,
51      add: add
52  }
```

ADD() : LE TEST

- Dans `/index.js` faire un ajout de prof tout en observant bien l'id du prof ajouté qui doit être le plus grand id+1 (ici 188)

/index.js :

```
1  const profs = require('./data/profs');
2
3  console.log(profs.getAll());
4
5  profs.add("Gaudin","Hervé",217);
6
7  // Hervé Gaudin doit avoir pour id 188
8  console.log(profs.getAll());
```

```
[nodemon] starting `node index.js`
[
  { id: 187, nom: 'Ruchaud', prenom: 'William', bureau: '210' },
  { id: 24, nom: 'Oulad Moussa', prenom: 'Mustapha', bureau: '213' },
  { id: 23, nom: 'Chervy', prenom: 'Benjamin', bureau: '218' }
]
[
  { id: 187, nom: 'Ruchaud', prenom: 'William', bureau: '210' },
  { id: 24, nom: 'Oulad Moussa', prenom: 'Mustapha', bureau: '213' },
  { id: 23, nom: 'Chervy', prenom: 'Benjamin', bureau: '218' },
  { id: 188, nom: 'Gaudin', prenom: 'Hervé', bureau: 217 }
]
[nodemon] clean exit - waiting for changes before restart
```

DELETEBYID()

- La fonction `deleteById()` supprime un prof par son id.
- Nous utiliserons à nouveau `array.filter()` pour produire une nouvelle version du tableau en ne conservant que les profs dont l'id est différent de celui à supprimer.
- La fonction retourne `true` ou `false` suivant qu'une suppression a eu lieu ou non.
- Bien penser à l'ajouter à l'export.

/data/profs.js (extrait) :

```
48  function deleteById (id) {
49      let l = profs.length;
50      profs = profs.filter((p) => {
51          return p.id != id;
52      })
53      return l != profs.length;
54  }
55
56  module.exports = {
57      getAll : getAll,
58      getById : getById,
59      add: add,
60      deleteById:deleteById
61  }
```

DELETEBYID() : LES TESTS

- Dans `/index.js` faire un test avec un prof qui existe et un qui n'existe pas.

/index.js :

```
1   const profs = require('./data/profs');
2
3   console.log(profs.getAll());
4
5   // Doit supprimer William Ruchaud et retourner true
6   console.log(profs.deleteById(187));
7
8   // Doit retourner false
9   console.log(profs.deleteById(1));
10
11  console.log(profs.getAll());
12
```

```
[nodemon] starting `node index.js`
[
  { id: 187, nom: 'Ruchaud', prenom: 'William', bureau: '210' },
  { id: 24, nom: 'Oulad Moussa', prenom: 'Mustapha', bureau: '213' },
  { id: 23, nom: 'Chervy', prenom: 'Benjamin', bureau: '218' }
]
true
false
[
  { id: 24, nom: 'Oulad Moussa', prenom: 'Mustapha', bureau: '213' },
  { id: 23, nom: 'Chervy', prenom: 'Benjamin', bureau: '218' }
]
[nodemon] clean exit - waiting for changes before restart
```

UPDATE()

- La fonction `update()` met à jour un prof dont l'id est renseignées.
- Cette fonction utilisera `getById()` pour obtenir l'objet à mettre à jour.
- Elle retourne `true` si la mise à jour à eu lieu, `false` sinon.
- Penser à l'ajouter à l'export.

/data/profs.js (extrait) :

```
56  function update (id, nom, prenom, bureau) {
57      let prof = getById(id);
58      if(!prof) return false;
59
60      prof.nom = nom;
61      prof.prenom = prenom;
62      prof.bureau = bureau;
63      return true;
64  }
65
66  module.exports = {
67      getAll : getAll,
68      getById : getById,
69      delete: deleteById,
70      add: add,
71      update: update
72  }
73
74
```

UPDATE() : LES TESTS

- Dans `/index.js` faire un test avec un prof qui existe et un qui n'existe pas.
- Le bureau de William Ruchaud doit passer de 210 à 217.

/index.js :

```
1   const profs = require('./data/profs');
2
3   console.log(profs.getAll());
4
5   // Modification qui doit retourner true
6   console.log(profs.update(187,"Ruchaud","William",217));
7
8   // Modification qui doit retourner false
9   console.log(profs.update(1,"Ruchaud","William",217));
10
11  console.log(profs.getAll());
```

```
[nodemon] starting `node index.js`
[
  { id: 187, nom: 'Ruchaud', prenom: 'William', bureau: '210' },
  { id: 24, nom: 'Oulad Moussa', prenom: 'Mustapha', bureau: '213' },
  { id: 23, nom: 'Chervy', prenom: 'Benjamin', bureau: '218' }
]
true
false
[
  { id: 187, nom: 'Ruchaud', prenom: 'William', bureau: 217 },
  { id: 24, nom: 'Oulad Moussa', prenom: 'Mustapha', bureau: '213' },
  { id: 23, nom: 'Chervy', prenom: 'Benjamin', bureau: '218' }
]
[nodemon] clean exit - waiting for changes before restart
```

FIN DU MODÈLE

- Nous en avons fini avec notre modèle.
- Évidemment, il y aurait beaucoup de chose à faire et mettre en place, mais au moins nous avons un CRUD de base qui va nous permettre de construire une petite API REST.
- À garder en tête : à chaque redémarrage, donc à chaque sauvegarde de fichier nous repartons avons uniquement les 3 profs initiaux.

API REST avec Express
Partie 3

MISE EN PLACE D'EXPRESS

- Dans `/index.js`, effacer le contenu de test et mettre en place une base pour lancer express en écoute sur le port 8080 avec une réponse simple sur l'URL / en get.

/index.js :

```
1   const express = require('express');
2
3   const app = express();
4
5   app.get('/', (req,res) => {
6       res.send("API REST avec Express");
7   })
8
9   app.listen(8080,()=>{
10      console.log("Ecoute du port 8080");
11  })
```

- Tester :
  - "Ecoute du port 8080" doit apparaître dans la console
  - Dans un navigateur l'URL `http://localhost:8080/` doit répondre "API REST avec Express"

PREMIÈRE ROUTE

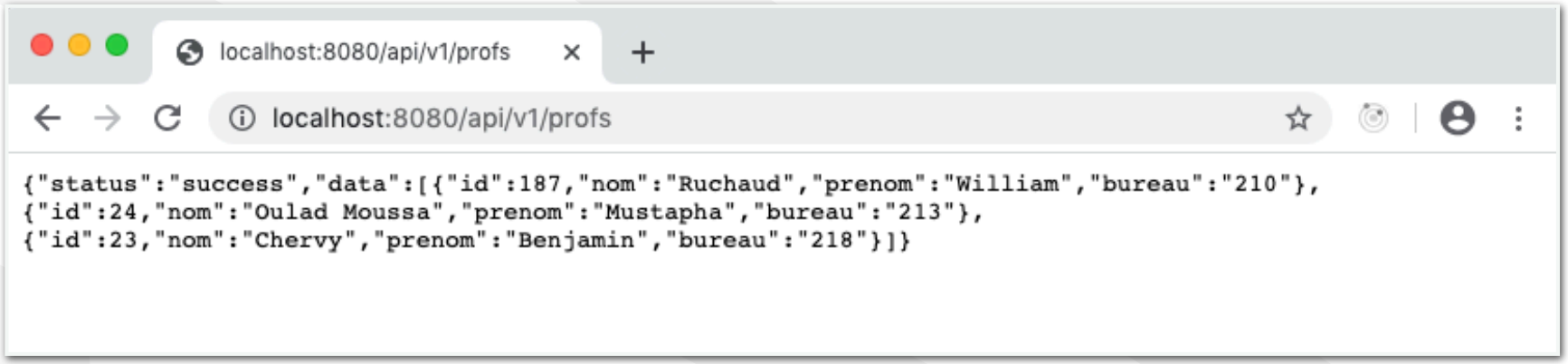
- Ajouter la première route en GET vers /api/v1/profs

/index.js :

```
1   const express = require('express');
2   const profs = require('./data/profs');
3
4   const app = express();
5
6   app.get('/', (req,res) => {
7       res.send("API REST avec Express");
8   })
9
10  app.get('/api/v1/profs', (req,res) => {
11      res.status(200).json(
12          {
13              "status":"success",
14              "data":profs.getAll()
15          });
16  })
17
18  app.listen(8080,()=>{
19      console.log("Ecoute du port 8080");
20  })
21
```

PREMIER TEST

- Parce que c'est une route en GET, elle peut être testée dans un navigateur.

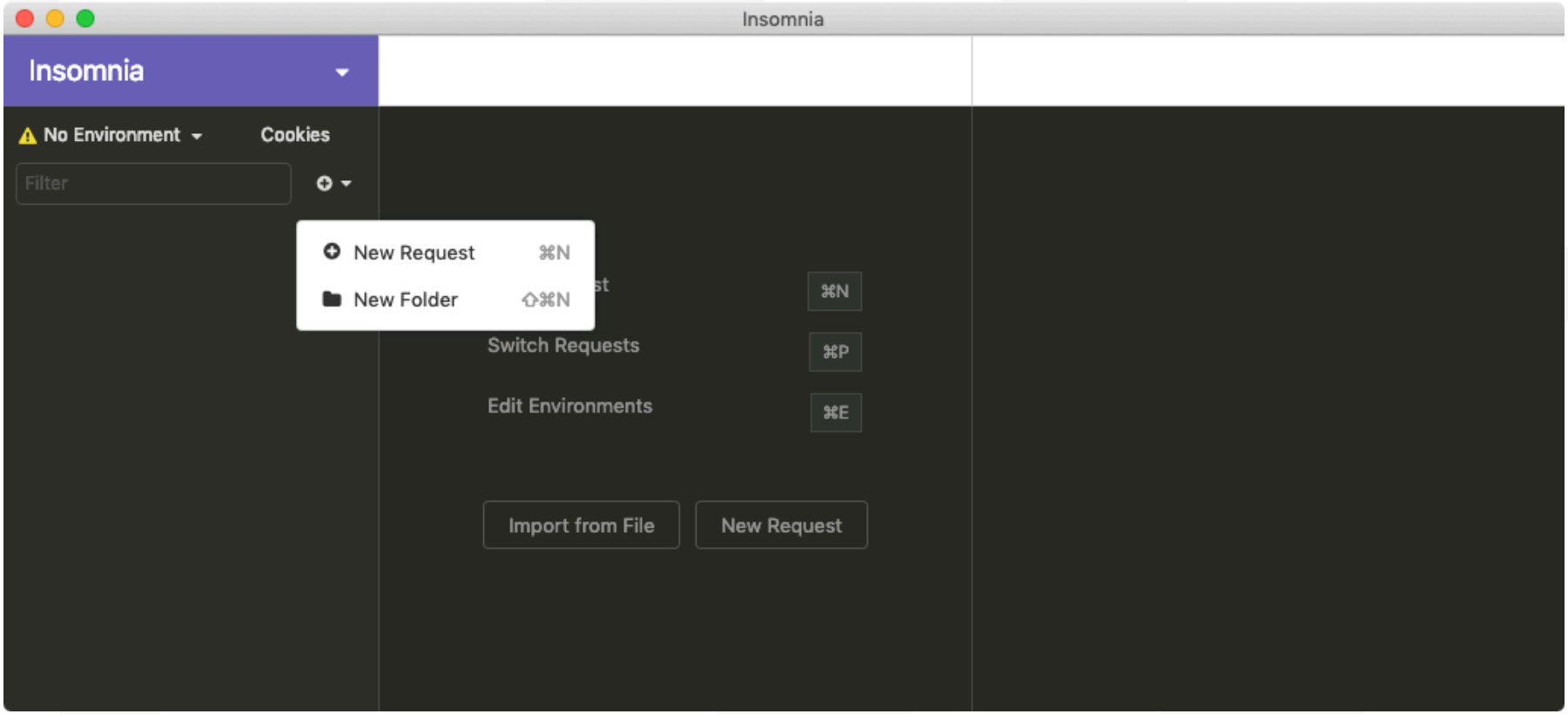


- Il n'y a pas de formatage particulier, les données sont listées au kilomètre.

- Un navigateur ne nous permettra pas de tester plus que les requêtes GET sans effectuer de codage.
- C'est pourquoi il nous faut un outil spécialisé pour tester les API, il en existe plusieurs gratuits :
  - Insomnia (que j'utiliserai)
  - Postman qui fait partie des plus célèbres

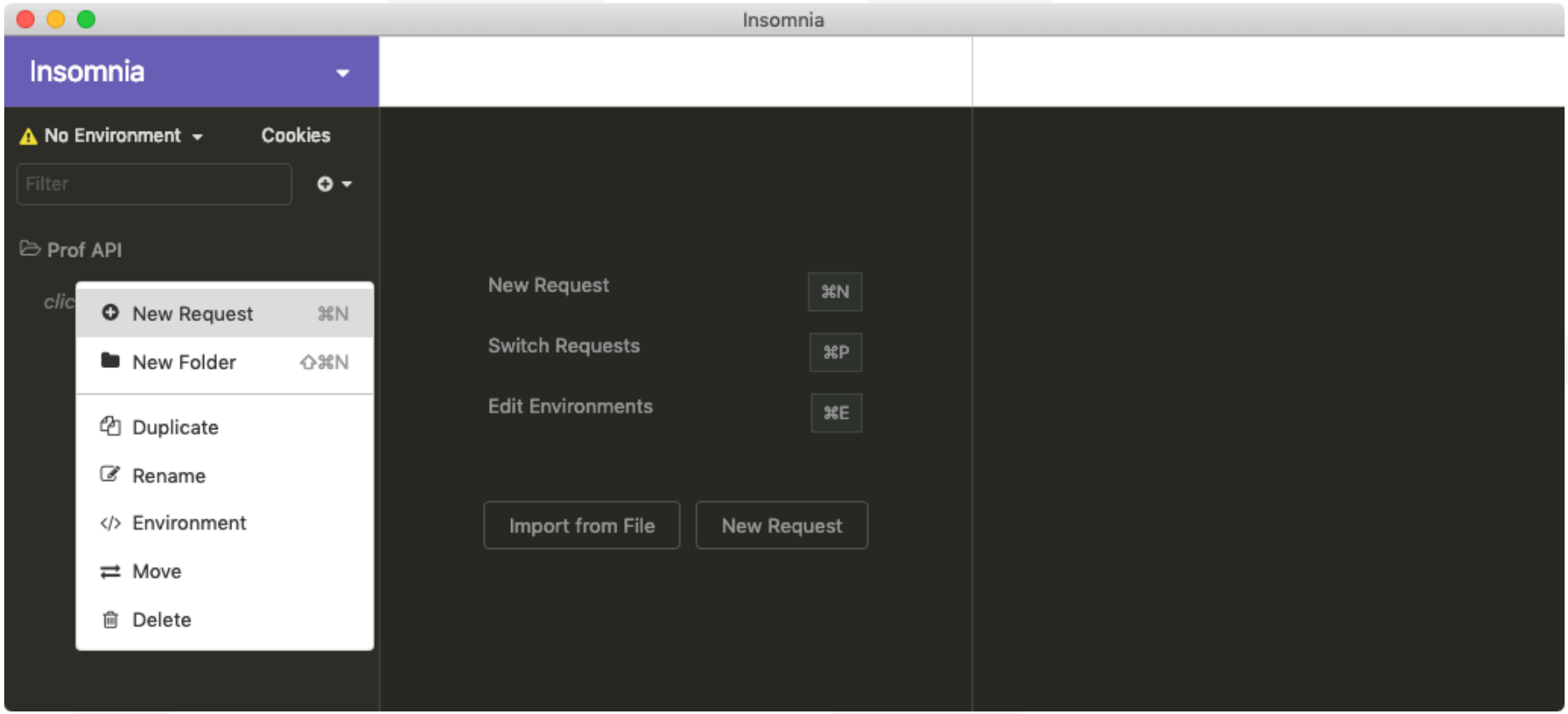
INSOMNIA

- Lancer Insomnia
- Histoire de démarrer proprement, je vous conseille de faire un dossier pour classer vos tests.
- Avec le bouton +, créer un dossier (Folder) et le nommer "Profs API".



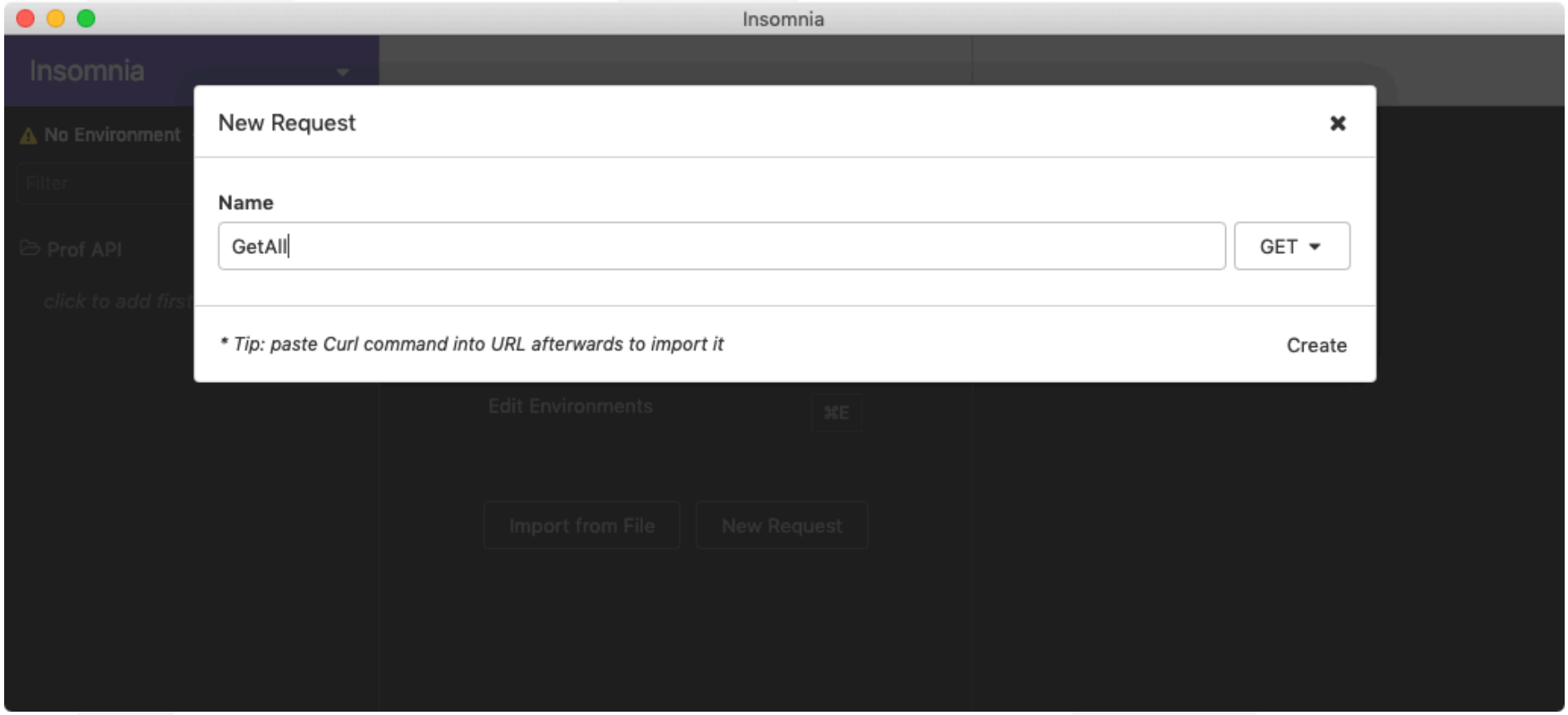
INSOMNIA

- En survolant le nom du dossier, cela fait apparaître une petite flèche qui permet d'ouvrir un menu contextuel.
- Y sélectionner New Request.



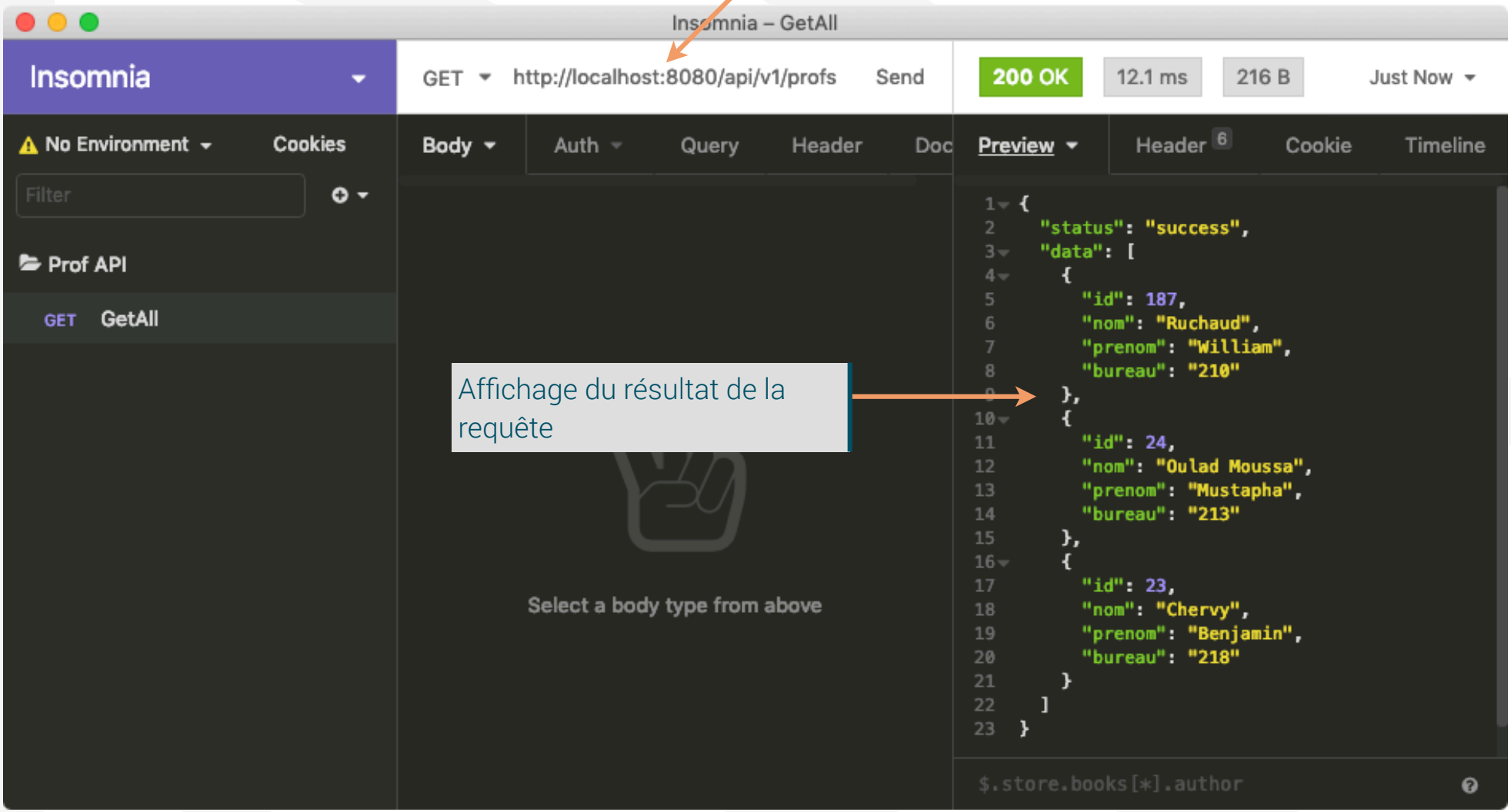
INSOMNIA

- Dans la boîte de dialogue rentrer un nom pour la requête : "getAll"



INSOMNIA

- Saisir l'URL
- Appuyer sur Send



À PROPOS DES ROUTES

- Notre première route est donc `/api/v1/profs`
- Le premier élément, `/api/` est assez évident. Il permet de faire la distinction entre des URLs destinées à un navigateur et celles destinées à des clients API.
- Le second est `v1`. C'est une indication du numéro de version.
  - Il faut voir les choses sur le long terme !
  - Si vous êtes une société comme Allociné et que vous vendez l'accès à vos données via une API (pratique courante), des clients vont dépendre de vos API.
  - Quelque part vous vous engagez à fournir un service stable dans le temps à ces clients (vous ne vous appelez pas Google)
  - Si votre API doit évoluer, vous pouvez modifier la liste des routes en v2, v3…
- Attention, il s'agit que de la partie routes de l'API, en gros la partie service.
- La partie traitement peut complètement changer derrière, tant qu'elle fournit le même service.

À PROPOS DU RÉSULTAT

- Petit rappel : REST n'est qu'un style d'architecture qui n'impose pas grand chose en terme de retour (format, structure, code).
- Il en résulte des pratiques différentes, chacune souvent avec d'excellentes raisons.

- Nous avons mis en place le retour suivant :
  `res.status(200).json(…)`
  - `status(xxx)` nous permet de renseigner un code d'état HTTP.
  - 200 est le code pour "succès de la requête"
  - `json()` pour encoder un résultat au format JSON.

- Cela fait partie des bonnes pratiques que d'indiquer le code d'état HTTP.

PRINCIPAUX CODES HTTP

- Il y a 5 familles de codes HTTP à 3 chiffres (source Wikipedia) :
  - 1xx : Information
  - 2xx : Succès
  - 3xx : Redirection
  - 4xx : Erreur du client Web
  - 5xx : Erreur du serveur

- Principaux codes 2xx

Code	Message	Explication
200	OK	Requête traitée avec succès
201	Created	Requête traitée avec succès et création d'un document
202	Accepted	Requête traitée avec succès mais sans garantie de résultat
204	No Content	Requête traitée avec succès, mais aucun contenu à renvoyer

PRINCIPAUX CODES HTTP

Principaux codes 4xx

Code	Message	Explication
400	Bad Request	La syntaxe de la requête est erronée
401	Unauthorized	Une authentification est nécessaire pour accéder à la ressource
403	Forbidden	Le serveur a compris la requête, mais refuse de l'exécuter, problème de droits d'accès
404	Not Found	Ressource non trouvée
405	Method Not Allowed	Méthode de requête non autorisée
409	Conflict	La requête ne peut pas être traitée en l'état actuel

Principaux code 5xx

Code	Message	Explication
500	Internal Server Error	Erreur interne du serveur
501	Not Implemented	Fonctionnalité réclamée non supportée par le serveur
503	Service Unavailable	Service temporairement indisponible ou en maintenance

STRUCTURE DU RÉSULTAT

- Pourquoi indiquer un statut dans le résultat alors qu'il y en a déjà un dans le code HTTP ?
- C'est une pratique courante dans les API.
- Dans le cas d'une réponse "correcte", il n'y a pas grand chose à dire.
- Dans le cas d'une réponse incorrecte, les codes de statut HTTP ne sont pas assez riches et précis, il faut souvent rajouter un message qui précise la nature de l'erreur.
- C'est pourquoi, il est courant d'avoir une structure standardisée des données dans un cas comme dans l'autre.

```
1  {
2      "status":"success",
3      "data":[]
4  }
```

```
1  {
2      "status":"error",
3      "message":"votre message d'erreur"
4  }
```

API REST avec Express
Partie 4

ROUTES À METTRE EN PLACE

- Voici la liste totale des routes que nous souhaitons mettre en place pour gérer nos profs

Méthode	Route	Fonction
GET	/api/v1/profs	Liste tous les profs
GET	/api/v1/profs/:id	Détaille un prof à partir de son id
POST	/api/v1/profs	Ajoute un prof
PUT	/api/v1/profs/:id	Met à jour un prof à partir de son id
DELETE	/api/v1/profs/:id	Supprime un prof à partir de son id

- Je rappelle que nous faisons un CRUD minimaliste et que notre API de travaille qu'avec une seule entité.
- On comprends assez vite que le fichier `/index.js` va vite se trouver encombré et devenir illisible.
- Il y a nécessité à structurer l'application.

PRINCIPE DE LA SOLUTION

- Nous allons éclater le code en plusieurs fichiers.
  - Un fichier contrôleur qui va effectuer tous les traitements, ceux de `/data/profs.js` ainsi que les vérifications nécessaires
  - Un fichier routeur qui fera le lien entre méthode+route et un traitement du contrôleur.

- Express permet assez facilement de créer des routeurs et de lui indiquer de s'en servir.
- En parallèle nous rajouterons `body-parser` dans `/index.js` car nos opérations en aurons besoin.

MISE EN PLACE DU CONTRÔLEUR

- Créer un dossier `/controllers`
- Créer un fichier `/controllers/profsController.js`
- Y copier le code suivant :

/controllers/profsController.js :

```
1   const profs = require('../data/profs');
2
3   module.exports = {
4       getAll : function(req,res) {
5           res.status(200).json({
6               'status':'succes',
7               'data'   : profs.getAll()
8           })
9       },
10
11      getById:function(req,res)    { res.send("profs.getById"); },
12      add:function(req,res)        { res.send("profs.add"); },
13      update:function(req,res)     { res.send("profs.update"); },
14      delete:function(req,res)     { res.send("profs.delete"); }
15  }
```

- Commentaire : le code des autres fonction est un code provisoire "tassé" pour la capture, il sera modifié prochainement

MISE EN PLACE DU ROUTEUR

- Créer un dossier `/routers`
- Créer un fichier `/routers/profsRouter.js`
- Y copier le code suivant :

/routers/profsRouter.js :

```
1   const express = require('express');
2   const profsController = require('../controllers/profsController');
3
4   function buildRoutes() {
5       let router = express.Router();
6
7       router.route('/profs/').get(profsController.getAll);
8       router.route('/profs/:id').get(profsController.getById);
9       router.route('/profs/').post(profsController.add);
10      router.route('/profs/:id').delete(profsController.delete);
11      router.route('/profs/:id').put(profsController.update);
12
13      return router;
14  }
15
16  exports.router = buildRoutes();
```

- Commentaire : le code est assez simple à comprendre, à chaque fois on associe une route avec une fonction du contrôleur.

MODIFICATION DE /INDEX.JS

- Modifier le code de `/index.js` et supprimer la première route de profs.

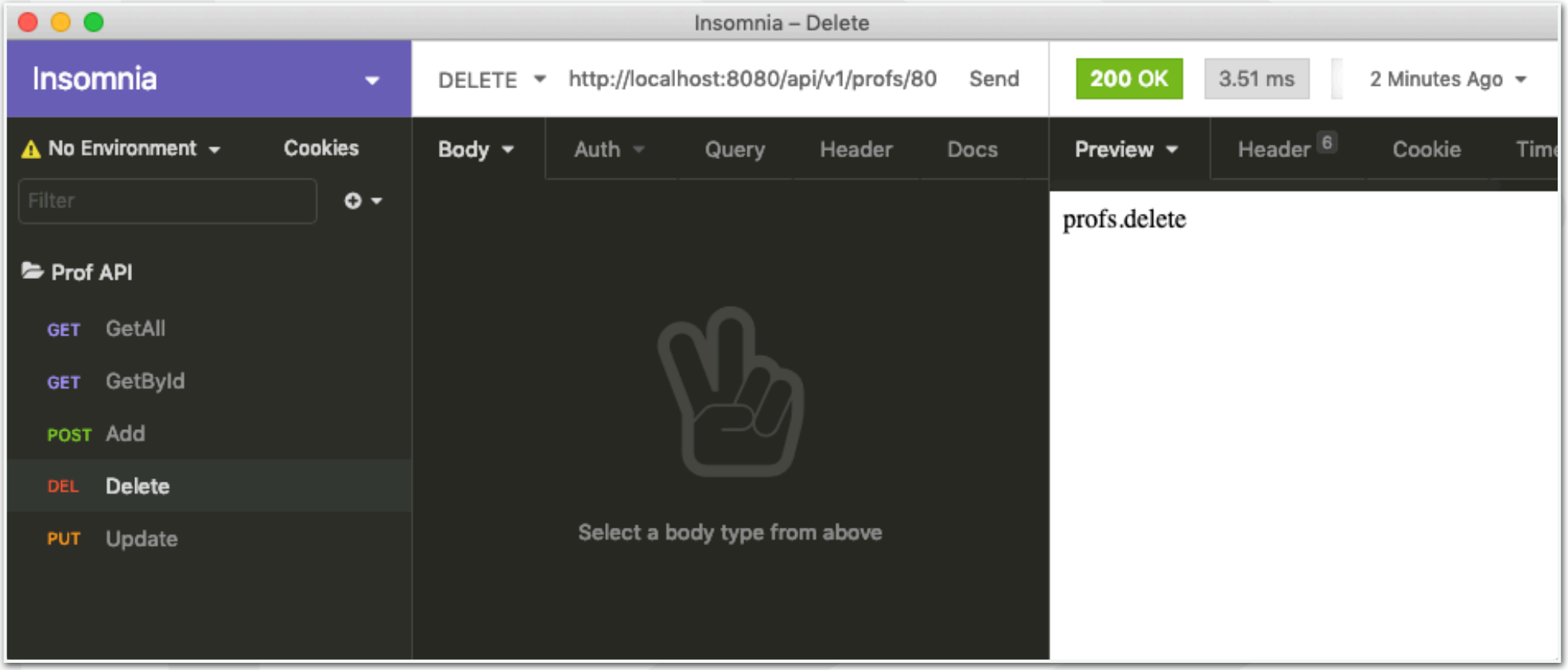
/index.js :

```
1   const express = require('express');
2   const bodyParser = require('body-parser');
3
4   const profsRouter = require('./routers/profsRouter').router;
5
6   const app = express();
7   app.use(bodyParser.urlencoded({extended:true}));
8   app.use(bodyParser.json());
9
10  app.use('/api/v1',profsRouter);
11
12  app.get('/', (req,res) => {
13      res.send("API REST avec Express");
14  })
15
16  app.listen(8080,()=>{
17      console.log("Ecoute du port 8080");
18  })
19
```

- Commentaire : on retrouve le code pour utiliser `body-parser` et juste l'inclusion du routeur dédié au profs et sa mise en route avec `use()`.

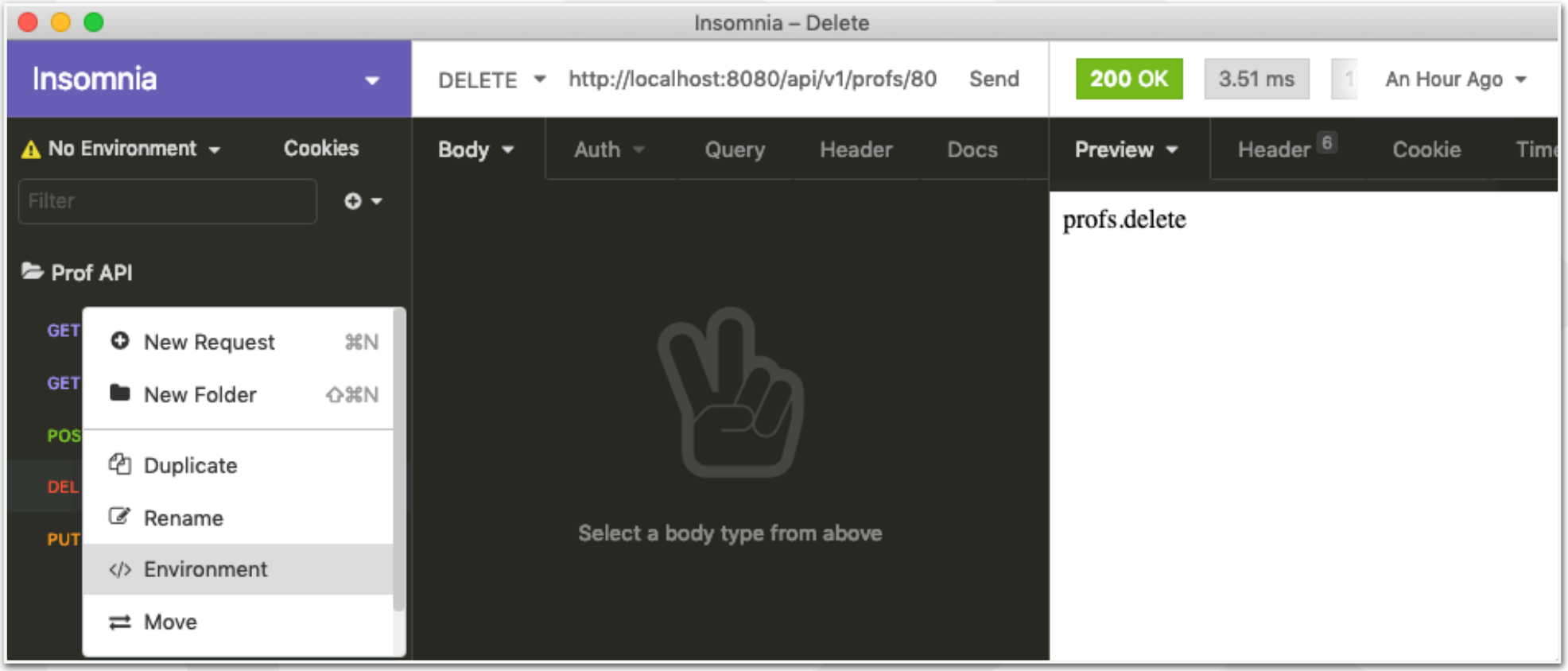
VÉRIFICATIONS

- Nous avons prévus des messages par défaut pour les différentes routes. Nous pouvons donc les tester avec Insomnia.
- Je vous recommande de procéder par duplication de GetAll pour éviter de devoir recopier la base de l'URL.
- Attention de bien rentrer un paramètre pour GetById, Delete et Update.



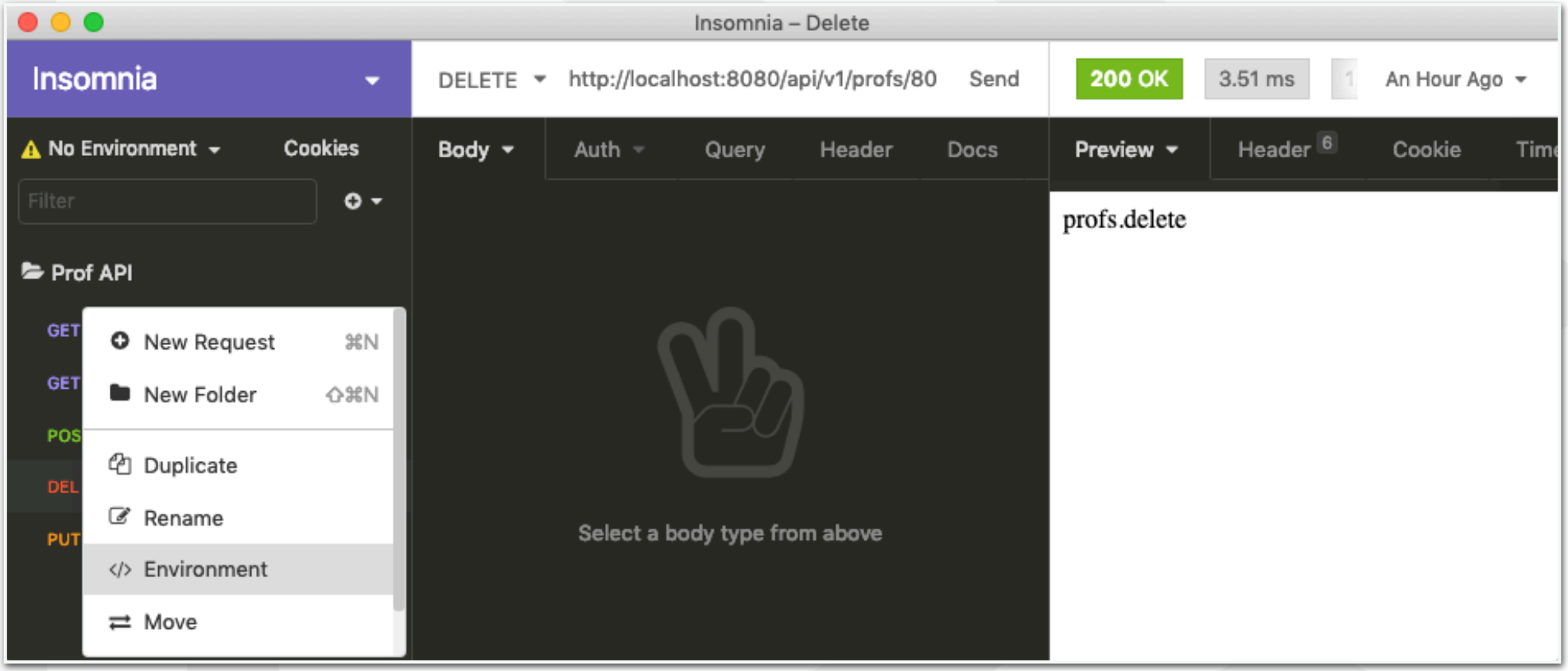
AMÉLIORATION

- La duplication nous a permis d'avancer rapidement, mais Insomnia permet mieux : les variables d'environnement.
- Aller dans le menu contextuel du dossier de nos tests et sélectionner "Environnment"



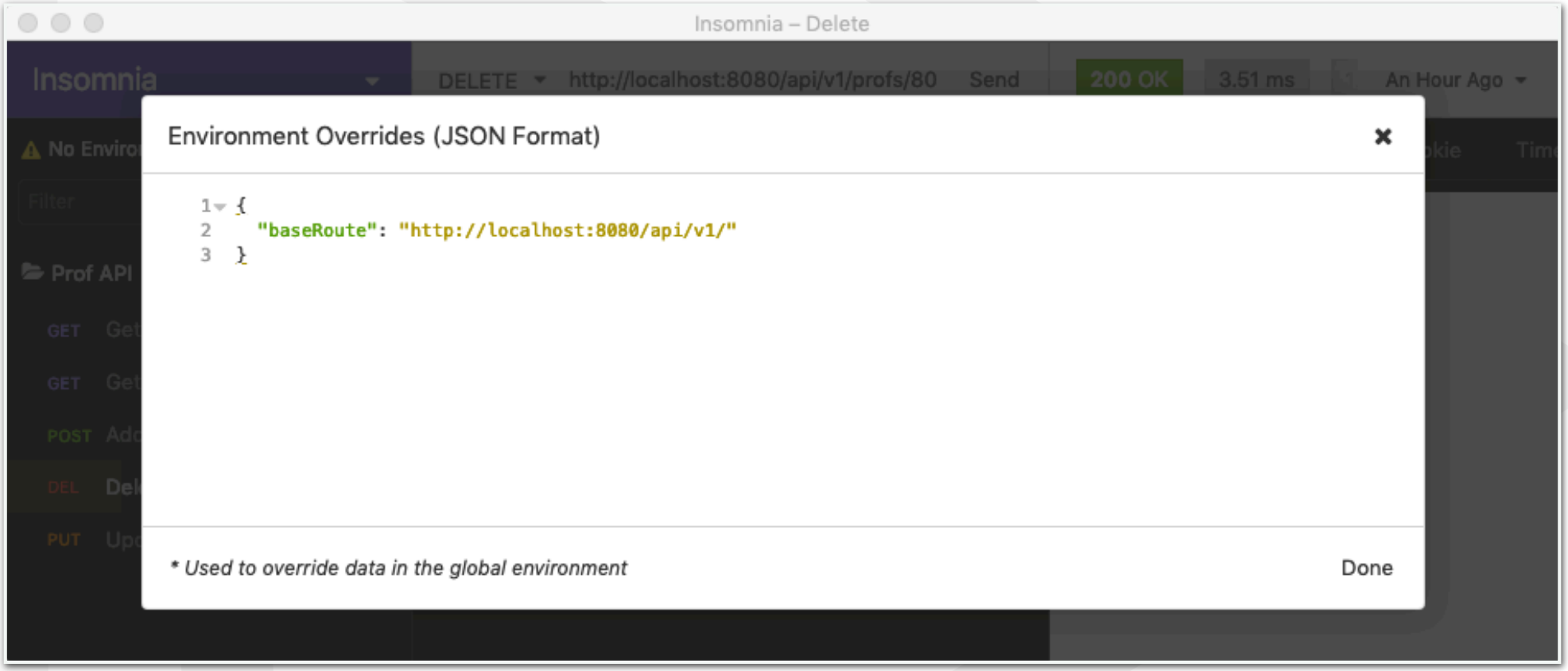
AMÉLIORATION

- La duplication nous a permis d'avancer rapidement, mais Insomnia permet mieux : les variables d'environnement.
- Aller dans le menu contextuel du dossier de nos tests et sélectionner "Environnment"



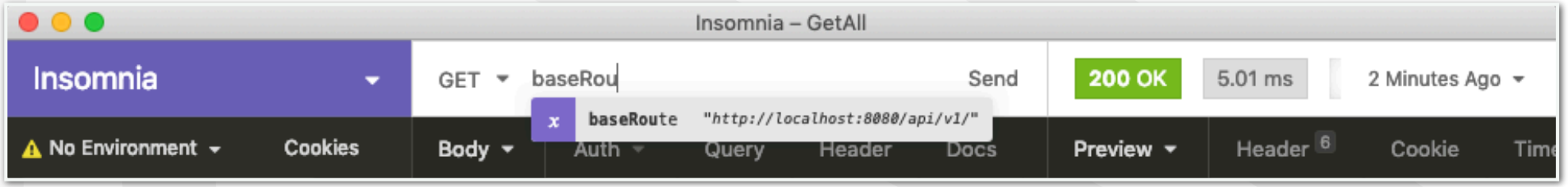
AMÉLIORATION

- Dans la boîte de dialogue, entrer au format JSON une variable "baseRoute" avec pour valeur "http://localhost:8080/api/v1/"

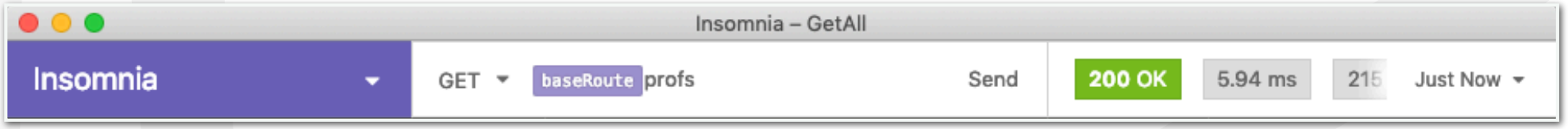


AMÉLIORATION

- Editer l'URL de GetAll :
  - Effacer tout
  - Commencer à taper `baseRou` puis demander la complétion pour choisir baseRoute qui apparaîtra avec un formatage particulier.



  - Ajouter `profs`



  - Tester avec Send, le résultat doit toujours être le même

- Faire de même pour les autres tests, et vérifier que le résultat ne change pas

AMÉLIORATION

- Nous avons à présent un système de routes fonctionnel
- Notre code est organisé en fichiers séparés permettant de se repérer plus facilement
- Il nous reste à rendre les actions effectives, le travail qui suit va donc se focaliser sur le fichier `/controllers/profsController.js`

API REST avec Express
Partie 5

PROCHAINES ÉTAPES

- Nous avons à présent un système de routes fonctionnel
- Notre code est organisé en fichiers séparés permettant de se repérer plus facilement
- Il nous reste à rendre les actions effectives, le travail qui suit va donc se focaliser sur le fichier `/controllers/profsController.js`

ROUTE GETBYID

- Rappel : l'id du prof arrivera en fin de la route

  Exemple `GET http://localhost:8080/api/v1/profs/187`

  devra retourner William Ruchaud

- Dans notre routeur, nous avons indiqué `/profs/:id` notre paramètre s'appèlera donc id (original)
- Il pourra être récupéré dans la variable de requête : `req.params.id`
- Au niveau des réponses, deux cas sont possibles :
  - Un prof possède l'id demandée : code 200
  - Aucun prof ne le possède : code 404

- Nous allons rester simple, et pas rajouter de vérification sur la nature de l'id.

ROUTE GETBYID

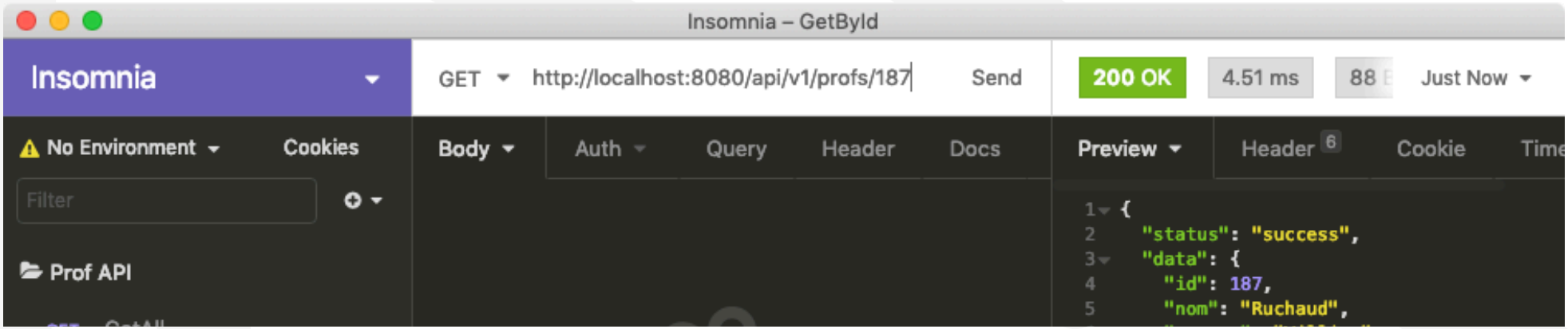
- Dans le fichier `/controllers/profsController.js` modifier la fonction `getById()`.

/controllers/profsController.js :

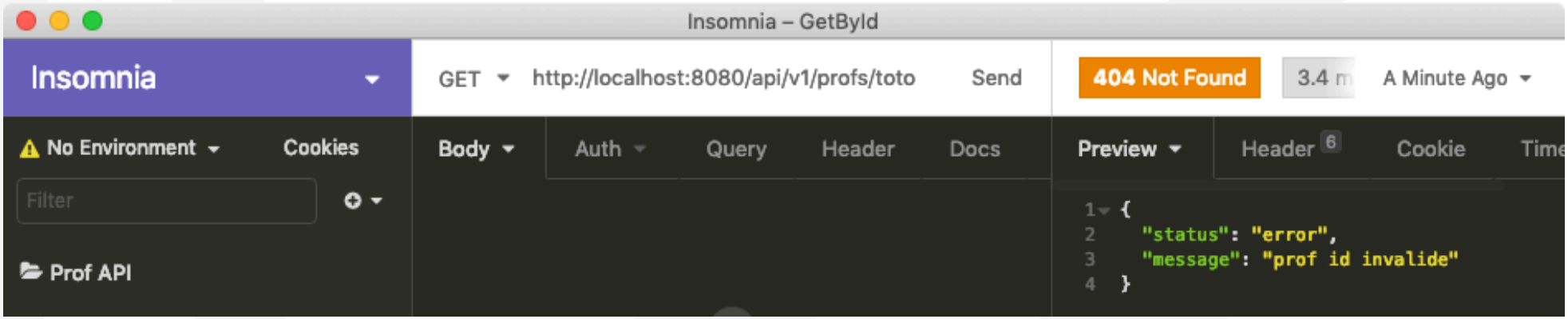
```
11      getById:function(req,res) {
12          let id = req.params.id;
13          let prof = profs.getById(id);
14          if(!prof) {
15              res.status(404).json({
16                  'status':'error',
17                  'message':"prof id invalide"
18              });
19              return;
20          }
21          res.status(200).json({
22              'status':'success',
23              'data': prof
24          });
25      },
```

VÉRIFICATIONS

- Avec Insomnia tester les routes :
  - `http://localhost:8080/api/v1/profs/187` : les données de William Ruchaud doivent apparaître avec le code 200.



  - `http://localhost:8080/api/v1/profs/toto` : on doit voir apparaître notre JSON d'erreur avec le code 404



ROUTE ADD

- Nous allons maintenant faire la route du POST.
- Les informations à enregistrées seront donc envoyées dans le corps de la requête (body). C'est là qu'intervient `body-parser`.
- Toujours pareil, nous allons volontairement rester simple et ne tester que l'existence des 3 données (nom, prénom, bureau) sans en vérifier plus (mais on le devrait).
- Nous aurons 2 codes de réponse possible :
  - 201 : la ressource est créée.
  - 409 : il n'est pas possible de traiter la requête en l'état.

ROUTE GETBYID

- Dans le fichier `/controllers/profsController.js` modifier la fonction `add()`.

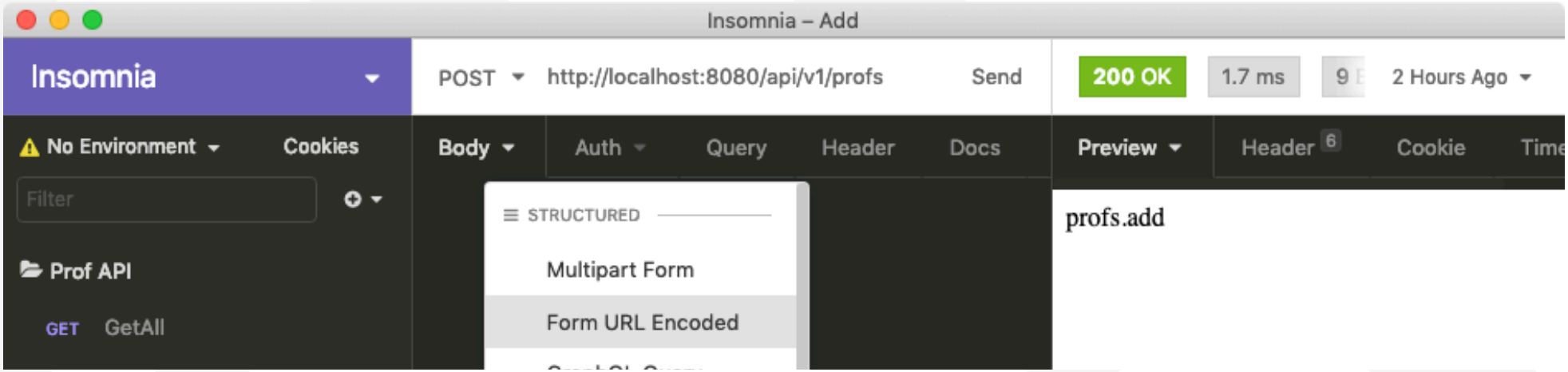
/controllers/profsController.js :

```
27      add:function(req,res) {
28          let { nom, prenom, bureau } = req.body;
29          if(nom == null || prenom == null || bureau == null) {
30              res.status(409).json({
31                  'status':'error',
32                  'message': 'Données incomplètes pour prof'
33              });
34              return;
35          }
36          profs.add(nom,prenom,bureau);
37          res.status(201).json({
38              'status':'success',
39              'message':'Prof ajouté'
40          })
41      },
```

- Noter l'utilisation de la déstructuration ligne 28 pour récupérer en un seul appel les 3 données nécessaires.

VÉRIFICATIONS

- Il nous faut rajouter des données à envoyer dans la requête POST.
- Déplier le menu **Body** en dessous de l'URL et choisir **Form URL encoded**.



- Remplir avec les données suivantes :

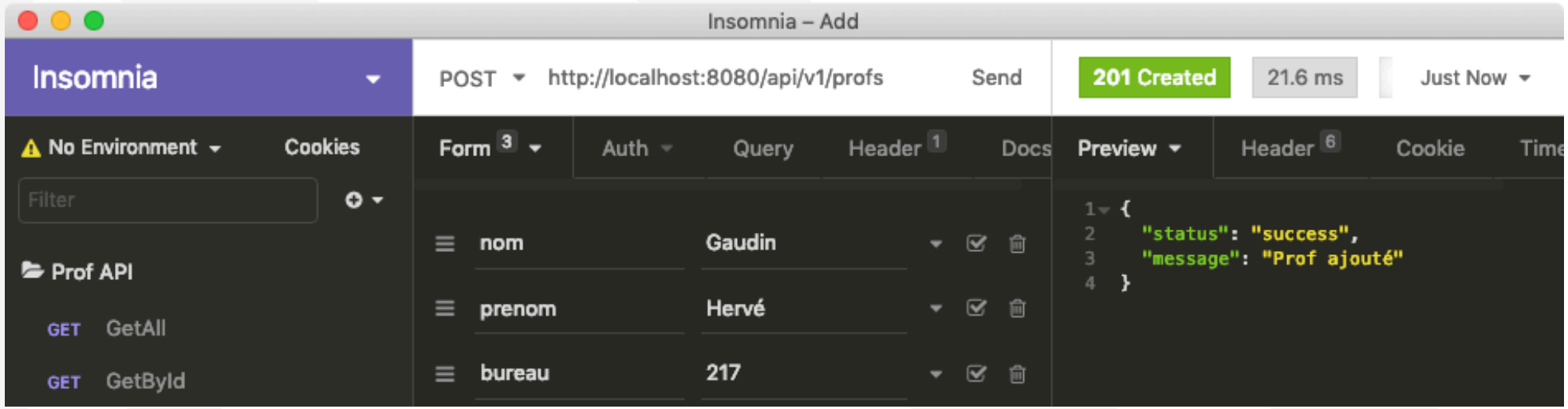
  nom : Gaudin

  prenom : Hervé

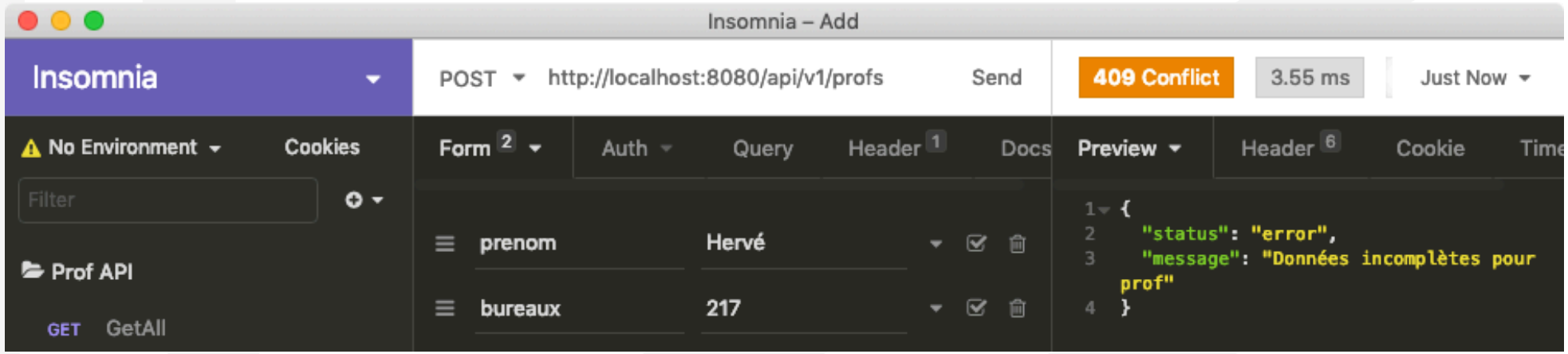
  bureau : 217
- Et envoyer

VÉRIFICATIONS

- La réponse doit avoir le code 201.



- Supprimer un des champs ou ajouter une lettre quelconque au nom du champ, la réponse doit être 409.



ROUTE UPDATE

- Nous allons maintenant faire la route du PUT.
- C'est un mélange du `getById()` et du `add()` :
  - Comme dans `getById()` l'id sera renseigné dans la route.
  - Comme dans `add()` les données à modifiées seront dans le body.
- La simplicité est toujours de mise, il faut fournir les 3 données (nom, prénom et bureau) pour faire une mise à jour.
- En matière de codes de retour possibles :
  - 404 : pas de prof à l'id indiqué
  - 204 : mise à jour effectuée (mais pas de retour du contenu)
  - 409 : les données transmises ne sont pas correctes/complètes

ROUTE UPDATE

- Dans le fichier `/controllers/profsController.js` modifier la fonction `update()`.

/controllers/profsController.js :

```
43      update:function(req,res) {
44          let id = req.params.id;
45
46          let { nom, prenom, bureau } = req.body;
47          if(nom == null || prenom == null || bureau == null) {
48              res.status(409).json({
49                  'status':'error',
50                  'message': 'Données incomplètes pour prof'
51              });
52              return;
53          }
54
55          if(!profs.update(id,nom,prenom,bureau)) {
56              res.status(404).json({
57                  'status':'error',
58                  'message':"prof id invalide"
59              });
60              return;
61          }
62          res.status(204).json({
63              'status':'success',
64              'message':'Prof modifié'
65          })
66      },
```

VÉRIFICATIONS

- Il nous faut faire 3 vérifications :
  - Un cas qui fonctionne (exemple : changer le bureau de quelqu'un qui existe)
  - Un cas qui échoue par ce que l'id ne correspond à rien : réponse 404
  - Un cas avec une donnée manquante : réponse 409

ROUTE DELETE

- Il ne nous reste plus que la route DELETE.
- Cette route va ressembler à celle du `GetById()` puisque l'id du prof à supprimer sera dans la route.
- Au niveau de code de retour, il y aura :
  - 204 : suppression effectuée mais pas de contenu en retour
  - 404 : ressource non trouvée

ROUTE DELETE

- Dans le fichier `/controllers/profsController.js` modifier la fonction `delete()`.

/controllers/profsController.js :

```
69      delete:function(req,res) {
70          let id = req.params.id;
71          if(!profs.deleteById(id)) {
72              res.status(404).json({
73                  'status':'error',
74                  'message':"prof id invalide"
75              });
76              return;
77          }
78          res.status(204).json({
79              'status':'success',
80              'message' : 'prof supprimé'
81          });
82      }
```

VÉRIFICATIONS

- Il nous faut faire 3 vérifications :
  - Un cas qui fonctionne (id 187 par exemple)
  - Un cas où l'id n'existe pas (1 par exemple)
  - Rejouer le cas id 187 qui revient maintenant à l'id qui n'existe pas

AJOUT D'UN NOUVEAU ROUTEUR

- Juste histoire d'éprouver notre architecture, nous allons ajouter un routeur et le contrôleur associé.
- Nous nous limiterons à un retour textuel, il ne s'agit que d'établir la marche à suivre même si elle est relativement simple.
- Commençons par le contrôleur `/controllers/messagesController.js`

/controllers/messagesController.js :

```
1  module.exports = {
2      getAll : function(req,res) {
3          res.status(200).json({
4              status:"success",
5              data:'coucou'
6          })
7      }
8  }
```

AJOUT D'UN NOUVEAU ROUTEUR

- Poursuivons avec le routeur `/routers/messagesRouter.js`

/controllers/messagesController.js :

```
1   const express = require('express');
2   const messagesController = require('../controllers/messagesController');
3
4   function buildRoutes() {
5       let router = express.Router();
6
7       router.route('/messages/').get(messagesController.getAll);
8
9       return router;
10  }
11
12  exports.router = buildRoutes();
```

AJOUT D'UN NOUVEAU ROUTEUR

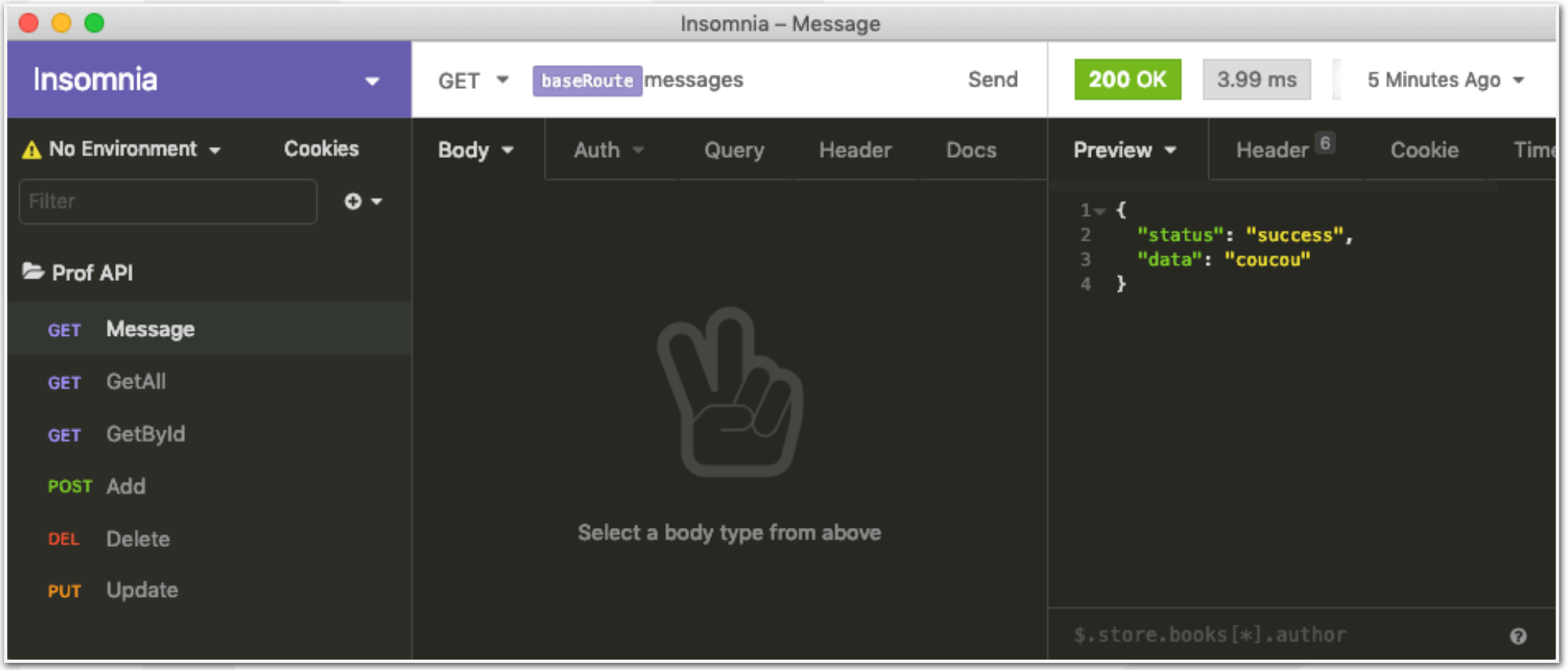
- Puis l'ajout dans `/index.js`

/index.js :

```
1   const express = require('express');
2   const bodyParser = require('body-parser');
3
4   const profsRouter = require('./routers/profsRouter').router;
5   const messagesRouter = require('./routers/messagesRouter').router;
6
7   const app = express();
8   app.use(bodyParser.urlencoded({extended:true}));
9   app.use(bodyParser.json());
10
11  app.use('/api/v1',profsRouter);
12  app.use('/api/v1',messagesRouter);
13
14  app.get('/', (req,res) => {
15      res.send("API REST avec Express");
16  })
17
18  app.listen(8080,()=>{
19      console.log("Ecoute du port 8080");
20  })
21
```

LE TEST

- Il ne reste plus qu'à tester avec Insomnia



- Et de façon évidente, les autres routes doivent continuer à répondre (non régression) !

API REST avec Express
Partie 6

MISE AU POINT

- Lorsque l'on travaille sur une API, on est parfois confronté au problème d'une réponse incorrecte, voir d'une non réponse.
- Comme il n'y a pas de résultat visuel, on peut se poser la question de ce qu'il se passe.
- Bien sûr, on peut multiplier les `console.log()`, mais fatalement, il arrive un moment où cela devient compliqué.
- C'est là qu'il faut se tourner vers un vrai moteur de log qui va stocker l'activité de l'API dans un fichier que l'on pourra regarder, fouiller, filtrer dans tous les sens.
- Nous allons utiliser le middleware Morgan pour le faire.

NOTION DE MIDDLEWARE

- Un middleware est une brique logicielle qui va intervenir dans notre cas en amont des traitements.
- Express fonctionne avec des middleware, et nous en avons déjà un en place : body-parser.
- Il scanne chacune des requêtes et transforme les données transmises en quelque chose de facilement exploitable.
- En fait, dans Express on peut empiler les middleware qui seront donc déclenchés les uns après les autres à chaque requête.

MORGAN

- Le but n'est pas de présenter morgan dans le détail, le temps manque pour ça.
- Nous allons faire un log fichier de toutes les requêtes dans des fichiers qui se trouverons dans un dossier /log.
- Ces fichiers changeront de façon journalière et auront un format pré-défini nommé "combined".

- Installer morgan : `npm install morgan rotating-file-stream`
- Créer le dossier `/log`

MORGAN

- Éditer le fichier `/index.js`

/index.js :

```
1   const express = require('express');
2   const bodyParser = require('body-parser');
3
4   const morgan = require('morgan')
5   const path = require('path')
6   const rfs = require('rotating-file-stream')
7
8   const profsRouter = require('./routers/profsRouter').router;
9   const messagesRouter = require('./routers/messagesRouter').router;
10
11  const app = express();
12  app.use(bodyParser.urlencoded({extended:true}));
13  app.use(bodyParser.json());
14
15  const accessLogStream = rfs.createStream('access.log', {
16      interval: '1d',
17      path: path.join(__dirname, 'log')
18  })
19
20  app.use(morgan('combined', { stream: accessLogStream }))
21
22  app.use('/api/v1',profsRouter);
23  app.use('/api/v1',messagesRouter);
```

MORGAN

- Voici ce qui est stocké

/index.js :

```
1   ::1 - - [03/May/2020:13:27:24 +0000] "GET /api/v1/profs HTTP/1.1" 200 215 "-"
        "insomnia/7.1.1"
2   ::1 - - [03/May/2020:13:30:02 +0000] "GET /api/v1/profss HTTP/1.1" 404 152 "-"
        "insomnia/7.1.1"
3
```

- Il est possible de formater soi-même les données sauvegardées.
- Mais c'est déjà une bonne piste pour savoir si un appel a bien abouti ou non, vers quelle URL et quel en a été le code de retour.

API REST avec Express
Partie 7

DOCUMENTATION

- Lorsqu'on crée une API, c'est rarement pour soi. C'est souvent une des briques d'un développement plus conséquent.
- Comme tout élément de service (couche basse), la documentation est nécessaire.
- Dans le monde des API, il existe une spécification OpenAPI (OAS) et des outils comme Swagger (Swagger Editor, Swagger UI et Swagger Codegen) pour éditer ces spécifications et éventuellement aller jusqu'à une génération automatique de l'API et du squelette d'un client.
- Note au passage : ce n'est pas si simple.
- Quoi qu'il en soit, ces spécifications et ces outils sont pratiques.
- Tout ceci se trouve ici : https://swagger.io

DOCUMENTATION

- La documentation c'est généralement la 5ème voir 6ème ou plus encore roue du carrosse.
- Quand on peut automatiser une certaine remontée d'informations pourquoi s'en priver.
- express-oas-generator permet cela : cataloguer les routes de l'API
- Même si c'est basique, c'est appréciable, surtout vu le coût de mise ne place (3 lignes de code pour démarrer).

- Installer le package : `npm install express-oas-generator`

EXPRESS-OAS-GENERATOR

- Éditer le fichier `/index.js`
- OAS Generator doit être le premier middleware

/index.js (1/2) :

```
1   const express = require('express');
2   const bodyParser = require('body-parser');
3
4   const morgan = require('morgan')
5   const path = require('path')
6   const rfs = require('rotating-file-stream')
7
8   const expressOasGenerator = require('express-oas-generator');
9
10  const profsRouter = require('./routers/profsRouter').router;
11  const messagesRouter = require('./routers/messagesRouter').router;
12
13  const app = express();
14
15  expressOasGenerator.handleResponses(app, {});
16
17  app.use(bodyParser.urlencoded({extended:true}));
18  app.use(bodyParser.json());
19
20  const accessLogStream = rfs.createStream('access.log', {
21      interval: '1d',
22      path: path.join(__dirname, 'log')
23  })
```

EXPRESS-OAS-GENERATOR

- La suite…

/index.js (1/2) :

```
25  app.use(morgan('combined', { stream: accessLogStream }))
26
27  app.use('/api/v1',profsRouter);
28  app.use('/api/v1',messagesRouter);
29
30  app.get('/', (req,res) => {
31      res.send("API REST avec Express");
32  })
33
34  expressOasGenerator.handleRequests();
35
36  app.listen(8080,()=>{
37      console.log("Ecoute du port 8080");
38  })
39
```

- Très important : il faut effectuer au moins une requête sur l'API pour que la génération de la documentation fonctionne.

EXPRESS-OAS-GENERATOR

- Avec un navigateur aller sur http://localhost:8080/api-docs