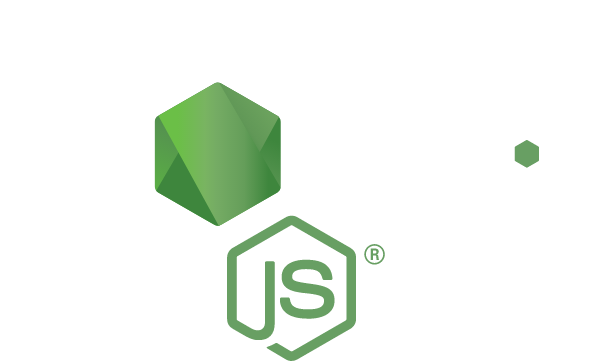
|  |  |
| --- | --- |
| **BTS SIO** | **BTS SIO (SLAM)** |
|  | **Bloc 3** |
|  |  |
| **Routes et paramètres** | |



**NodeJS**

|  |  |  |  |
| --- | --- | --- | --- |
| **date** | **révision** | | |
|  | Timothée Robert | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
|  |  | | |
| **BTS SIO** | |  | **BTS SIO (SLAM)** |
|  | | |  |
| Introduction | | |  |

Nous avons vu la semaine dernière comment installer NodeJS et gérer les différentes versions avec nvm. Nous avons commencé à l’utiliser avec l’IDE PHPStorm.

# Routes Get

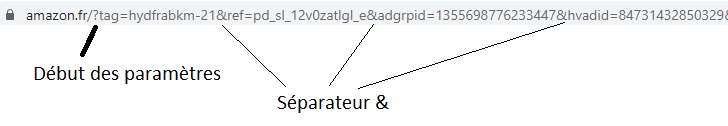
## Introduction et définition

Pour les URL, de manière classique, on peut utiliser des paramètres.

Définitions : *paramètre*

En informatique, un *paramètre* est une [donnée](https://www.techno-science.net/glossaire-definition/Donnee.html) manipulée par une section de code et connue du code appelant cette section.[[1]](#footnote-1)

Exemple : URL de type GET avec des paramètres



Dans cet exemple, l’URL amazon.fr est suivi de paramètres. Il y en a 4 :

* + - Un paramètre nommé tag avec la valeur hydfrabkm-21
    - Un paramètre nommé ref avec la valeur …
    - Un paramètre adgrpid
    - Et le dernier hvadid

Chaque couple paramètre / valeur est suivi d’un séparateur &

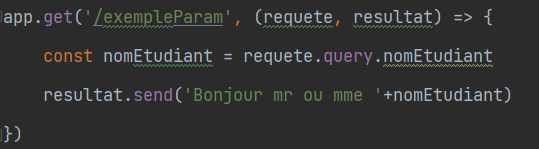
Nous allons voir qu’avez NodeJS nous avons le choix :

1. Soit on peut directement récupérer les paramètres de l’URL

Les paramètres commencent après le ? , sont de la forme nom=valeur et sont séparés par des & (ampersand)

Pour récupérer la valeur d’un paramètre nommé par exemple nomParametre , on va utiliser le paramètre query de la requête express.

Exemple :



Si on demande l’URL <http://localhost:3000/exempleParam?nomEtudiant=Robert> alors il renverra Bonjour mr ou mme Robert

Ici exempleParam est le nom de la route. Le caractère ? marque le début des paramètres de l’URL. nomEtudiant est le nom du premier paramètre et Robert sa valeur.

Pour récupérer la valeur associée au paramètre nomEtudiant on utilise requete.query.nomEtudiant

2. Soit on peut utiliser une fonctionnalité un peu ***différente*** , les routes paramétrées

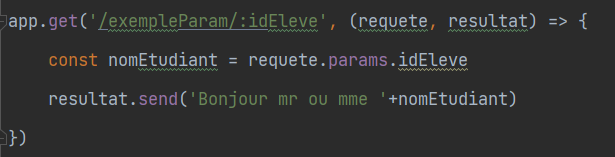
## Route avec un paramètre en NodeJS

### Introduction

Les routes de type GET peuvent prendre des paramètres qui sont automatiquement transmis à Express.

Le paramètre est nommé après le caractère : dans l’URL.

Exemple :



Explication :

La route exempleParam accepte un paramètre, connu sous le nom idEleve.

Utilisation :

L’URL <http://localhost:3000/exempleParam/Robert> affichera « Bonjour mr ou mme Robert » dans le navigateur

Exercice 1 : reproduire l’exemple.

L’exécuter et le tester.

Exercice 2 : ajouter une route avec un paramètre numérique

Choisir un nom de route, avec un paramètre numérique.

Cherche le moyen de valider le paramètre : s’il est numérique entier, afficher : « le nombre saisi est » avec la valeur du paramètre.

Sinon, affichez : « le paramètre » valeur du paramètre « est erroné ».

### Recherche d’une correspondance dans un fichier JS

L’utilité d’un paramètre peut être, dans une API par exemple, de filtrer les données en provenance d’une base de données ou d’un flux / fichier JS.

Nous allons voir la syntaxe et étudier un exemple qui va nous permettre également d’aborder des éléments syntaxiques remarquables du langage JavaScript.

Remarque : vous pouvez être étonnés de ne pas voir de double quote (« ) dans le tableau JavaScript. Les tableaux JSON et JS sont extrêmement proches mais JSON requiert des quotes pour les clefs tandis que JavaScript ne l’impose pas.

#### Préliminaire : passage en module ES et préparation

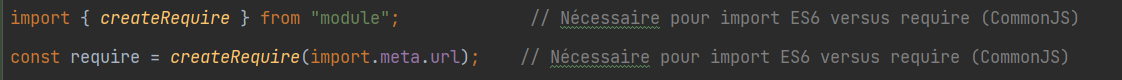
Nous allons dorénavant systématiquement utiliser les modules ES plutôt que la syntaxe CommonJS. C’est hors scope de ce qu’on a à étudier mais sachez que JavaScript a pas mal de variantes et d’évolutions avec des syntaxes légèrement différentes, nous allons choisir ES Module et nous y tenir.

Etape 1 : spécification du type module dans le fichier package.json

Ajouter la spécification dans le fichier package.json



Etape 2 : on ajoute 2 lignes tout en haut de notre programme pour lui dire que même si on utilise la syntaxe Module ES , on peut continuer à utiliser des require comme dans CommonJS (si vous n’y comprenez rien c’est normal …)

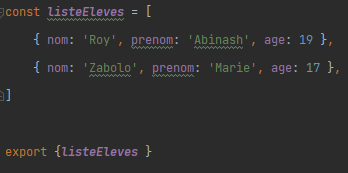


Rmq : les commentaires sont là « pour mémoire »

Etape 3 :

On crée un répertoire sources puis dedans un fichier JS nommé eleves.mjs (l’extension signifie module JavaScript) avec pour contenu :





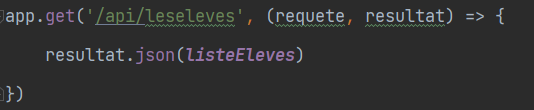
Commentaire : on a un tableau JavaScript nommé listeEleves qu’on exporte.

Etape 4: Import du tableau

Dans le programme principal, avant la définition des routes, après le require path, on importe le tableau JavaScript :



Etape 5 : Ajout d’une route et affichage du tableau



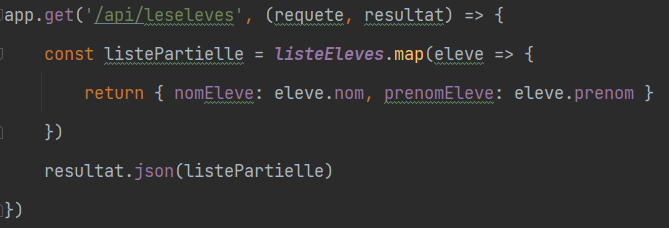
Exercice 3 :

* reproduire l’exemple
* qu’est ce qui est affiché lorsqu’on va sur la nouvelle route ?
* quel est le type de ce qui est affiché lorsqu’on va sur la nouvelle route ? Qu’en déduisez vous ?
* complétez le tableau des élèves en vous synchronisant avec les camarades, pour avoir une liste complète

#### La fonction JavaScript map

Parmi les principales fonctions introduites dans différents langages de développement ces dernières années on trouve ***map*** et ***reduce***.

On modifie la route précédente de la façon suivante.



Exercice 3 :

* reproduire l’exemple
* qu’est ce qui est affiché lorsqu’on va sur la route ?
* qu’est ce qui a changé ?
* trouvez et donnez la documentation de la fonction map
* expliquez ce qu’elle fait

#### Filtre : recherche de correspondance

On peut utiliser la fonction find pour chercher une correspondance dans un tableau JavaScript.

Cf la documentation de la fonction find : <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/find>

La méthode find d’une instance de Array renvoie le 1er élément du tableau (Array) fourni qui satisfait la fonction de test. Si aucune valeur du tableau ne satisfait la fonction de test, alors la valeur undefined est renvoyée.

Syntaxe :

Array.find(fonction)

Leplus souvent on affectera le résultat de cette méthode à une variable ou une constante.

Où Array est une instance d’un Array, donc un tableau Javascript et fonction une fonction booléenne qui renvoie true ou false.

La beauté du langage fonctionnel JavaScript est que la fonction peut être quelque chose du genre :

element => element == parametre où element va être successivement chaque élément du tableau (Array) et element == parametre la condition qui renvoie vrai ou faux.

Exercice 4 : recherche d’un élève par nom

Spécifications : on souhaite créer une nouvelle route de recherche d’un élève. On prendre en paramètre un nom d’élève. Si le nom d’élève spécifié dans l’URL correspond à un élève dans le tableau JavaScript (listeEleves) alors on renvoie l’élève correspondant dans le navigateur

**Hints** :

* 1. Pour la création de la route, on va reprendre ce qu’on a fait dans l’exercice 1, donc on choisit un « bon » nom de route et de paramètre, par exemple nomEleve et on n’oublie pas le : devant ou alors on fait l’hypothèse que la route va être appelée avec un paramètre d’URL de type ?nomEleve= …
  2. On se dit qu’on va utiliser le find qu’on a vu. Quelle est la syntaxe à utiliser ?
     + Le tableau auquel il s’applique, on l’a vu dans l’exercice précédent, c’est nomEleves
     + La fonction dans le find est un peu plus difficile à trouver : on va s’inspirer de la syntaxe fournie plus haut, en appelant par exemple eleve chaque element du tableau
     + On remarquera alors que element est un objet JavaScript pour chaque itération avec 3 propriétés, nom, prenom et age
     + On va alors comparer le nom (donc eleve.nom ) avec le paramètre de route ou d’URL (cf syntaxe Exercice 1)
     + Pour l’affichage on fera attention à ce que le resultat est lui-même un tableau JavaScript (avec le 1er élément trouvé).

Question : qu’obtenez vous si vous aucun élève ne correspond à la recherche (si le nom recherché ne correspond pas à un nom d’élève ?).

Spécifications :

1. On souhaite améliorer la recherche en recherchant juste sur une partie du nom de l’élève
2. On souhaite améliorer le résultat précédent en renvoyant une erreur standard si aucun élève n’est trouvé

**Hints** :

Pour la 1ère question il y a de nombreuses façons de faire : on pourra par exemple regarder la methode filter de la classe Array : <https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array/filter>

Et également la méthode includes qui s’applique à un Array.

Par exemple, on pourra regarder pour chaque élément du tableau listeEleves si le nom contient le paramètre de la route (ou de l’URL selon la méthode choisie).

Pour la deuxième question, on pourra étudier la documentation et utiliser la méthode status de la réponse express : <https://expressjs.com/en/4x/api.html#res.status>

Il faut juste trouver le bon code qui correspond à une ressource non trouvée : <https://developer.mozilla.org/fr/docs/Web/HTTP/Status> et personnaliser le message.

On se souviendra que la force de la programmation fonctionnelle est de pouvoir enchainer les fonctions, en l’occurrence ici status pour le code puis send pour un message et le retour au navigateur.

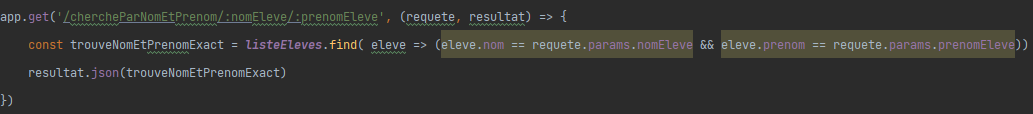
## route avec plusieurs paramètres en NodeJS

On sait qu’on peut avoir plusieurs paramètres dans une URL puisque chaque paramètre est séparé par le ampersand &

On peut également avoir plusieurs paramètres de route, avec la syntaxe alternative / :param1/ :param2

Exercice 5 : recherche d’un élève par nom et prénom

* Soit la route suivante



Expliquez ce qu’elle fait.

Donnez l’URL qui permet d’afficher l’élève Roy Abinash

* Ecrivez une autre route, qui prend également 2 paramètres mais sous forme de paramètres d’URL (séparés par des & donc) et qui donne le même résultat
* Si la correspondance n’est pas trouvée, renvoyer une erreur 404
* Quelle modification faudrait-il faire si on veut renvoyer les élèves du tableau dont le nom est le nom en paramètre ou bien le prénom est le prénom en paramètre ?

# Routes Post

## Introduction et définition

Définition complète : <https://developer.mozilla.org/fr/docs/Web/HTTP/Methods/POST>

La méthode ou verbe http POST envoie des données au serveur, généralement via un formulaire HTML.

Nous allons voir comment traiter des données de formulaire avec NodeJS et Express.

Dans un premier temps nous allons nous concentrer sur la récupération des données au niveau de la couche applicative NodeJS.

Ensuite nous aborderons la persistance de ces données dans une Base de données.

## Utilisation de POSTMAN

Postman est un outil extrêmement populaire chez les développeurs pour tester les applications Web et les API.

Avec Curl, outil en ligne de commande, il est incontournable pour le développement d’API et d’applications Web.

Le Soft est sur le réseau et vous pouvez l’installer sur votre poste s’il n’y est pas déjà et vous enregistrer.

## Pré requis NodeJS

Pour récupérer des données d’un formulaire dans une route express de type POST, NodeJS a besoin d’un package, body-parser

Etape 1 : installation du package



Etape 2 : Utilisation dans l’application

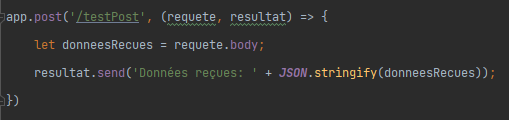
Dans notre fichier principal, au début, là où on déclare les packages :



## Création d’une route POST

Dans un premier temps nous allons créer une route POST simplement en lui passant des données « à la volée » via la route en utilisant l’outil POSTMAN

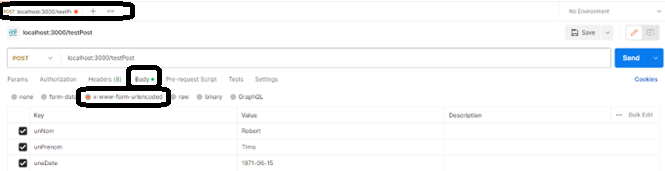
Exemple d’une route POST :



## Test avec Postman

Etape 1 : on lance POSTMAN

Ci-dessous la copie d’écran de la requête POSTMAN avec les éléments entourés



Détaillons les étapes pour arriver à ce screen

Etape 2 : on créer une nouvelle requête de type POST

C’est le bouton + à droite de l’URL

Etape 3 : URL

On renseigne à droite du POST l’URL de la route : dans notre exemple il s’agit de localhost :3000/testPost

Etape  4: Allez sur l’onglet BODY

Etape  5: Cochez le radio button x-www-form-urlencoded

Etape 6 : Ajout de paramètres avec des valeurs

Ajoutez quelques paramètres de votre choix avec des valeurs de votre choix

Etape 7 : Testez en soumettant la requête

## Traitement de formulaire

### Contexte

Habituellement nous avons vu que les routes POST traitent des données de formulaire.

Nous allons voir un exemple.

Le contexte de notre exemple est simplement un formulaire de login qui permet de se connecter à une application.

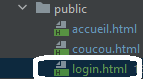
Il y a 2 éléments :

* La page HTML qui contient le formulaire et le lien vers la route de validation
* La route qui permet d’accéder au formulaire de login !
* La route POST qui effectue la validation

### Formulaire HTML

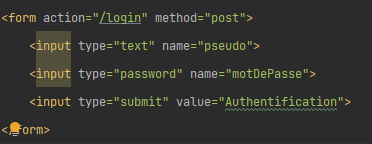
Ci-dessous un exemple simplissime de page HTML avec login et mot de passe.

Etape 1 : créer une page login.html dans un répertoire public



Etape 2 :

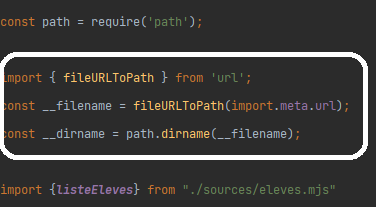
A l’intérieur de la balise body, le formulaire :



On remarque que le nom de la route est login, avec la méthode POST

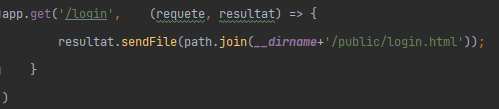
Etape 3 : compatibilité modules / CommonJS

Le mot clef \_\_dirname n’existe pas dans ES Module. Pour répliquer le même comportement nous devons ajouter les 3 lignes suivantes.



### Route vers le formulaire HTML

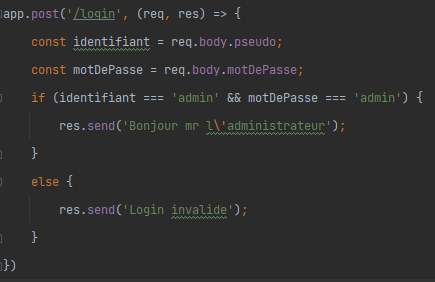
On peut simplement définir une route de type get vers le formulaire de l’étape précédente.



On remarque que le nom de la route est également login mais avec la méthode GET cette fois ci, ce qui permet de distinguer parfaitement cette route de la route précédente.

### Traitement des données du formulaire

Il reste à définir une route de type POST qui permet de traiter les données du formulaire.



Exercice 6 : test du formulaire

* Tester le formulaire : quelle URL permet d’y accéder ?
* Commentez ce que fait la route de traitement & validation du formulaire
* Ajouter dans le formulaire deux autres champs texte nommé nom et prenom
* Maintenant, si l’identifiant est autre que admin, affichez dan le navigateur

Le nom, le prénom, l’identifiant

1. Source : <https://www.techno-science.net/glossaire-definition/Parametre.html>. On distinguera par la suite les paramètres

   d’entrée et les paramètres de sortie [↑](#footnote-ref-1)