

Prérequis :

Installation d'une Debian propre en VM

Installation de gcc, g++, make

Compilation de NodeJs depuis les sources (make && make install)

Installation de mysql-server (login root, password root)

Installation de Strongloop (npm install strongloop)

Installation du connecteur MySQL (npm install loopback-connector-mysql)

Déclenchement du timer : 19h00

Création d'une application 'loopback' : slc loopback

```
thibaut@debian:~/Desktop/StrongLoop$ slc loopback

  --(o)--  |  Let's create a LoopBack
  ( _U_ )   |  application!
 /  A  \
 |  ~  |
 |  °  |  Y
 |  |  |

? What's the name of your application? BDE
? Enter name of the directory to contain the project: BDE
  create BDE/
    info change the working directory to BDE

Generating .yo-rc.json

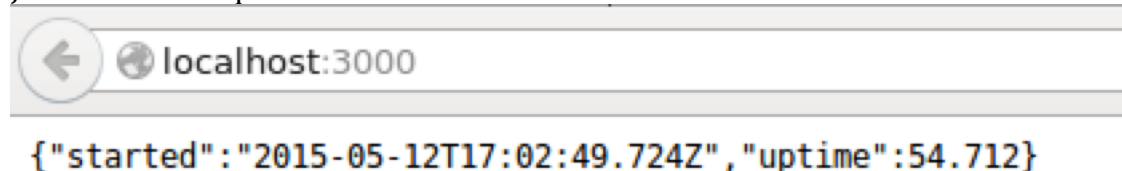
I'm all done. Running npm install for you to install the required dependencies.
```

Je vais dans le dossier crée : cd BDE

Juste pour le fun on lance déjà le serveur

```
thibaut@debian:~/Desktop/StrongLoop/BDE$ slc run
INFO strong-agent v1.5.1 profiling app 'BDE' pid '4188'
INFO strong-agent[4188] started profiling agent
INFO supervisor reporting metrics to `internal:`
supervisor running without clustering (unsupervised)
INFO strong-agent not profiling, agent metrics requires a valid license.
Please contact sales@strongloop.com for assistance.
Browse your REST API at http://0.0.0.0:3000/explorer
Web server listening at: http://0.0.0.0:3000/
```

Je vais aux URL qu'il me donne :



```
{"started": "2015-05-12T17:02:49.724Z", "uptime": 54.712}
```

The screenshot shows the StrongLoop API Explorer interface for the **Users** endpoint. The URL bar indicates `localhost:3000/explorer/`. The interface includes a header with the StrongLoop logo, the title "StrongLoop API Explorer", and a "Token Not Set" status with an "accessToken" input field and a "Set Access Token" button. Below the header, the "Users" section is displayed with a list of 14 methods. Each method is represented by a colored bar with its HTTP method, path, and description.

Method	Path	Description
POST	/Users	Create a new instance of the model and persist it into the data source.
PUT	/Users	Update an existing model instance or insert a new one into the data source.
GET	/Users	Find all instances of the model matched by filter from the data source.
HEAD	/Users/{id}	Check whether a model instance exists in the data source.
GET	/Users/{id}	Find a model instance by id from the data source.
DELETE	/Users/{id}	Delete a model instance by id from the data source.
PUT	/Users/{id}	Update attributes for a model instance and persist it into the data source.
GET	/Users/{id}/accessTokens	Queries accessTokens of User.
POST	/Users/{id}/accessTokens	Creates a new instance in accessTokens of this model.
DELETE	/Users/{id}/accessTokens	Deletes all accessTokens of this model.
GET	/Users/{id}/accessTokens/{fk}	Find a related item by id for accessTokens.
DELETE	/Users/{id}/accessTokens/{fk}	Delete a related item by id for accessTokens.
PUT	/Users/{id}/accessTokens/{fk}	Update a related item by id for accessTokens.
GET	/Users/{id}/accessTokens/count	Counts accessTokens of User.

Encore 9 autres méthodes en dessous que je ne détaille pas

Le détail d'une seule méthode (POST /Users)

The screenshot shows the StrongLoop API Explorer interface for the **POST /Users** method. The URL bar indicates `localhost:3000/explorer/#!/Users/create`. The interface includes a header with the StrongLoop logo, the title "StrongLoop API Explorer", and a "Token Not Set" status with an "accessToken" input field and a "Set Access Token" button. Below the header, the "POST /Users" section is displayed. It shows a sample response in JSON format:

```
{
  "status": "",
  "created": "",
  "lastUpdated": ""
}
```

Below the response, there is a "Response Content Type" dropdown set to `application/json`. Under the "Parameters" section, there is a table with columns: Parameter, Value, Description, Parameter Type, and Data Type. The table has one row for the `data` parameter, which is a body parameter of type `Model`. The "Data Type" column shows the "Model Schema" for the `Model` type, which is a JSON object with the following structure:

```
{
  "realm": "",
  "username": "",
  "credentials": "object",
  "challenges": "object",
  "email": "",
  "emailVerified": false,
  "verificationToken": "",
  "status": "",
  "created": "",
  "lastUpdated": ""
}
```

Below the parameters, there is a "Response Messages" section with a table showing the HTTP status code and reason. The table has one row for the `200` status code, with the reason "Request was successful".

Là il est 19h09, j'ai déjà la gestion des utilisateurs.

Je crée un nouvel objet : slc loopback :model

```
thibaut@debian:~/Desktop/StrongLoop/BDE$ slc loopback:model
? Enter the model name: Product
? Select the data-source to attach Product to: db (memory)
? Select model's base class: PersistedModel
? Expose Product via the REST API? Yes
? Custom plural form (used to build REST URL): Products
Let's add some Product properties now.

Enter an empty property name when done.
? Property name: price
  invoke    loopback:property
? Property type: number
? Required? Yes

Let's add another Product property.
Enter an empty property name when done.
? Property name: description
  invoke    loopback:property
? Property type: string
? Required? Yes

Let's add another Product property.
Enter an empty property name when done.
? Property name:
thibaut@debian:~/Desktop/StrongLoop/BDE$
```

La création, dans l'ordre :

- nom du modèle (objet)
- où le sauvegarder ? (par défaut j'ai qu'une base de données bidon en RAM)
- de quelle classe je veux hériter (quasiment toujours PersistedModel)
- est-ce que je veux générer toutes les méthodes pour Product
- le nom du modèle au pluriel

Les propriétés, dans l'ordre :

- nom de la propriété
- type de la propriété
- obligatoire ou non (pour la validation)

Je relance mon serveur et je vais à la page de l'explorer :

Products Show/Hide | List Operations | Expand Operations | Raw

POST	/Products	Create a new instance of the model and persist it into the data source.
PUT	/Products	Update an existing model instance or insert a new one into the data source.
GET	/Products	Find all instances of the model matched by filter from the data source.
HEAD	/Products/{id}	Check whether a model instance exists in the data source.
GET	/Products/{id}	Find a model instance by id from the data source.
DELETE	/Products/{id}	Delete a model instance by id from the data source.
PUT	/Products/{id}	Update attributes for a model instance and persist it into the data source.
GET	/Products/{id}/exists	Check whether a model instance exists in the data source.
GET	/Products/count	Count instances of the model matched by where from the data source.
GET	/Products/findOne	Find first instance of the model matched by filter from the data source.
POST	/Products/update	Update instances of the model matched by where from the data source.

Users Show/Hide | List Operations | Expand Operations | Raw

Ah j'ai oublié de mettre un attribut 'quantity' :

```
thibaut@debian:~/Desktop/StrongLoop/BDES$ slc loopback:property
? Select the model: Product
? Enter the property name: quantity
? Property type: number
? Required? Yes
thibaut@debian:~/Desktop/StrongLoop/BDES$
```

Là, vous l'aurez compris (c'est chiant les screenshot), si je relance le serveur et que je vais sur l'explorer, toutes les méthodes seront régénérées pour prendre en compte la nouvelle description de l'objet.

Je déclare une nouvelle source de données MySQL avec comme nom 'mysql'

```
thibaut@debian:~/Desktop/StrongLoop/BDES$ slc loopback:datasource
? Enter the data-source name: mysql
? Select the connector for mysql: MySQL (supported by StrongLoop)
thibaut@debian:~/Desktop/StrongLoop/BDES$
```

Je vais dire à l'application qu'elle doit enregistrer le modèle 'Product' dans la datasource 'mysql'. Pour ça j'édite le fichier server/model-config.json avec nano (oui avec nano, tellement c'est easy).

```
GNU nano 2.2.6      File: server/model-config.json      Modified
},
"AccessToken": {
  "dataSource": "db",
  "public": false
},
"ACL": {
  "dataSource": "db",
  "public": false
},
"RoleMapping": {
  "dataSource": "db",
  "public": false
},
"Role": {
  "dataSource": "db",
  "public": false
},
"Product": {
  "dataSource": "mysql",
  "public": true
}
```

J'ai oublié les screenshot mais bon, je suis allé dans mysql pour créer une base BDE avec une table Product. Dans Product j'ai mis les champs et les types définis plus haut en ligne de commande.

Je vais jouer dans l'explorer et j'entre des données dans la méthode POST /Product. Voilà le retour :

Request URL

http://localhost:3000/api/Products

Response Body

```
{
  "price": 0,
  "description": "lol",
  "quantity": 0,
  "id": 0
}
```

Response Code

200

Response Headers

```
{
  "X-Powered-By": "Express",
  "Vary": "Origin, Accept-Encoding",
  "Access-Control-Allow-Credentials": "true",
  "Content-Type": "application/json; charset=utf-8",
  "Content-Length": "51",
  "Etag": "W/\"33-1166c897\"",
  "Date": "Tue, 12 May 2015 17:33:36 GMT",
  "Connection": "keep-alive"
}
```

Confirmation mysql :

```
mysql> select * from Product;
+-----+-----+-----+-----+
| price | description | quantity | id |
+-----+-----+-----+-----+
|      0 | lol          |          0 | 0 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql> █
```

Voilà j'ai une API fonctionnelle et auto documentée. J'ai un peu trainé à vrai dire je n'avais pas encore bossé avec le connecteur MySQL donc j'ai dû consulter la doc de Strongloop et je rédige ce papier en même temps que je fais les commandes.

Autrement ce que je peux dire c'est que :

- je parle de générateur, c'est pour simplifier. Je n'ai pas de code sous les yeux, tout se passe dans le cœur de Strongloop, tout est dynamique.
- J'ai créé mes objets et mes tables à la main mais j'ai cru voir dans la documentation qu'ils proposaient un module de 'discovery' qui crée les objets à partir de mysql directement
- Strongloop en l'état me permet de générer (vraiment générer cette fois) des ressources pour AngularJS. Pour faire court, j'aurais directement côté client un objet Product avec les méthodes qui vont bien ainsi que toute la documentation des méthodes créées.
- L'explorer de Strongloop est une documentation amplement suffisante et dynamiquement générée donc pas besoin de le maintenir.

Une dernière chose ... je viens de terminer et il est 19h43.

Merci aux lecteurs !