Redis

1. Qu’est-ce que Redis ?
2. Explication des types présent sur redis
3. Comment utiliser redis sur laravel
4. Qu’est-ce que Redis ?

Redis est une base de donnée **en-mémoire** (in-memory, peut être persistable) écrite en langage C. Il s’agit donc d’un logiciel qu’on utilise avec des lignes de commandes (heureusement, Laravel le fait à notre place et on utilise juste des méthodes php).

Chaque commande qu’on envoie à redis renvoie une réponse.  
  
**Keys**

Le principe de keys ressemble énormément à celui de json par exemple. On a un key et une value associée.  
  
Un key peut être n’importe quoi (du texte, une image jpeg, du binaire..) qui peut avoir une grandeur max de 512 mb.

Exemple :  
"name" ou \xac\xed\x00\x05t\x00\tvc etc.. source : <https://redis.io/topics/data-types-intro#redis-keys>

* **Il est conseillé d’utiliser des keys comme "user:459:name" ou "user:459:password" ou**

**"user:459:mot-de-passe" ou**

**"user:459:mot.de.passe"**

1. Explication des types présent sur redis

Avant toute chose les commandes redis sont formées  de keys pour lesquelles on a des values associées

Command Key Value

**SET** name John

Réponse de redis : "OK"

**GET** name

Réponse de redis : "John"

Command Key Value [key/values … ]

**MSET** name :1 John name :2 Gérard name :3 Jean-Neymarre

Réponse de redis : "OK"

Command Key [key… ]

**MGET** name :1 name :2 name :3 name:4

Réponse de redis :

1. "John"
2. "Gérard"
3. "Jean-Neymarre"
4. nil 🡪 null (name :4 n’existe pas)

Maintenant passons aux types de redis à proprement parler :

* **Strings**
* **List**
* **Sets**
* **Sorted sets**
* **Hashes**
  1. **Strings** (<https://redis.io/commands#string>)

Les strings sont le moyen le plus simple de stocker des valeurs avec redis. Il suffit d’utiliser l’une des commandes (sur le lien ci-dessus) pour stocker des variables.

Exemple (incrémentation d’une variable) :

Command Key Value

**SET** counter 1

Réponse redis : "OK"

Command Key

**INCR** counter

Réponse redis : (integer) 2

Exemple (suppression de key) :

Command Key

**DEL** counter

Réponse redis : (integer) 1 🡪 Renvoie le nombre de key affectées, ici 1 car il n’y a que counter qui a été supprimé

* 1. **List** (<https://redis.io/commands#list>)

Les lists sont un moyen simple d’avoir une liste d’éléments qui se suivent. Il est simple d’ajouter un élément au tout début de la liste ou à la fin. Pour cela un utilise les commandes :

Exemple (créer une liste mylist et y rajouter des données à droite ou à gauche et afficher ce qu’elle contient)

Command Key Value [Value...]

**RPUSH** mylist 1 2 3 4 5 🡪 rajoute 1 2 3 4 5 à droite de mylist

Réponse redis : (integer) 5 🡪 mylist a été effecté par 5 éléments

Command Key Value [Value...]

**LPUSH** mylist 0 🡪 rajoute 0 à gauche de mylist

Réponse redis : (integer) 1 🡪 mylist a été effecté par 1 élément

Command Key Value [Value...]

**LRANGE** mylist 0 -1 🡪 Retourner les valeurs de mylist allant de 0 à -1 (-1 = dernier, -2 = avant-dernier etc..)

Réponse redis :

1. "0"
2. "1"
3. "2"
4. "3"
5. "4"
6. "5"
   1. **Sets** (<https://redis.io/commands#set>)
   2. **Sorted sets** (<https://redis.io/commands#sorted_set>)
   3. **Hashes** (<https://redis.io/commands#hash>)
7. Comment utiliser redis sur laravel (<https://laravel.com/docs/5.7/redis>)

Après avoir installé redis comme le demande Laravel sur le lien ci-dessus on peut utiliser redis sur laravel.

Pour cela on peut appeler n’importe quelle commande de Redis grâce à Laravel en l’invoquant comme si on invoquait une méthode :

Redis::set('name:1', 'John');

Redis::get('name:1'); 🡪 Retournera John

$id = 1 ;

Redis::get('name:' . $id); 🡪 Retournera John

**Structure de données**



De manière générale, pour accéder à l’attribut d’une entité on utilise le formatage suivant :

"nomDeLaTable.attribut**:**NomDeLaTable.attribut={{ $val }}"

Le nom de la table (en camel case) :

Dans notre cas on pourra avoir ici :

**todos**, **tasks** ou **users**

Chaque attribut lié à la table.  
Si la table est Todos

On peut accéder à son attribut title par

"todos.title"

On pourrait traduire oralement les deux points par un "WHERE" ou "FOR"

Par exemple :

"todos.title**:**todos.id=1"

Pourrait se traduire oralement par : **Get todos.title WHERE todos.id is equal 1**

On pourrait ainsi récupérer l’id de l’utilisateur ayant le mail : gille.ejone @cpnv.ch

"users.id**:**users.mail=gille.ejone @cpnv.ch"

**Récupérer toutes les valeurs**

Mais si on souhaite récupérer toutes les valeurs de l’user il serait long de demander tous les attributs les uns après les autres.  
On peut alors utiliser le raccourci ci-dessous pour récupérer toutes les valeurs de users avec le mail gille.ejone… (notez il n’y a pas de .id après users)

"users**:**users.mail=gille.ejone @cpnv.ch"

Cela pourrait se traduire oralement par : **Get all users infos WHERE users.mail is equal gille.ejone** **@cpnv.ch**

Pour se représenter le tout go voir la page : <https://jsoneditoronline.org/?id=73687b6d22844733a245f7e5237b66b6>

Ou bien :

{

"users:users.mail=gille.ejone@cpnv.ch": {

"id": 1,

"name": "Gille",

"surname": "Éjone",

"mail": "gille.ejone@cpnv.ch",

"password": "e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4"

},

"todos:users.id=1" : [1,2],

"todos.title:todos.id=1": "Finir le projet XML1",

"todos.date:todos.id=1": "2019-01-25",

"todos.title:todos.id=2": "Finir le projet PRW2",

"todos.date:todos.id=2": "2019-01-25",

"tasks:todos.id=1" : [

"Comprendre Redis",

"Faire un document sur Redis",

"Passer du temps à créer une structure de données",

"Faire comprendre tout ceci aux autres membres du groupe",

"Finir le site et la liaison avec les données",

"Rendre le projet le dimanche avec la semaine com"

],

"tasks:todos.id=2" : [

"Pleurer",

"Pleurer",

"Pleurer",

"Essuyer ses larmes",

"Voilà, il faut rendre le projet !"

],

"count(users)": 1,

"count(todos)": 2

}

1. Scénario affichage des todos

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: |  | | |
| Use Case Name: | Se connecter | | |
| Created By: |  | Last Updated By: |  |
| Date Created: |  | Date Last Updated: |  |

|  |  |
| --- | --- |
| Actor: | Client |
| Description: |  |
| Preconditions: | Arriver sur la page d’accueil |
| Postconditions: |  |
| Priority: |  |
| Frequency of Use: |  |
| Normal Course of Events: | 1. L’utilisateur arrive sur la page d’accueil (qui renvoie en fait vers /login) 2. Le système renvoie l’utilisateur vers la page d’accueil car il est déjà connecté 3. Le système affiche les todos de l’utilisateur sur la page d’accueil |
| Alternative Courses: | 2a. L’utilisateur n’est pas connecté, le système le renvoie vers la page /login (connexion requises) |
| Exceptions: |  |
| Includes: | Authentification |
| Special Requirements: |  |
| Assumptions: |  |
| Notes and Issues: |  |

1. Scénario affichage des tâches d’un todo

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: |  | | |
| Use Case Name: | Affichage des tâches d’un todo | | |
| Created By: |  | Last Updated By: |  |
| Date Created: |  | Date Last Updated: |  |

|  |  |
| --- | --- |
| Actor: | Client |
| Description: |  |
| Preconditions: | Authentification |
| Postconditions: |  |
| Priority: |  |
| Frequency of Use: |  |
| Normal Course of Events: | 1. L’utilisateur clique sur un des todos dans sa liste 2. Le système renvoie l’utilisateur vers /todos/{idTodo} 3. Le système affiche toutes les tâches à faire pour {idTodo} avec une case à cocher pour signaler qu’une tâche est finie et des boutons : « Enregistrer » et « + » (pour rajouter une tâche dans la todo). |
| Alternative Courses: |  |
| Exceptions: |  |
| Includes: | Authentification |
| Special Requirements: |  |
| Assumptions: |  |
| Notes and Issues: |  |

1. Scénario mettre une tâche d’un todo en « terminée »

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: |  | | |
| Use Case Name: | Mettre une tâche d’un todo en « terminée » | | |
| Created By: |  | Last Updated By: |  |
| Date Created: |  | Date Last Updated: |  |

|  |  |
| --- | --- |
| Actor: | Client |
| Description: |  |
| Preconditions: | 1. Authentification, Avoir cliqué sur un todo depuis la page d’accueil (Scénario : affichage des tâches d’un todo) |
| Postconditions: |  |
| Priority: |  |
| Frequency of Use: |  |
| Normal Course of Events: | 1. L’utilisateur clique sur la checkbox de la tâche 2. Il appuie sur « Enregistrer » 3. Le système enregistre la todo comme étant terminée |
| Alternative Courses: |  |
| Exceptions: |  |
| Includes: | Authentification |
| Special Requirements: |  |
| Assumptions: |  |
| Notes and Issues: |  |

1. Scénario ajouter une tâche dans un todo

|  |  |  |  |
| --- | --- | --- | --- |
| Use Case ID: |  | | |
| Use Case Name: | Ajouter une tâche dans un todo | | |
| Created By: |  | Last Updated By: |  |
| Date Created: |  | Date Last Updated: |  |

|  |  |
| --- | --- |
| Actor: | Client |
| Description: |  |
| Preconditions: | 1. Authentification, Avoir cliqué sur un todo depuis la page d’accueil (Scénario : affichage des tâches d’un todo) |
| Postconditions: |  |
| Priority: |  |
| Frequency of Use: |  |
| Normal Course of Events: | 1. L’utilisateur clique sur le bouton « + » sur la page 2. Le système va chercher la vue correspondante à une tâche (en jQuery) et l’affichera en-dessous des autres de manière dynamique |
| Alternative Courses: |  |
| Exceptions: |  |
| Includes: | Authentification |
| Special Requirements: |  |
| Assumptions: |  |
| Notes and Issues: |  |