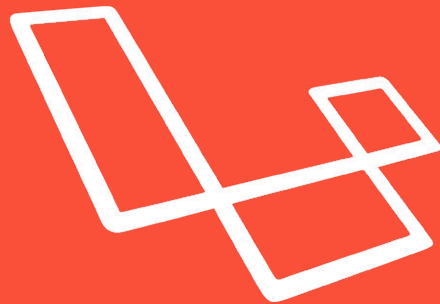


Laravel & MongoDB

Meetup Neoxia Paris - 09/11/2016



Sommaire

1. MongoDB
2. Pourquoi MongoDB ?
3. MongoDB & Laravel
4. Pourquoi... pas MongoDB ?

- NoSQL
- Orienté Documents (format BSON)
- Pas de schéma
- UUID (Universal Unique Identifier)
- Requêtage en Javascript (Stack MEAN)
- Full-Featured (recherche géographique, full-text...)
- Jargon :

SQL	MongoDB
Table	Collection
Ligne	Document
Colonne	-

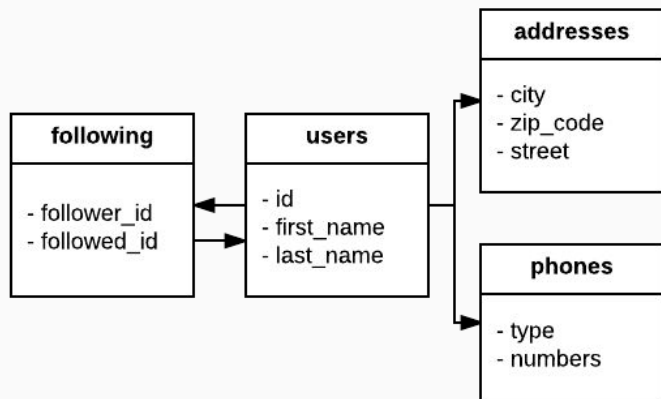
AGILITÉ DES DÉVELOPPEMENTS

- Pas de schéma donc moins de migrations
- Stockage de données Json telles quelles (parfait pour une API REST)
- Coexistence de plusieurs modèles de données
- UUID permet de diviser ou de réunir des collections

MODÉLISER DES DONNÉES COMPLEXES

Évite la création de nombreuses table pour modéliser un objet

SQL



MongoDB

```
{
  _id: "542537f26c16f53747a4cab2",
  first_name: "John",
  last_name: "Doe",
  email: "john.doe@gmail.com",
  addresses: [{
    city: "Paris",
    zip_code: "75008",
    street: "33 avenue des Champs Élysées"
  }],
  phones: [{
    type: "Work",
    number: "01 45 98 56 23"
  }],
  followeds: ["7f26c1542536fab253747a4c", "26cf6f53747a4cab21542537"]
}
```

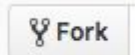
MODÉLISER DES DONNÉES COMPLEXES

Permet de modéliser facilement des types d'objets variables

```
{
  _id: "542537f26c16f53747a4cab2",
  name: "Quiz 1",
  type: "quiz_vrai_faux",
  questions: [
    "Laravel est plus efficace que Symfony ?",
    "PHP est plus pratique que Ruby ?"
  ]
}
```

```
{
  _id: "542537f26c16f53747a4cab2",
  name: "Quiz 1",
  type: "quiz_qcm",
  questions: [
    {
      statement: "Quel est le meilleur framework ?",
      answers: [
        1: "Symfony",
        2: "Zend",
        3: "Laravel"
      ]
    }
  ]
}
```

LIBRAIRIES

- php-mongo
 - jenssegers/laravel-mongodb  Unstar 2,123  Fork 523
 - Héritage des classes de Laravel
 - Très simple à utiliser
- Attention au packaging de php-mongo sur certaines distribution
- Package stable mais imparfait (petite communauté)



ROBUSTESSE DES DONNÉES

- Pas de schéma = données fragiles
- L'application est responsable de la robustesse des données à la place de la BDD

→ Solution : validation des données au niveau de la couche modèle

```
public static function boot()  
{  
    parent::boot();  
    static::saving(function ($object) {  
        $object->validate();  
    });  
}
```



ROBUSTESSE DES DONNÉES

- Pas de schéma = données fragiles
- L'application est responsable de la robustesse des données à la place de la BDD

→ Solution : delete en cascade manuel

```
public static function boot()  
{  
    parent::boot();  
    static::deleting(function ($object) {  
        $object->children()->delete();  
    });  
}
```



ROBUSTESSE DES DONNÉES

- Pas de schéma = données fragiles
- L'application est responsable de la robustesse des données à la place de la BDD

→ Solution : cast des données dans le bon type (typage fort en MongoDB)

```
public function setOrderAttribute($value)
{
    $this->attributes['order'] = (int) $value;
}
```



RELATIONS

- SQL like : 1-1, 1-n, n-n
- Many to many sans table pivot

users

```
{
  _id: "542537f26c16f53747a4cab2",
  name: "John",
  comments_ids: [
    "6c16f53747a4cab2542537f2",
  ]
}
```

comments

```
{
  _id: "6c16f53747a4cab2542537f2",
  content: "Lorem ipsum dolor sit amet...",
  users_ids: [
    "542537f26c16f53747a4cab2",
  ]
}
```

RELATIONS

- Relations embarquées : tout l'objet est contenu dans son parent
- On a bien deux `Model` Laravel avec leurs propres IDs

users

```
{
  _id: "542537f26c16f53747a4cab2",
  name: "John",
  comments: [
    {
      _id: "6c16f53747a4cab2542537f2",
      content: "Lorem ipsum dolor sit amet...",
    }
  ]
}
```



RELATIONS

- Relations custom : libérez votre créativité !
 - Définir quel objet est responsable d'une relation
 - Données pivots complexes
 - Optimisation de requêtes
- Il est facile de faire de l'héritage sous les relations classiques
- Attention à ne pas introduire trop de complexité dans le code



DÉNORMALISATION

- Pas de jointure en MongoDB donc chaque requête coûte cher
Eager loading possible mais peut mieux faire
- Solution : dénormaliser ? Attention aux contraintes de duplication !

users

```
{
  _id: "542537f26c16f53747a4cab2",
  name: "John",
  comments_ids: [
    {
      _id: "6c16f53747a4cab2542537f2",
      title: "Le commentaire du siècle",
    }
  ]
}
```

comments

```
{
  _id: "6c16f53747a4cab2542537f2",
  title: "Le commentaire du siècle",
  content: "Lorem ipsum dolor sit amet...",
  created_at: "2016-11-03T14:16:54.133Z"
}
```

CONSTRUIRE SON SCHÉMA

- Vision métier
 - Vision usage
 - Vision performance
- Les problèmes de performances ou des requêtes complexes à réaliser arrivent vite en cas d'erreur !



Pourquoi... pas MongoDB ?

- Quand la robustesse des données est essentielle
 - E-commerce, paiements
 - Plusieurs applications utilisent la même base de données
- Quand il n'y a pas de complexité dans la modélisation des données
 - Pas la peine de se poser autant de question !
- Pour les petits projets
 - Une dépendance en moins et un projet plus standard
- Si l'équipe de développement est junior
 - Les erreurs arrivent très vite
 - Le recul sur d'autres projets est essentiel
- Besoin de scalabilité intermédiaire
 - Un cluster MongoDB c'est beau mais pas facile à monter !



Fin