

# PHP Blog Project

## 15. API Rest : Insertion de données (Create)

Avant de nous attaquer au développement de la méthode insert() dans BaseRepository nous allons le faire pour ArticleRepository, SerieRepository et TechRepository, cela nous aidera à généraliser pour coder l'insert() de BaseRepository

### La méthode insert dans ArticleRepository

```
api > Repository > ArticleRepository.php > ArticleRepository > insert()
7  class ArticleRepository extends BaseRepository
    1 reference | 0 overrides | prototype
16  public function insert() : Article | bool
17  {
18      $requestBody = HttpRequest::get(HttpReqAttr::BODY);
19      $article = new Article($requestBody);
20      $article->id_article = null;
21      $entityArray = get_object_vars($article);
22      $columns = implode(",", array_keys($entityArray));
23      $values = implode(",", array_map(function () { return "?"; }, $entityArray));
24      $params = array_values($entityArray);
25      $sql = "INSERT INTO article ($columns) VALUES ($values)";
26      $queryResponse = $this->preparedQuery($sql, $params);
27      if($queryResponse->result && $queryResponse->statement->rowCount() == 1){
28          $lastInsertId = self::$connection->lastInsertId();
29          $article->id_article = intval($lastInsertId);
30          return $article;
31      }
32      return false;
33  }
34 }
```

Nous commençons par récupérer le body de la requête (ligne 18) puis nous passons ce tableau associatif en paramètre du constructeur de la classe Article (ligne 19) ce qui permet d'éliminer les éventuelles colonnes inexistantes dans la table article.

Pour rappel, dans le constructeur de la classe BaseEntity, nous n'affectons une valeur que si la propriété existe dans la classe (et donc dans la table correspondante)

```
api > Entity > BaseEntity.php > BaseEntity > __construct()
4  class BaseEntity
    0 references | 0 overrides
6  function __construct($fields = []){
7      $pk = "id_" . lcfirst(str_replace("Entity\\", "", get_called_class()));
8      $this->{$pk} = 0;
9      foreach($fields as $k => $v){
10         if(property_exists($this, $k)){ ←
11             $this->{$k} = $v;
12         }
13     }
14 }
```

Nous affectons ensuite null à l'id\_article (ligne 20) car il s'agit de l'insertion d'une nouvelle ligne. Une fois le body de la requête "nettoyé" grâce à la création d'une entité Article nous récupérons l'objet sous forme de tableau associatif avec la méthode get\_object\_vars() puis nous construisons les variables \$columns, \$values et \$params dont nous aurons besoin pour la requête SQL préparée.

Si la requête s'est bien passée (ligne 27) nous récupérons l'id de la ligne insérée pour l'affecter à notre entité Article que nous retournons.

Si la requête n'a pas fonctionné, nous renvoyons false.

Nous pouvons maintenant utiliser la méthode insert() dans notre méthode post() d'ArticleController pour créer un article

```
api > Controller > ArticleController.php > ArticleController > post()
5  class ArticleController extends BaseController
    0 references | 0 overrides
17  public function post() : array
18  {
19      $articleRepository = new ArticleRepository();
20      $insertedArticle = $articleRepository->insert();
21      return ["result" => $insertedArticle];
22  }
```

Nous testons avec Thunder Client

```
POST http://api.php-blog-project.loc/article Send
Query Headers 2 Auth Body 1 Tests Pre Run
JSON XML Text Form Form-encode GraphQL Binary
JSON Content Format
1  {
2      "nimpotequoi":"adress@email.com",
3      "title":"Titre de l'article",
4      "id_appuser": 10
5  }
```

Et nous obtenons une erreur

```
Status: 200 OK Size: 744 Bytes Time: 456 ms
Response Headers 6 Cookies Results Docs {}
1  <br />
2  <b>Fatal error</b>: Uncaught Error: Cannot access private property
   Repository\ArticleRepository::$connection in C
   : \wamp64\www\php_blog_project\api\Repository\ArticleRepository.php:28
3  Stack trace:
```

C'est dû à la propriété static \$connection de BaseRepository qui est private et donc inaccessible depuis sa classe fille ArticleRepository.

Pour remédier à cela nous la passons en protected

```
api > Repository > BaseRepository.php > BaseRepository
8  class BaseRepository
    4 references
10  protected static $connection = null;
```

Nous testons à nouveau et obtenons cette fois ci un article mais il est incomplet

Status: 200 OK    Size: 78 Bytes    Time: 428 ms

Response    Headers 6    Cookies    Results    Docs

```
1 {
2   "result": {
3     "title": "Titre de l'article",
4     "id_appuser": 10,
5     "id_article": 49
6   }
7 }
```

Pour récupérer complètement l'article inséré en DB nous modifions légèrement la méthode insert() d'ArticleRepository

```
25 $sql = "INSERT INTO article ($columns) VALUES ($values)";
26 $queryResponse = $this->preparedQuery($sql, $params);
27 if($queryResponse->result && $queryResponse->statement->rowCount() == 1){
28     $lastInsertId = self::$connection->lastInsertId();
29     $article = $this->getOneById($lastInsertId);
30     return $article;
31 }
32 return false;
```

Résultat

Status: 200 OK    Size: 180 Bytes    Time: 450 ms

Response    Headers 6    Cookies    Results    Docs





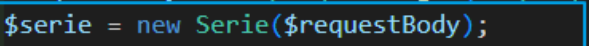

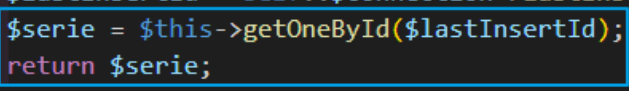
```
1 {
2   "result": {
3     "title": "Titre de l'article",
4     "summary": null,
5     "img_src": null,
6     "published_at": null,
7     "updated_at": null,
8     "is_deleted": null,
9     "id_appuser": 10,
10    "id_serie": null,
11    "id_article": 50
12  }
13 }
```





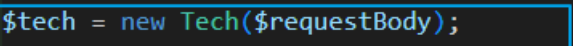

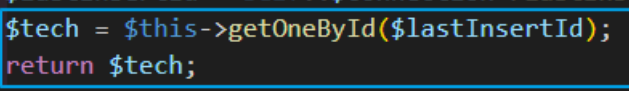
En DB nous avons bien les 2 articles précédemment créés.

id_article	title	summary	img_src	published_at	updated_at	is_deleted	id_appuser	id_serie
50	Titre de l'article	NULL	NULL	NULL	NULL	NULL	10	NULL
49	Titre de l'article	NULL	NULL	NULL	NULL	NULL	10	NULL
48	aliquet pulvinar sed nisl	Morbi vel lectus in quam fringilla rhoncus. Mauris...	https://picsum.photos/id/48/400/300	2022-11-26 00:00:00	2023-06-18 00:00:00	0	10	5

## La méthode insert dans SerieRepository et TechRepository

Les variations de code d'une classe à l'autre sont entourées en bleu pour les méthodes insert()

```
api > Repository >  SerieRepository.php >  SerieRepository >  insert()
6  class SerieRepository extends BaseRepository
    1 reference | 0 overrides
15  public function insert() :  | bool
16  {
17      $requestBody = HttpRequest::get(HttpReqAttr::BODY);
18       $serie = new Serie($requestBody);
19      $serie->id_serie = null;
20      $entityArray = get_object_vars($serie);
21      $columns = implode(",", array_keys($entityArray));
22      $values = implode(",", array_map(function () { return "?"; }, $entityArray));
23      $params = array_values($entityArray);
24      $sql = "INSERT INTO  ($columns) VALUES ($values)";
25      $queryResponse = $this->preparedQuery($sql, $params);
26      if($queryResponse->result && $queryResponse->statement->rowCount() == 1){
27          $lastInsertId = self::$connection->lastInsertId();
28           $serie = $this->getOneById($lastInsertId);
29          return $serie;
30      }
31      return false;
32  }
```

```
api > Repository >  TechRepository.php >  TechRepository >  insert()
6  class TechRepository extends BaseRepository
    1 reference | 0 overrides
15  public function insert() :  | bool
16  {
17      $requestBody = HttpRequest::get(HttpReqAttr::BODY);
18       $tech = new Tech($requestBody);
19      $tech->id_tech = null;
20      $entityArray = get_object_vars($tech);
21      $columns = implode(",", array_keys($entityArray));
22      $values = implode(",", array_map(function () { return "?"; }, $entityArray));
23      $params = array_values($entityArray);
24      $sql = "INSERT INTO  ($columns) VALUES ($values)";
25      $queryResponse = $this->preparedQuery($sql, $params);
26      if($queryResponse->result && $queryResponse->statement->rowCount() == 1){
27          $lastInsertId = self::$connection->lastInsertId();
28           $tech = $this->getOneById($lastInsertId);
29          return $tech;
30      }
31      return false;
32  }
```

## Les méthodes post des contrôleurs

```
api > Controller > SerieController.php > SerieController > post()
1  <?php namespace Controller;
2      use Core\HttpResponse;
3      use Entity\Serie;
4      use Repository\SerieRepository;
5      0 references | 0 implementations
6  class SerieController extends BaseController
7  {
8      0 references | 0 overrides
9      public function get() : array | Serie | null
10     {
11         $serieRepository = new SerieRepository();
12         if($this->id <= 0){
13             $series = $serieRepository->getAll();
14             return $series;
15         }
16         $serie = $serieRepository->getOneById($this->id);
17         return $serie;
18     }
19     0 references | 0 overrides
20     public function post() : array
21     {
22         $serieRepository = new SerieRepository();
23         $insertedSerie = $serieRepository->insert();
24         return ["result" => $insertedSerie];
25     }
26 }
```

```
api > Controller > TechController.php > TechController > post()
1  <?php namespace Controller;
2      use Core\HttpResponse;
3      use Entity\Tech;
4      use Repository\TechRepository;
5      0 references | 0 implementations
6  class TechController extends BaseController
7  {
8      0 references | 0 overrides
9      public function get() : array | Tech | null
10     {
11         $techRepository = new TechRepository();
12         if($this->id <= 0){
13             $techs = $techRepository->getAll();
14             return $techs;
15         }
16         $tech = $techRepository->getOneById($this->id);
17         return $tech;
18     }
19     0 references | 0 overrides
20     public function post() : array
21     {
22         $techRepository = new TechRepository();
23         $insertedTech = $techRepository->insert();
24         return ["result" => $insertedTech];
25     }
26 }
```

Les tests avec Thunder Client

POST

http://api.php-blog-project.loc/serie

Send

Query

Headers 2

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

1

{

2

"nimpotequoi":"adress@email.com",

3

"title":"Titre de la s rie"

4

}

Status: 200 OK

Size: 109 Bytes

Time: 479 ms

Response

Headers 6

Cookies

Results

Docs

1

{

2

"result": {

3

"title": "Titre de la s rie",

4

"summary": null,

5

"img\_src": null,

6

"is\_deleted": null,

7

"id\_serie": 13

8

}

9

}

id_serie	1	title	summary	img_src	is_deleted
	13	Titre de la s�rie	NULL	NULL	NULL
	12	augue luctus tincidunt	Duis ac nibh. Fusce lacus purus, aliquet at, feugi...	https://picsum.photos/id/12/400/300	0

POST

http://api.php-blog-project.loc/tech

Send

Query

Headers 2

Auth

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

JSON Content

Format

1

{

2

"nimpotequoi":"adress@email.com",

3

"label":"Titre de la tech"

4

}

Status: 200 OK

Size: 85 Bytes

Time: 452 ms

Response

Headers 6

Cookies

Results

1

{

2

"result": {

3

"label": "Titre de la tech",

4

"img\_src": null,

5

"is\_deleted": null,

6

"id\_tech": 13

7

}

8

}

id_tech	1	label	img_src	is_deleted
	13	Titre de la tech	NULL	NULL
	12	WAMP	https://picsum.photos/id/12/400/300	0

## Mutualisation dans BaseRepository

En comparant les méthodes insert() des 3 classes Repository, nous pouvons mutualiser le code et créer la méthode insert() dans la classe mère.

```
api > Repository > BaseRepository.php > BaseRepository > insert()
8  class BaseRepository
    3 references | 0 overrides
71  public function insert() : BaseEntity | false
72  {
73      $tableName = $this->getTableName();
74      $requestBody = HttpRequest::get(HttpReqAttr::BODY);
75      $entityClassName = $this->getEntityClassName();
76      $entity = new $entityClassName($requestBody);
77      $entity->{"id_$tableName"} = null;
78      $entityArray = get_object_vars($entity);
79      $columns = implode(",", array_keys($entityArray));
80      $values = implode(",", array_map(function () { return "?"; }, $entityArray));
81      $params = array_values($entityArray);
82      $sql = "INSERT INTO $tableName ($columns) VALUES ($values)";
83      $queryResponse = $this->preparedQuery($sql, $params);
84      if($queryResponse->result && $queryResponse->statement->rowCount() == 1){
85          $lastInsertId = self::$connection->lastInsertId();
86          $entity = $this->getOneById($lastInsertId);
87          return $entity;
88      }
89      return false;
90  }
```

Nous pouvons supprimer (ou commenter dans un premier temps) les méthodes insert() des classes filles et tester avec Thunder Client.

Nous profitons d'être dans la classe BaseRepository pour typer les fonctions (ce qui n'avait pas été fait lors du développement de la classe en MVC)

Nous repassons également la propriété static \$connection en private (elle n'est plus utilisée dans les classes filles)

-> voir page suivante

```

api > Repository > BaseRepository.php > BaseRepository
7 references | 3 implementations
8 class BaseRepository
9 {
10     4 references
11     private static $connection = null;
12     1 reference
13     private function connect() : PDO...
14 }
15 (keyword) function
16 6 references | 0
17 protected function preparedQuery($sql, $params = []) : object...
18 }
19
20 2 references
21 private function getBaseClassName() : string...
22 }
23 3 references
24 private function getTableName() : string...
25 }
26
27 3 references
28 private function getEntityClassName() : string...
29 }
30
31 3 references | 0 overrides
32 public function getAll() : array...
33 }
34
35 4 references | 0 overrides
36 public function getOneById($id) : BaseEntity | null...
37 }
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77

```

A vous d'essayer de **généraliser les méthodes get et post des contrôleurs** pour les créer dans leur classe mère BaseController.

git :

[https://github.com/DWWM-23526/PHP\\_BLOG\\_PROJECT/tree/Step15](https://github.com/DWWM-23526/PHP_BLOG_PROJECT/tree/Step15)