Compte Rendu Big Data

Eloan ANDRE

January 2026

Contents

1 Afficher les manuscrits

Référence	Titre	Auteur	Actions
510	Correspondance, vue générale sur la ci-devant province d'Auvergne haute et basse	Pierre-Marie Gault de Saint-Germain	voir / éditer / supprimer
509	Géographie de tous les châteaux, des ci-devant fiefs, maisons de campagne et autres lieux remarquables	Jean Deval de Saunade	voir / éditer / supprimer
517	Mémoire sur la province d'Auvergne fait, en 1698	Antoine-François Le Fèvre d' Ormesson	voir / éditer / supprimer
519	Mémoire sur l'Auvergne, pour servir de supplément à celui de M. d'Ormesson	Simon-Charles -Sébastien Bernard Ballainvilliers	voir / éditer / supprimer
373	Grammaire comparée par P. Leclerc. Cours professé à la faculté des lettres de Clermont. 1883-1888		voir / éditer / supprimer
521	Mémoire sur l'Auvergne, pour servir de supplément à celui de M. d'Ormesson	Simon-Charles -Sébastien Bernard Ballainvilliers	voir / éditer / supprimer
480	Etat de la division de l'Armée Royale aux ordres de Monsieur et de Mgr Comte d'Artois en 1792" [par le Comte d'Espinchal (?)]	Joseph-Thomas Espinchal	voir / éditer / supprimer
380	Sources de l'histoire sous le gouvernement personnel de Louis XIV	Georges Desdevises du Dézert	voir / éditer / supprimer
527	Etat général de la population de la généralité de Riom, année 1785.		voir / éditer / supprimer
503	Géographie de toutes les villes, bourgs, paroisses et chapelles de l'Auvergne haute et basse, et de toutes celles qui se trouvent sur ses frontières	Jean Deval de Saunade	voir / éditer / supprimer
507	Géographie de toutes les villes, bourgs, paroisses et chapelles de l'Auvergne haute et basse, et de toutes celles qui se trouvent sur ses frontières	Jean Deval de Saunade	voir / éditer / supprimer
525	Procès-verbal de la tournée faite par M. Meulan, receveur général des finances d'Auvergne, en exécution des ordres de monseigneur le contrôleur général, contenus dans sa lettre du dernier juillet 1740.	Pierre-Louis-Nicolas Meulan	voir / éditer / supprimer
493	Notes historiques et tableaux généalogiques des familles de Sévigné et de Grignan, par Paul Le Blanc	Paul Le Blanc	voir / éditer / supprimer

Afin d'afficher l'ensemble des manuscrits, il faut tout d'abord les récupérer les différents manuscrits présents dans la base mongodb, pour cela, nous la stockons dans une liste à l'aide de :

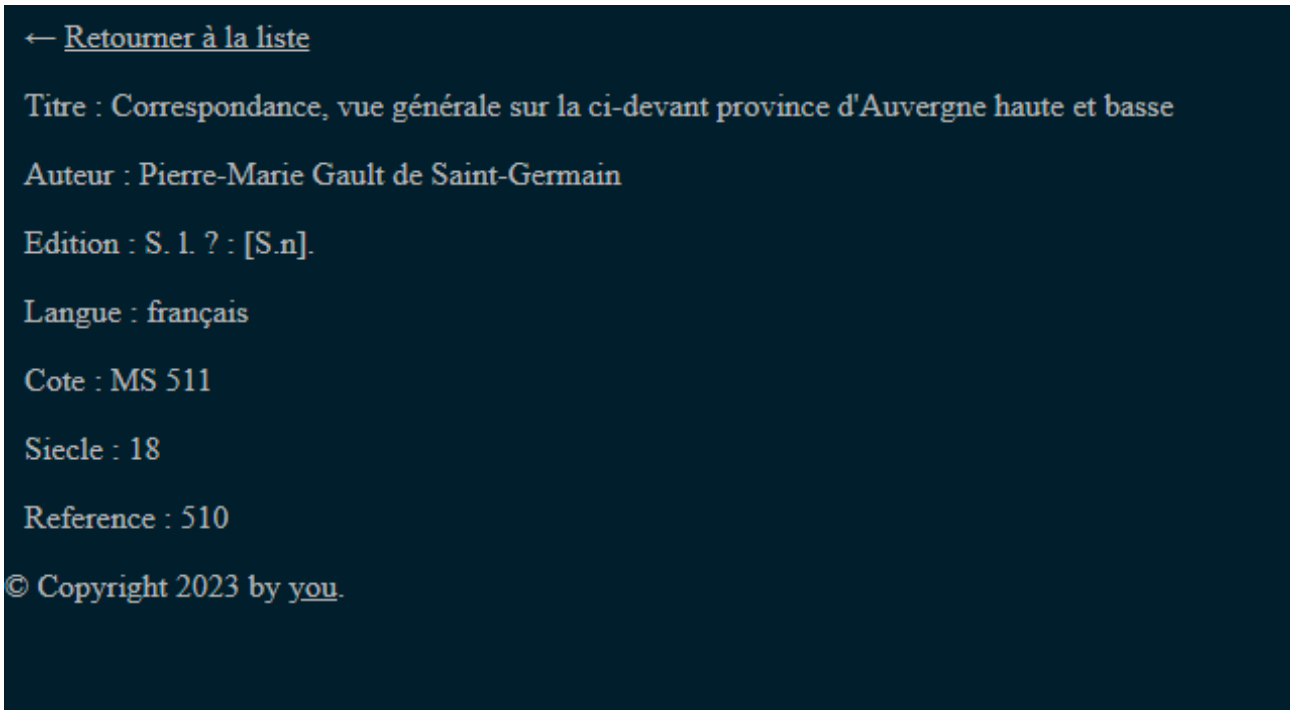
```
$manager->selectCollection('tp')->find([])->toArray();
```

qui permet de récupérer, sans filtre, l'ensembles des livres de la collection. Nous verrons ensuite que nous allons rajouter des filtres afin de paginer la page.
Nous affichons ensuite cette liste dans le front .twig :

```
{% for document in list %}
<tr>
    <td>{{ document['objectid'] }}</td>
    <td>{{ document['titre'] }}</td>
    <td>{{ document['auteur'] }}</td>
    <td>
        <a href="get.php?id={{document['_id']}}&page={{page}}">voir</a>
        <a href="edit.php?id={{document['_id']}}&page={{page}}">éditer</a>
        <a href="delete.php?id={{document['_id']}}&page={{page}}">supprimer</a></td>
</tr>
{% endfor %}
```

Puisque nous ne sommes que dans la page principale, nous n'affichons que la référence, le titre et l'auteur du livre, nous permettons ensuite une redirection vers les pages d'édition, d'affichage et de suppression en passant l'id du document en paramètre GET afin de les récupérer sur cette page.

2 Récupération d'un manuscrit



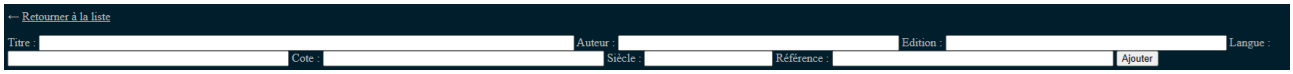
La récupération des informations d'un manuscrit passe par l'id donné dans le GET de la page :

```
$entity = $manager->selectCollection('tp')->findOne(['_id' => new ObjectId($_GET['id'])]);
```

Ensuite, nous affichons cette élément dans le front .twig correspondant :

```
<p>&larr; <a href="/index.php?page={{page}}">Retourner à la liste</a></p>
<p> Titre : {{ entity['titre'] }} </p>
<p>Auteur : {{ entity['auteur'] }} </p>
<p> Edition : {{ entity['edition'] }} </p>
<p> Langue : {{ entity['langue'] }} </p>
<p> Cote : {{ entity['cote'] }} </p>
<p> Siecle : {{ entity['siecle'] }} </p>
<p> Reference : {{ entity['objectid'] }} </p>
```

3 Création d'un manuscrit



La création d'un manuscrit se fait depuis un formulaire POST, lorsque ce formulaire n'est pas rempli, la page create.php affiche le front, sinon, elle crée l'élément et renvoit à la page principale.

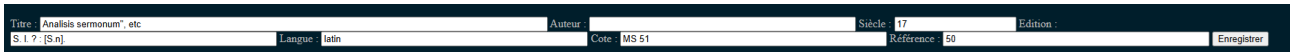
```
<form action="/create.php" method="post">
    <label for="title">Titre :</label>
    <input type="text" id="title" name="title" size="100">

    <label for="author">Auteur :</label>
    <input type="text" id="author" name="author" size="50">

    <label for="edition">Edition :</label>
    <input type="text" id="edition" name="edition" size="50">

    <label for="langue">Langue :</label>
    <input type="text" id="langue" name="langue" size="50">

    <label for="cote">Cote :</label>
    <input type="text" id="cote" name="cote" size="50">

    <label for="century">Siècle :</label>
    <input type="number" id="century" name="century">

    <label for="objectid">Référence :</label>
    <input type="text" id="objectid" name="objectid" size="50">

    <button type="submit">Ajouter</button>
</form>
```

Ensuite, nous pouvons ajouter le document dans la base MongoDB :

```
$doc = [
    "titre" => $_POST["title"],
    "auteur" => $_POST["author"],
    "edition" => $_POST["edition"],
```

```
    "langue" => $_POST["langue"],
    "cote" => $_POST["cote"],
    "siecle" => $_POST["century"],
    "objectid" => $_POST["objectid"],
];
$result = $manager->selectCollection('tp')->insertOne($doc);
```

4 Édition d'un document



L'édition d'un document se fait sur 2 pages, la première affiche la page où seront affichés les différentes informations à modifier de notre manuscrit, pour cela, nous avons besoin de récupérer les informations actuelles grâce au paramètre GET de la page qui nous fournit l'identifiant du manuscrit :

```
$entity = $manager->selectCollection('tp')->findOne(['_id' => new ObjectId($_GET['id'])]);
```

Ensuite, l'édition se fait via un formulaire POST qui redirigera sur la page traitant le formulaire au moment de l'enregistrement :

```
<form action="/update.php?id={{entity['_id']}}&page={{page}}" method="post">
    <input type="hidden" id="id" name="id" value="{{ entity._id }}">

    <label for="titre">Titre :</label>
    <input type="text" id="titre" name="titre" value="{{entity["titre"]}}" size="100">

    <label for="auteur">Auteur :</label>
    <input type="text" id="auteur" name="auteur" size="50" value="{{ entity["auteur"]}}">

    <label for="siecle">Siècle :</label>
    <input type="number" id="siecle" name="siecle" value="{{  entity["siecle"] }}">

    <label for="edition">Edition :</label>
    <input type="text" id="edition" name="edition" size="50" value="{{ entity["edition"] }}">

    <label for="langue">Langue :</label>
    <input type="text" id="langue" name="langue" size="50" value="{{ entity["langue"] }}">

    <label for="cote">Cote :</label>
    <input type="text" id="cote" name="cote" size="50" value="{{ entity["cote"] }}">

    <label for="objectid">Référence :</label>
    <input type="text" id="objectid" name="objectid" size="50" value="{{  entity["objectid"] }}">

    <button type="submit">Enregistrer</button>
</form>
```

Ensuite, la page de traitement modifiera l'ensemble du document à l'aide de son identifiant, puis redirigera vers la page principale :

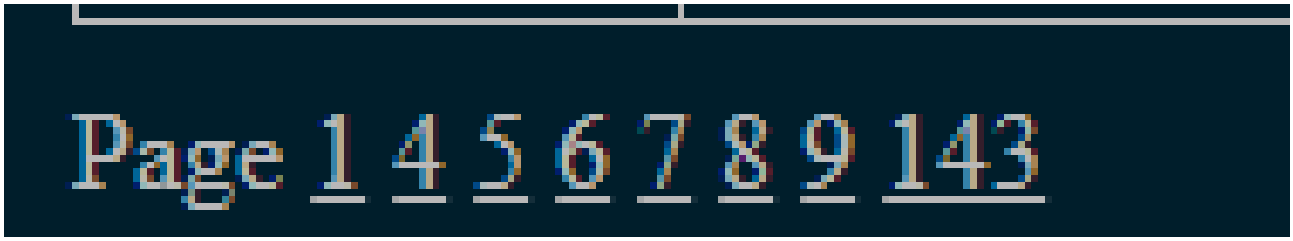
```
$manager->selectCollection('tp')->updateOne(
[ '_id' => new ObjectId($_GET['id']) ],
[ '$set' => [
    'titre' => $_POST['titre'],
    'auteur' => $_POST['auteur'],
    'siecle' => $_POST['siecle'],
    'edition' => $_POST['edition'],
    'langue' => $_POST['langue'],
    'cote' => $_POST['cote'],
    'objectid' => $_POST['objectid'],
]]
);
header('Location: /index.php?page=' . $page);
```

5 Suppression d'un document

La page de traitement de la suppression d'un document supprime un document à l'aide de son identifiant :

```
$manager->selectCollection('tp')->deleteOne(['_id' => new ObjectId($_GET['id'])]);
```

6 Pagination



La pagination permet de ne pas charger l'ensemble des manuscrits de la base MongoDB, pour cela, nous avons besoin de connaitre :
- La page actuelle
- Le nombre d'éléments par page (constante définie par le développeur)
- Le nombre de page totale

La page actuelle est récupérer par un paramètre GET de la page, si celui-ci n'est pas fourni, nous considérons que nous sommes à la première page :

```
$page = (int)($_GET['page'] ?? "1");
```

Pour le nombre de page totale, nous avons besoin de connaitre le nombre total de manuscrits ainsi que le nombre d'éléments par page. Ensuite, ce nombre est arrondi à la page supérieur, car la dernière page ne contient pas forcément autant d'éléments que le nombre de manuscrits par page.

```
$pageSize = 20;
$count = $manager->selectCollection('tp')->countDocuments();
$nbPage = (int)ceil($count / $pageSize);
```

Ensuite, nous avons besoin de récupérer les manuscrits de la page correspondante, nous avons donc besoin de *pageSize* éléments et de commencer à partir du *pageSize * (page-1)*ième éléments :

```
$list = $manager->selectCollection('tp')->find([], [
    'limit' => $pageSize,
    'skip' => $pageSize * ($page-1)
])->toArray();
```

Ensuite, le numéro de page actuelle est passer aux différentes pages afin de pouvoir revenir à la page courante lors du retour en arrière.

Afin de changer de page, nous avons besoin de pouvoir nous déplacer :
- A la première page
- A la dernière page
- Jusqu'à 2 pages en arrières
- Jusqu'à 3 pages en avant

Nous procédons donc de la manière suivante afin de ne pas dépasser les limites des nombres de pages :

```
<a href="app.php?page=1">1</a>
    {% for i in max(2, (page-2))..min((page+3), nbPage-1) %}
        <a href="app.php?page={{i}}">{{i}}</a>
    {% endfor %}
<a href="app.php?page={{nbPage}}">{{ nbPage }}</a>
```

7 Redis

Nous avons décider de mettre en place différents caches :
- Un cache pour tout les manuscrits d'une page, de clé *tp_page:[page]*
- Un cache pour le nombre d'éléments totaux, de clé *tp_count*
- Un cache pour chaque manuscrit, de clé *tp_element:[id_manuscrit]*

Le cache de l'ensemble des manuscrits d'une page est donc charger en même temps que la page, si celui-ci existe déjà, on le charge depuis la base redis, sinon, depuis la base MongoDB puis on le cache dans la base redis pour 300 secondes :

```
 if ($redisClient && $redisClient->exists($cacheKey))
    {
        $list = json_decode($redisClient->get($cacheKey), true);

        # Déserialisation de l'id
        foreach ($list as &$item) {
            if (isset($item['_id']['$oid'])) {
                $item['_id'] = $item['_id']['$oid'];
            }
        }
    }
    else
    {
        $list = $manager->selectCollection('tp')->find([], [
```

```
            'limit' => $pageSize,
            'skip' => $pageSize * ($page-1)
        ])->toArray();
        $redisClient?->setex($cacheKey, 300, json_encode($list));
    }
```

Nous utilisons la même approche lors de la consultation d'un manuscrit et le calcul du nombre total de manuscrits pour leurs mise en cache.

Lors de la modification d'un document, nous modifions le cache du manuscrit correspondant et nous invalidont celui de la page dont il provient, afin de ne pas affichés des informations dépassés :

```
$redisClient?->setex($cacheKey, 300, json_encode($_POST));
$redisClient?->del($cacheKeyPage);
```

Lors de la suppression d'un document, nous avons besoin de non seulement supprimer cet élément du cache, le nombre de manuscrits total, mais aussi d'invalider tout les caches de pages qui suivait celle actuelle, car, un décalage s'est fait de tout les éléments. Ainsi, nous scannons toutes les pages qui ont été mises en cache, et nous supprimons celles qui sont supérieurs à la page courante :

```
$cursor = 0;

do {
    [$cursor, $keys] = $redisClient?->scan($cursor, ['MATCH' => 'tp_page:*']);

    $toDelete = array_filter($keys, function($k) use ($page) {
        return preg_match('/tp_page:(\d+)/', $k, $m) && (int)$m[1] >= $page;
    });

    if ($toDelete) {
        $redisClient?->del($toDelete);
    }

} while ($cursor);
```

8 Tests de charges

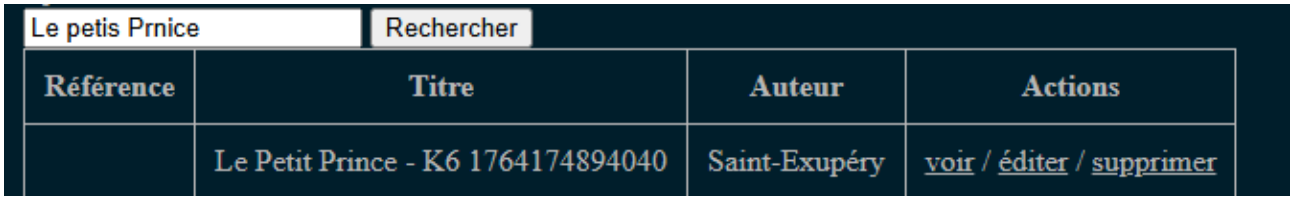
Afin de réaliser les tests de charges, nous avons demander à ChatGPT de nous écrire un scénario de tests.
Ce scénario considère que :
- 80% des utilisateurs effectuent de la lecture sur une des 5 premières page aléatoire
- 15% des utilisateurs effectuent une création de document
- 5% des utilisateurs effectuent une suppression de document

Ensuite, un tests de charge est effectué selon les stages suivants :
- Les 5 premières secondes, 5 utilisateurs utilisent l'application
- Pendant 10 secondes, 10 utilisateurs utilisent l'application
- Pendant 10 secondes, 50 utilisateurs utilisent l'application (pic de charge)
- Pendant 5 secondes, 10 utilisateurs utilisent l'application
- Pendance 5 secondes, 5 utilisateurs utilisent l'application

Nous effectuons ensuite le test avec ou sans l'activation du cache. Nous avons observer que ce temps n'était pas très différents dans les 2 cas.
Cela peut s'expliquer par notre jeu de tests qui peut ne pas refleter assez un parcours réelle d'utilisateur et invalider trop souvent le cache de l'application.
Cela peut aussi s'expliquer par le fait que notre mise en cache n'est pas optimale et est validées beaucoup trop souvent et doit donc être rechargé.

Les résultats se retrouvent ici :
Test sans cache : ./test_no_cache.txt
Test avec cache : ./test_with_cache.txt

9 Elasticsearch



Elasticsearch permet à l'application de gérer un système de recherche intelligent, une première indexation doit être réalisé pour indexer l'ensemble des documents actuellement présent en base.
Premièrement, nous avons besoin créer un index pour pouvoir les retrouver, si l'index existe déjà, nous le supprimons puisque indexation.php n'est appellé qu'une seule fois.

```
$indexName = 'books';
$client->indices()->delete([
    'index' => $indexName
]);
$client->indices()->create([
    'index' => $indexName,
]);
```

Ensuite, nous récupérons l'ensemble des documents, et pour chacun d'entre eux, nous les indexons à l'aide de leur identifiant. Étant donné qu'il peut y avoir beaucoup de documents, nous procédons par batch : Nous ne sauvegardons l'indexation que tout les 1000 manuscrits afin de limiter les appels réseaux :

```
$i = 0;
foreach ($list as $document) {


    $params['body'][] = [
        'index' => [
            '_index' => $indexName,
            '_id'    => (string) $document['_id'],
        ]
    ];

    $params['body'][] = [
```

```
            'titre'    => $document['titre'],
            'auteur' => $document['auteur'],
            'edition' => $document['edition'],
            'langue' => $document['langue'],
            'cote' => $document['cote'],
            'siecle' => $document['siecle'],
            'objectid' => $document['objectid'],
        ];

        if ($i % 1000 == 0) {
            $responses = $client->bulk($params);

            // erase the old bulk request
            $params = ['body' => []];

            // unset the bulk response when you are done to save memory
            unset($responses);
        }

        $i = $i + 1;
    }

    if (!empty($params['body'])) {
        $responses = $client->bulk($params);
    }
```

Nous avons aussi besoin d'indexer un nouvel élément à la création, le réindexer lors de sa modification et le désindexer lors de sa suppression, cela est fait depuis les 3 pages correspondantes :

Création :

```
$esClient->index([
    'index' => $indexName,
    'id' => (string) $result->getInsertedId(),
    'body' => [
        'titre'    => $_POST['title'],
        'auteur' => $_POST['author'],
        'edition' => $_POST['edition'],
        'langue' => $_POST['langue'],
        'cote' => $_POST['cote'],
        'siecle' => $_POST['century'],
        'objectid' => $_POST['objectid'],
    ]
]);
```

Édition :

```
$esClient->update([
    'index' => $indexName,
    'id'    => (string) $_GET['id'],
    'body'  => [
        'doc' => [
            'titre'    => $_POST['titre'],
```

```
            'auteur'   => $_POST['auteur'],
            'siecle'   => $_POST['siecle'],
            'edition'  => $_POST['edition'],
            'langue'   => $_POST['langue'],
            'cote'     => $_POST['cote'],
            'objectid' => $_POST['objectid'],
        ]
    ]
]);
Suppression :
$esClient->delete([
    'index' =>  $indexName,
    'id'    => (string) $_GET['id']
]);
```

Pour utiliser cette indexation, nous avons ajouter une barre de recherche lié à un formulaire GET qui recharge la page avec la recherche :

```
<form method="get" action="">
    <input type="text" name="book" placeholder="Rechercher un livre ou un auteur" value="{{ book|defaul
    <button type="submit">Rechercher</button>
</form>
```

Ensuite, si la page principale récupère se paramètre GET, il n'effectue pas le cache ni la pagination, car cela aurait peu de sens pour une recherche qui retourne peu d'éléments, il utilise ensuite Elasticsearch pour rechercher les ids des documents correspondant à la recherche "flou" sur le titre et l'auteur afin de pouvoir détecter à la fois les fautes de frappes, mais aussi la mise au pluriel de certains mots.
Après avoir récupéré cette liste, il récupères les ids correspondants depuis la base mongodb et les affiche.

```
$response = $elasticClient->search([
        'index' => $indexName,
        'body'  => [
            'query' => [
                'multi_match' => [
                    'query'  => $searchQuery,
                    'fields' => ['titre', 'auteur'],
                    'fuzziness' => 'AUTO',
                ]
            ]
        ]
    ]);

    $hits = $response['hits']['hits'] ?? [];
    $filteredIds = [];
    foreach ($hits as $hit) {
        $filteredIds[] = $hit['_id'];
    }

    $collection = $manager->selectCollection('tp');
```

```
if (!empty($filteredIds)) {
    $list = $collection->find([
        '_id' => ['$in' => array_map(fn($id) => new MongoDB\BSON\ObjectId($id), $filteredIds)]
    ])->toArray();
} else {
    header('Location: /index.php');
}
```

10 Conclusion

Ces TP m'ont donc appris à concevoir une page web en utilisant des solutions NoSQL.

J'ai ainsi pu apprendre à me servir, dans un contexte PHP des outils tels que MongoDB, Redis, Graphana/K6 et de ElasticSearch.

En complément, j'ai aussi pu revoir mes bases en PHP ainsi que d'utiliser twig que je n'avais jamais utiliser auparavant.