Intégration de MongoDB - EcoRide

Projet de covoiturage écoresponsable

A. Installation de MongoDB

i. Installation en local

Vérifier la version PHP exacte
Dans un **Invite de commandes** (cmd) ou PowerShell :

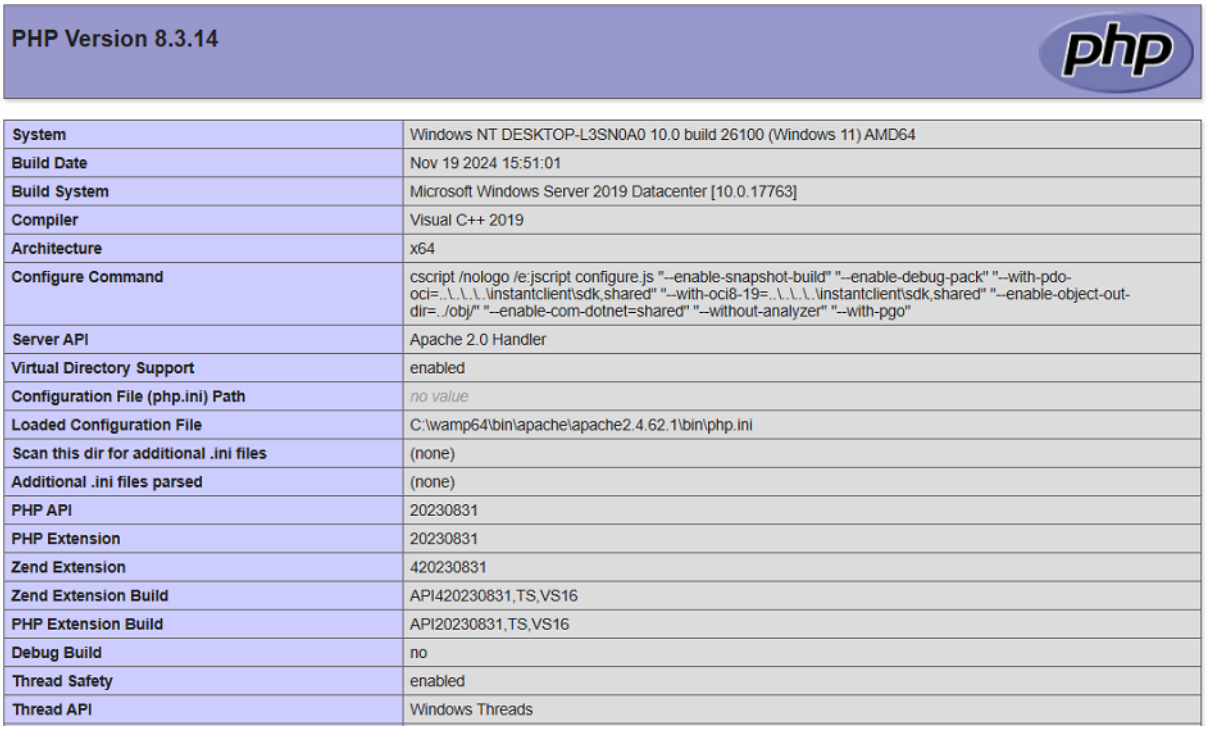
```
$  php -v
   php -i | findstr /I "Thread Safety"
```

Si nous sommes en **PHP 8.3.14**, **Thread Safety = enabled (ZTS)**, **x64** → il nous faut un **php_mongodb.dll pour PHP 8.3.14, TS, x64**.

Installation en 4 mini-étapes :

Étape 1 — Installation de MongoDB

1) Vérifier la version PHP utilisée par Apache
Dans le navigateur : **http://localhost** → menu Wamp → **phpinfo()**.
Notons la **version PHP chargée par Apache** et le **"Loaded Configuration File"** (ça nous dira quel php.ini modifier).
Possible que la version Apache ≠ 8.3.14 (CLI). Si c'est différent, on téléchargera la DLL correspondant **à la version Apache**, pas la CLI.



PHP Version 8.3.14

System	Windows NT DESKTOP-L3SN0A0 10.0 build 26100 (Windows 11) AMD64
Build Date	Nov 19 2024 15:51:01
Build System	Microsoft Windows Server 2019 Datacenter [10.0.17763]
Compiler	Visual C++ 2019
Architecture	x64
Configure Command	cscript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=..\..\..\..\instantclient\sdk,shared" "--with-oci8-19=..\..\..\..\instantclient\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	*no value*
Loaded Configuration File	C:\wamp64\bin\apache\apache2.4.62.1\bin\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20230831
PHP Extension	20230831
Zend Extension	420230831
Zend Extension Build	API420230831,TS,VS16
PHP Extension Build	API20230831,TS,VS16
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads

On a tout ce qu'il faut. Nous sommes sur **Apache + PHP 8.3.14 TS x64**, et le **php.ini chargé par Apache** est :
C:\wamp64\bin\apache\apache2.4.62.1\bin\php.ini

2) Télécharger le bon php_mongodb.dll

- Prendre le binaire Windows de l'extension **MongoDB** pour **PHP 8.3.14**, **TS**, **x64**.
- Le fichier ressemble à : php_mongodb-...-8.3-**ts**-**x64**.dll
- Renommons-le en **php_mongodb.dll** si besoin.

Pour télécharger l'extension **MongoDB** adaptée à l'environnement (**PHP 8.3.14, Thread Safe, x64, VS16** sous WampServer 3.3.7), voici exactement ce que nous devons faire :

1️⃣ Aller sur le site officiel de PECL

- Ouvrons https://pecl.php.net/package/mongodb
- Cliquons sur la dernière version stable (actuellement **2.1.1**)
- Cliquons sur le lien DLL (https://pecl.php.net/package/mongodb/2.1.1/windows) à droite de mongodb-2.1.1.tgz (2106.0kB)
- Allons dans le tableau sur la version de PHP et cliquons sur 8.3 Thread Safe (TS) x64 (https://downloads.php.net/~windows/pecl/releases/mongodb/2.1.1/php_mongodb-2.1.1-8.3-nts-vs16-x64.zip)

📌 Pourquoi c'est le bon fichier pour nous

- php_mongodb → extension MongoDB
- 2.1.1 → version stable la plus récente
- 8.3 → la version de PHP
- ts → Thread Safe (le PHP l'est)
- vs16 → compilé avec Visual Studio 2019 (utilisé pour PHP 8.x)
- x64 → architecture 64 bits (Wamp 3.3.x est en 64 bits)

3) Étapes après le téléchargement

1. Dézippons le fichier → nous obtennons **php_mongodb.dll**
2. Copions php_mongodb.dll dans :
   C:\wamp64\bin\php\php8.3.14\ext
3. Éditons :
   C:\wamp64\bin\apache\apache2.4.62.1\bin\php.ini
   et ajoutons à la fin :
   extension=php_mongodb.dll
4. (Optionnel mais recommandé) Éditons aussi :
   C:\wamp64\bin\php\php8.3.14\php.ini
   et ajoutons la même ligne.

5. Redémarrons Wamp (Restart All Services)
6. Vérifions :

```
$  php -m | findstr /I mongo
```

→ doit afficher mongodb.

Étape 2 — Configuration .env.local et dépendance Composer

1️⃣ Installer la librairie PHP officielle MongoDB

Dans le projet Symfony :

```
$  composer require mongodb/mongodb
```

C'est l'API PHP haut-niveau qui facilitera la connexion et les requêtes.

2️⃣ Ajouter les variables d'environnement

Dans **.env.local** (crée-le s'il n'existe pas) :

```
### MongoDB ###
MONGODB_URI="mongodb://127.0.0.1:27017"
MONGODB_DB="ecoride"
MONGODB_COLLECTION="logs"
```

Étape 3 — Déclarer MongoDB dans Symfony

Ouvrons **config/services.yaml** et ajoutons en bas :

```
    MongoDB\Client:
        arguments:
            - '%env(MONGODB_URI)%'

    App\Service\MongoLogService:
        arguments:
            $client: '@MongoDB\Client'
            $dbName: '%env(MONGODB_DB)%'
            $collectionName: '%env(MONGODB_COLLECTION)%'
```

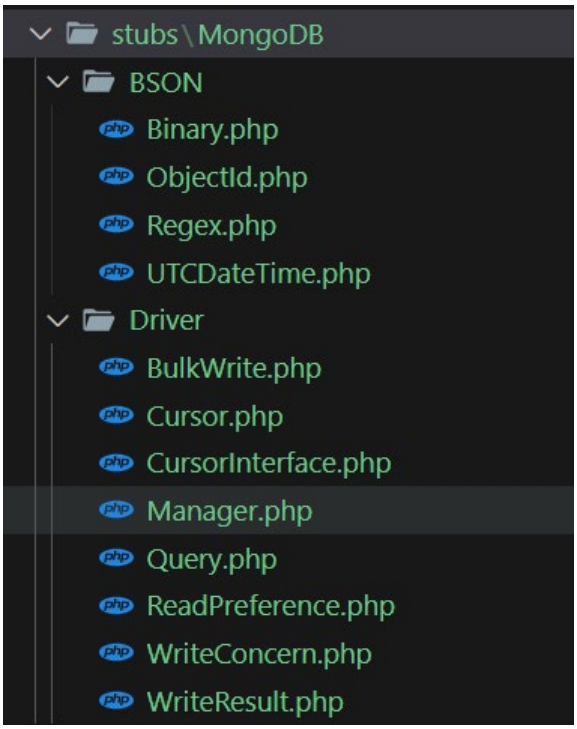
Étape 4 — Création des stubs pour VSCode

- Les **stubs** sont des fichiers .php contenant uniquement les signatures des classes et méthodes de l'extension (par ex. MongoDB\Client, MongoDB\BSON\UTCDateTime, etc.).
- Ils n'ajoutent **aucune fonctionnalité** à PHP → c'est juste pour **l'auto-complétion et la suppression des faux "undefined"** dans VSCode/Intelephense.
- Le package mongodb/mongodb (Composer) **ne fournit pas** ces stubs.

Processus :

1. Nous dézippons tous les fichiers de **mongodb_full_stubs_complete.zip**.
2. Nous les mettons dans un dossier stubs/ à la racine du projet Symfony (**C:\<Ton dossier>\sym-ecoride\stubs**).
3. Nous ajoutons le chemin dans les paramètres VS Code pour Intelephense.

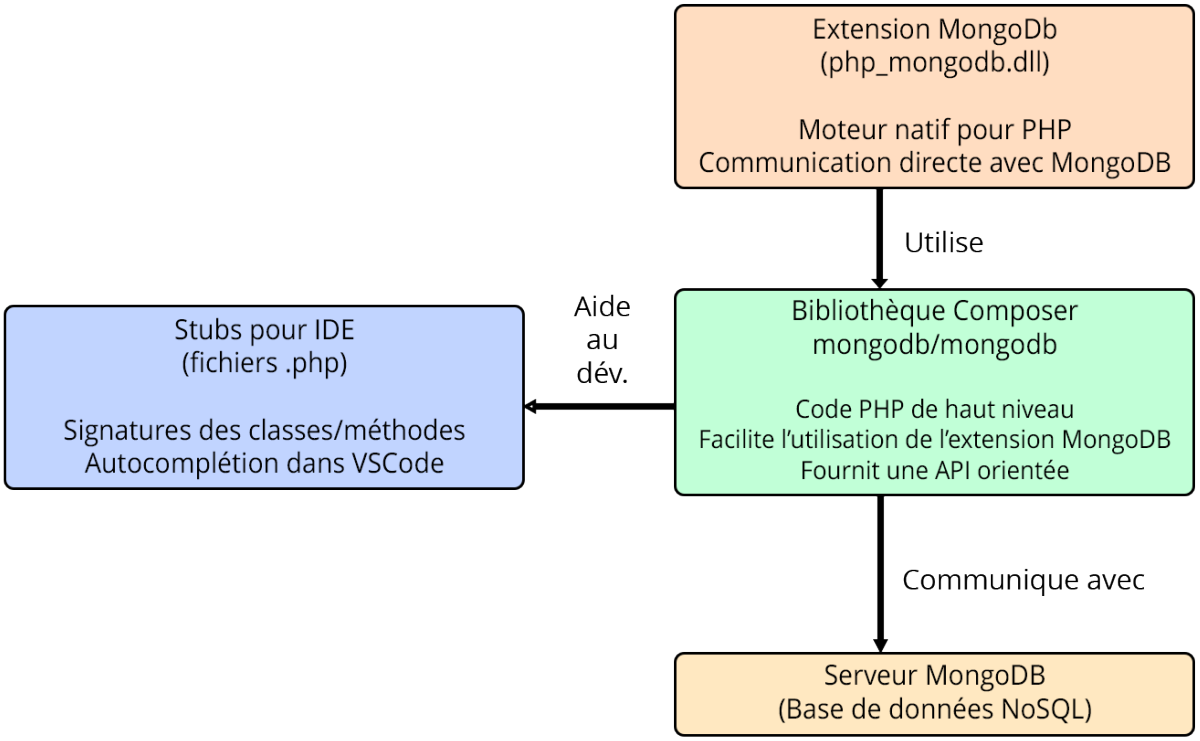
📁 **Organisation finale dans le projet**



Paramétrage VS Code (CTRL+SHIFT+P)
Dans settings.json de VS Code :

```
"intelephense.environment.includePaths": [
    "stubs"
]
```

Ensuite **redémarrons VS Code**.



ii. Installation en production

En production, le plus simple est effectivement de passer par **MongoDB Atlas** (hébergement cloud officiel de MongoDB).

En plus de toutes les étapes à respecter ci-dessus, voici la procédure **pas à pas.**

Création du compte MongoDB Atlas

1. Allons sur https://www.mongodb.com/cloud/atlas
2. Créons un compte (gratuit possible, offre **Free Tier**).
3. Connectons-nous à l'interface Atlas.

Création d'un cluster

1. Dans le dashboard Atlas, cliquons sur **Build a Database**.
2. Choisissons **Free** (M0 Sandbox, 512Mo, suffisant !).
3. Choisissons un fournisseur cloud (**AWS**, Azure, GCP) → peu importe pour le test.
4. Choisissons une région proche de tes utilisateurs (ex. Paris si dispo).
5. Donnons un nom au cluster (ex. **ecoride-cluster**).
6. Cliquons **Create Cluster** (⚠️ la création peut prendre 1-3 minutes).

Création de l'utilisateur MongoDB

1. Allons dans **Database Access** (menu gauche).
2. Cliquons **Add New Database User**.
3. Nom d'utilisateur : ecoride_user (par exemple).

4. Mot de passe : généré ou choisi.
5. Donnons-lui le rôle **Atlas Admin** ou **Read and write to any database** (suffisant pour notre usage).
6. Sauvegardons.

Autorisation des IP

1. Allons dans **Network Access** (menu gauche).
2. Cliquons **Add IP Address**.
3. Si notre serveur a une IP fixe → mettons-la ici.
4. Si nous voulons tout autoriser (⚠️ moins sécurisé) → choisissons **Allow access from anywhere (0.0.0.0/0)**.
5. Sauvegardons.

Différences :

- **Add Your Current IP Address** → autorise uniquement **l'IP actuelle**. ⚠️ Si nous changeons de connexion (ex. Wi-Fi, 4G), ça ne marchera plus sans réajouter la nouvelle IP.
- **Add a Different IP Address** → nous mettons une IP fixe spécifique (utile si le serveur a une IP dédiée).
- **Allow Access from Anywhere** (0.0.0.0/0) → toutes les IP peuvent se connecter (pratique en dev/démo, mais moins sécurisé).

Connexion depuis Symfony

1. Dans **Clusters** → **Connect** → **Connect your application**.
2. Choisissons **Driver: PHP**, version >=8.0.
3. Copions l'URL de connexion donnée, elle ressemble à :

```
mongodb+srv://ecoride_user:<db_password>@ecoride-cluster.dlcsfia.mongodb.net/?retryWrites=true&w=majority&appName=ecoride-cluster
```

4. Dans .env.prod (ou sur l'hébergeur), mettons :

MongoDB
MONGODB_URI="mongodb+srv://ecoride_user:<db_password>@ecoride-cluster.dlcsfia.mongodb.net/?retryWrites=true&w=majority&appName=ecoride-cluster
MONGODB_DB="ecoride"
MONGODB_COLLECTION="logs"

Nous allons simplement remplacer <db_password> par le mot de passe que nous avons créé pour l'utilisateur ecoride_user.

Points importants

- Si le mot de passe contient des caractères spéciaux (@, #, &, etc.), il faut **les encoder en URL**.
  Par exemple, @ devient %40, # devient %23…
- Nous pouvons utiliser un encodeur comme https://www.urlencoder.org/.

- Mettons bien ces valeurs dans .env.prod (ou dans les variables d'environnement de l'hébergeur).
- En local, nous pouvons garder mongodb://127.0.0.1:27017.

Sur Hostinger, l'extension **MongoDB PHP** est disponible, mais **pas activée par défaut** sur tous les plans.

📌 Ce qu'il faudra vérifier juste après le déploiement :

1. Se connecter en SSH sur l'hébergement.
2. Tapons :

```
$  php bin/console about
   php -r "var_dump(extension_loaded('mongodb'));"
```

   - Si nous voyons mongodb → tout est bon.
   - Si rien ne s'affiche → il faudra l'activer depuis le **hPanel**.

📌 Pour activer sur Hostinger :

1. Allons dans **hPanel** → **Advanced** → **PHP Configuration**.
2. Onglet **Extensions**.
3. Cochons mongodb et sauvegarde.
4. Redémarrons le service PHP (1-2 minutes).

Voici un mini script Symfony que l'on peut ajouter dans src/Command/TestMongoConnectionCommand.php pour vérifier :

1. **Que l'extension MongoDB est bien activée**
2. **Que la connexion à MongoDB Atlas fonctionne**
3. **Que ta collection logs est accessible**

```php
<?php

namespace App\Command;

use MongoDB\Client;
use Symfony\Component\Console\Attribute\AsCommand;
use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;

#[AsCommand(
    name: 'app:test:mongo',
    description: 'Teste la connexion à MongoDB et la présence de l\'extension'
)]
class TestMongoConnectionCommand extends Command
{
    public function __construct(private Client $mongoClient, private string $dbName,
private string $collectionName)
    {
        parent::__construct();
    }
```

```
    protected function execute(InputInterface $input, OutputInterface $output): int
    {
        // 1. Vérifier si l'extension MongoDB PHP est chargée
        if (!extension_loaded('mongodb')) {
            $output->writeln('<error>❌ L\'extension MongoDB PHP n\'est pas activée
!</error>');
            return Command::FAILURE;
        }
        $output->writeln('<info>☑ L\'extension MongoDB PHP est bien activée.</info>');

        try {
            // 2. Tester la connexion
            $output->writeln('<info>⏳ Tentative de connexion à MongoDB...</info>');
            $db = $this->mongoClient->selectDatabase($this->dbName);

            // 3. Vérifier la collection
            $collection = $db->selectCollection($this->collectionName);
            $count = $collection->countDocuments();

            $output->writeln("<info>☑ Connexion réussie à la base '{$this->dbName}',
collection '{$this->collectionName}'</info>");
            $output->writeln("<comment>📦 Nombre de documents : {$count}</comment>");
        } catch (\Exception $e) {
            $output->writeln('<error>❌ Erreur de connexion à MongoDB : ' . $e-
>getMessage() . '</error>');
            return Command::FAILURE;
        }

        return Command::SUCCESS;
    }
}
```

Configuration pour que ça fonctionne
Dans config/services.yaml :

```
    App\Command\TestMongoConnectionCommand:
        arguments:
            $dbName: '%env(MONGODB_DB)%'
            $collectionName: '%env(MONGODB_COLLECTION)%'
```

Lancer le test en SSH sur Hostinger

```
$  php bin/console app:test:mongo
```

B. Création d'un service MongoLogService pour un test

i. src/Service/MongoLogService.php

```php
<?php

namespace App\Service;

use MongoDB\Client;
use MongoDB\Collection;
use MongoDB\BSON\UTCDateTime;
use App\Entity\User;

final class MongoLogService
{
    private Collection $collection;

    public function __construct(
        private Client $client,
        private string $dbName,
        private string $collectionName
    ) {
        $this->collection = $this->client->selectCollection($this->dbName, $this->collectionName);
    }

    public function log(string $type, array $payload = [], ?User $user = null): void
    {
        $this->collection->insertOne([
            'type'      => $type,
            'payload'   => $payload,
            'userId'    => $user?->getId(),
            'userEmail' => $user?->getEmail(),
            'createdAt' => new UTCDateTime((new \DateTimeImmutable())->getTimestamp() * 1000),
        ]);
    }

    public function recent(int $limit = 20): array
    {
        // On récupère directement un tableau au lieu de manipuler un Cursor
        $docs = iterator_to_array(
            $this->collection->find([], [
                'sort'  => ['createdAt' => -1],
                'limit' => $limit,
            ])
        );

        $out = [];
        foreach ($docs as $doc) {
            $out[] = [
```

```
                'type'      => $doc['type'] ?? null,
                'payload'   => $doc['payload'] ?? [],
                'userId'    => $doc['userId'] ?? null,
                'userEmail' => $doc['userEmail'] ?? null,
                'createdAt' => isset($doc['createdAt'])
                    ? $doc['createdAt']->toDateTime()->format('Y-m-d H:i:s')
                    : null,
                '_id'       => (string)($doc['_id'] ?? ''),
            ];
        }

        return $out;
    }
}
```

ii. Contrôleur de test rapide

```
<?php

namespace App\Controller;

use App\Service\MongoLogService;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class MongoTestController extends AbstractController
{
    #[Route('/mongo-test', name: 'app_mongo_test')]
    public function index(MongoLogService $mongoLogService): Response
    {
        // Écriture d'un log de test
        $mongoLogService->log(
            'test',
            ['message' => 'Ceci est un test depuis MongoTestController'],
            $this->getUser()
        );

        // Lecture des derniers logs
        $logs = $mongoLogService->recent(10);

        return $this->render('mongo_test/index.html.twig', [
            'logs' => $logs,
        ]);
    }
}
```

iii. templates/mongo_test/index.html.twig

```
{% extends 'base.html.twig' %}

{% block title %}Test MongoDB{% endblock %}

{% block body %}
<div class="container mt-4">
    <h1 class="mb-4">📜 Derniers logs MongoDB</h1>

    {% if logs is not empty %}
        <table class="table table-bordered table-striped">
            <thead class="table-dark">
                <tr>
                    <th>Type</th>
                    <th>Payload</th>
                    <th>User ID</th>
                    <th>User Email</th>
                    <th>Créé le</th>
                    <th>ID Mongo</th>
                </tr>
            </thead>
            <tbody>
                {% for log in logs %}
                    <tr>
                        <td>{{ log.type }}</td>
                        <td>
                            <pre class="mb-0">{{
log.payload|json_encode(constant('JSON_PRETTY_PRINT')) }}</pre>
                        </td>
                        <td>{{ log.userId ?? '-' }}</td>
                        <td>{{ log.userEmail ?? '-' }}</td>
                        <td>{{ log.createdAt ?? '-' }}</td>
                        <td>{{ log._id }}</td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    {% else %}
        <div class="alert alert-warning">Aucun log trouvé.</div>
    {% endif %}
</div>
{% endblock %}
```

On va ajouter une **route pour vider la collection MongoDB** et un bouton dans la vue Twig.

1️⃣ Contrôleur avec suppression

```
<?php

namespace App\Controller;
```

```php
use App\Service\MongoLogService;
use MongoDB\Collection;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class MongoTestController extends AbstractController
{
    #[Route('/mongo-test', name: 'app_mongo_test')]
    public function index(MongoLogService $mongoLogService): Response
    {
        // Écriture d'un log de test
        $mongoLogService->log(
            'test',
            ['message' => 'Ceci est un test depuis MongoTestController'],
            $this->getUser()
        );

        // Lecture des derniers logs
        $logs = $mongoLogService->recent(10);

        return $this->render('mongo_test/index.html.twig', [
            'logs' => $logs,
        ]);
    }

    #[Route('/mongo-test/clear', name: 'app_mongo_test_clear')]
    public function clear(MongoLogService $mongoLogService): RedirectResponse
    {
        // On supprime tous les documents de la collection
        $mongoLogService->clearAll();

        $this->addFlash('success', 'Tous les logs ont été supprimés.');
        return $this->redirectToRoute('app_mongo_test');
    }
}
```

2 Ajout de la méthode clearAll() dans MongoLogService

```php
    public function clearAll(): void
    {
        $this->collection->deleteMany([]);
    }
```

3 Mise à jour de la vue templates/mongo_test/index.html.twig

```twig
{% extends 'base.html.twig' %}

{% block title %}Test MongoDB
{% endblock %}
```

```
{% block body %}
    <div class="container mt-4">
        <h1 class="mb-4">📜 Derniers logs MongoDB</h1>

        <div class="mb-3">
            <a href="{{ path('app_mongo_test_clear') }}" class="btn btn-danger"
onclick="return confirm('Voulez-vous vraiment effacer tous les logs ?')">
                🗑️ Vider les logs
            </a>
        </div>

        {% for label, messages in app.flashes %}
            {% for message in messages %}
                <div class="alert alert-{{ label }}">{{ message }}</div>
            {% endfor %}
        {% endfor %}

        {% if logs is not empty %}
            <table class="table table-bordered table-striped">
                <thead class="table-dark">
                    <tr>
                        <th>Type</th>
                        <th>Payload</th>
                        <th>User ID</th>
                        <th>User Email</th>
                        <th>Créé le</th>
                        <th>ID Mongo</th>
                    </tr>
                </thead>
                <tbody>
                    {% for log in logs %}
                        <tr>
                            <td>{{ log.type }}</td>
                            <td>
                                <pre class="mb-0">{{
log.payload|json_encode(constant('JSON_PRETTY_PRINT')) }}</pre>
                            </td>
                            <td>{{ log.userId ?? '-' }}</td>
                            <td>{{ log.userEmail ?? '-' }}</td>
                            <td>{{ log.createdAt ?? '-' }}</td>
                            <td>{{ log._id }}</td>
                        </tr>
                    {% endfor %}
                </tbody>
            </table>
        {% else %}
            <div class="alert alert-warning">Aucun log trouvé.</div>
        {% endif %}
    </div>
{% endblock %}
```

On peut enchaîner sur le petit **bouton "Ajouter un log de test"** pour rendre la page 100 % interactive.

1️⃣ Contrôleur mis à jour

```php
<?php

namespace App\Controller;

use App\Service\MongoLogService;
use MongoDB\Collection;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\RedirectResponse;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class MongoTestController extends AbstractController
{
    #[Route('/mongo-test', name: 'app_mongo_test')]
    public function index(MongoLogService $mongoLogService): Response
    {
        // Écriture d'un log de test
        // $mongoLogService->log(
        //     'test',
        //     ['message' => 'Ceci est un test depuis MongoTestController'],
        //     $this->getUser()
        // );

        // Lecture des derniers logs
        $logs = $mongoLogService->recent(10);

        return $this->render('mongo_test/index.html.twig', [
            'logs' => $logs,
        ]);
    }

    #[Route('/mongo-test/add', name: 'app_mongo_test_add')]
    public function add(MongoLogService $mongoLogService): RedirectResponse
    {
        $mongoLogService->log(
            'test',
            ['message' => 'Ajout manuel depuis le bouton'],
            $this->getUser()
        );

        $this->addFlash('success', 'Log ajouté avec succès.');
        return $this->redirectToRoute('app_mongo_test');
    }

    #[Route('/mongo-test/clear', name: 'app_mongo_test_clear')]
    public function clear(MongoLogService $mongoLogService): RedirectResponse
    {
```

```
        // On supprime tous les documents de la collection
        $mongoLogService->clearAll();

        $this->addFlash('success', 'Tous les logs ont été supprimés.');
        return $this->redirectToRoute('app_mongo_test');
    }
}
```

2️⃣ Vue Twig mise à jour

```
{% extends 'base.html.twig' %}

{% block title %}Test MongoDB{% endblock %}

{% block body %}
<div class="container mt-4">
    <h1 class="mb-4">📜 Derniers logs MongoDB</h1>

    <div class="mb-3">
        <a href="{{ path('app_mongo_test_add') }}" class="btn btn-primary">
            ➕ Ajouter un log de test
        </a>
        <a href="{{ path('app_mongo_test_clear') }}" class="btn btn-danger"
           onclick="return confirm('Voulez-vous vraiment effacer tous les logs ?')">
            🗑️ Vider les logs
        </a>
    </div>

    {% for label, messages in app.flashes %}
        {% for message in messages %}
            <div class="alert alert-{{ label }}">{{ message }}</div>
        {% endfor %}
    {% endfor %}

    {% if logs is not empty %}
        <table class="table table-bordered table-striped">
            <thead class="table-dark">
                <tr>
                    <th>Type</th>
                    <th>Payload</th>
                    <th>User ID</th>
                    <th>User Email</th>
                    <th>Créé le</th>
                    <th>ID Mongo</th>
                </tr>
            </thead>
            <tbody>
                {% for log in logs %}
                    <tr>
                        <td>{{ log.type }}</td>
                        <td>
```

```
                            <pre class="mb-0">{{
log.payload|json_encode(constant('JSON_PRETTY_PRINT')) }}</pre>
                        </td>
                        <td>{{ log.userId ?? '-' }}</td>
                        <td>{{ log.userEmail ?? '-' }}</td>
                        <td>{{ log.createdAt ?? '-' }}</td>
                        <td>{{ log._id }}</td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    {% else %}
        <div class="alert alert-warning">Aucun log trouvé.</div>
    {% endif %}
</div>
{% endblock %}
```

Avec cette version :

- **/mongo-test** → affiche uniquement les logs (aucune écriture).
- **/mongo-test/add** → ajoute un log puis redirige vers la liste.
- **/mongo-test/clear** → supprime tous les logs puis redirige vers la liste.

C. Intégration de MongoDB dans l'application Symfony

i. Modification de TripReservationController

On va se placer dans le **TripReservationController**, juste après qu'une réservation est validée, pour appeler MongoLogService.

```
    #[Route('/{id}-{slug}/reservation/book', name: 'app_trip_reservation_book', methods:
['POST'])]
    public function book(Trip $trip, EntityManagerInterface $em, TripReservationValidator
$validator, MongoLogService $mongoLogService): JsonResponse
    {
        /** @var User $user */
        $user = $this->getUser();
        // Utilisation du service de validation
        [$error, $code] = $validator->validate($trip, $user);
        if ($error) {
            return $this->json(['error' => $error], $code);
        }

        // Vérifie que le passager n'est pas déjà inscrit
        foreach ($trip->getTripPassengers() as $tp) {
            if ($tp->getUser() === $user) {
                return $this->json(['error' => 'Vous êtes déjà inscrit sur ce trajet.'],
400);
            }
```

```
        }

        $user->setCredit($user->getCredit() - $trip->getPricePerPerson());

        $tripPassenger = new TripPassenger();
        $tripPassenger->setTrip($trip)
            ->setUser($user)
            ->setValidationStatus('pending');
        $em->persist($tripPassenger);

        $em->persist($user);
        $em->persist($trip);
        $em->flush();

        // ☑ LOG dans MongoDB
        $mongoLogService->log(
            'trip.reservation',
            [
                'tripId'       => $trip->getId(),
                'driverId'     => $trip->getDriver()->getId(),
                'driverPseudo' => $trip->getDriver()->getPseudo(),
                'price'        => $trip->getPricePerPerson(),
                'seatsBooked'  => 1, // tu peux mettre la vraie valeur si multi-places
                'departure'    => $trip->getDepartureAddress(),
                'arrival'      => $trip->getArrivalAddress(),
            ],
            $user
        );

        $this->addFlash('success', 'Votre réservation a bien été enregistrée ! Vous serez débité uniquement si le trajet n\'est pas annulé.');

        return $this->json(['success' => true]);
    }
```

ii. Explications des étapes :

1. **Récupération de l'utilisateur**

```
        /** @var User $user */
        $user = $this->getUser();
```

On récupère l'utilisateur actuellement connecté, nécessaire pour toutes les validations.

2. **Validation métier via un service dédié**

```
        [$error, $code] = $validator->validate($trip, $user);
        if ($error) {
            return $this->json(['error' => $error], $code);
        }
```

- Le service TripReservationValidator centralise les règles métier (crédits, disponibilités, etc.).

- Si une erreur est détectée, la méthode renvoie une réponse JSON avec le message et le code HTTP approprié.

3. **Vérification que l'utilisateur n'est pas déjà inscrit**

```
        foreach ($trip->getTripPassengers() as $tp) {
            if ($tp->getUser() === $user) {
                return $this->json(['error' => 'Vous êtes déjà inscrit sur ce trajet.'],
400);
            }
        }
```

- On parcourt la liste des passagers déjà inscrits sur le trajet.
- Si l'utilisateur est déjà présent, on bloque la réservation.

4. **Mise à jour des crédits de l'utilisateur**

```
$user->setCredit($user->getCredit() - $trip->getPricePerPerson());
```

- On retire le montant du prix par personne du solde de crédits de l'utilisateur.

5. **Création de l'inscription (TripPassenger)**

```
        $tripPassenger = new TripPassenger();
        $tripPassenger->setTrip($trip)
            ->setUser($user)
            ->setValidationStatus('pending');
        $em->persist($tripPassenger);
```

- Un nouvel objet TripPassenger est créé pour lier l'utilisateur au trajet.
- Le statut est mis sur "pending" (en attente de validation).

6. **Enregistrement en base SQL**

```
        $em->persist($user);
        $em->persist($trip);
        $em->flush();
```

- On persiste les modifications sur l'utilisateur, le trajet et le passager.
- flush() écrit toutes ces données dans la base **relationnelle** (MySQL).

7. **Log dans MongoDB**

```
        $mongoLogService->log(
            'trip.reservation',
            [
                'tripId'       => $trip->getId(),
                'driverId'     => $trip->getDriver()->getId(),
                'driverPseudo' => $trip->getDriver()->getPseudo(),
                'price'        => $trip->getPricePerPerson(),
                'seatsBooked'  => 1, // tu peux mettre la vraie valeur si multi-places
                'departure'    => $trip->getDepartureAddress(),
                'arrival'      => $trip->getArrivalAddress(),
            ],
            $user
        );
```

- Utilisation de la méthode log() du MongoLogService pour créer un document dans MongoDB.

- Le log contient toutes les infos importantes : trajet, conducteur, prix, places réservées, adresse de départ/arrivée, utilisateur.

8. **Message de confirmation**

```
        $this->addFlash('success', 'Votre réservation a bien été enregistrée ! Vous serez débité uniquement si le trajet n\'est pas annulé.');
```

- Ajout d'un message flash qui sera affiché à l'utilisateur.

9. **Réponse JSON**

```
        return $this->json(['success' => true]);
```

- La méthode renvoie un JSON confirmant le succès de la réservation.

Points importants :

- **Utilisation combinée SQL + NoSQL** :
  - SQL (MySQL) pour stocker la réservation de manière transactionnelle.
  - NoSQL (MongoDB) pour historiser l'événement et l'exploiter à des fins statistiques.
- **Architecture claire** :
  - Validation métier déléguée à un service (TripReservationValidator).
  - Persistance gérée par Doctrine (EntityManagerInterface).
  - Historisation déléguée au MongoLogService.

iii. Rôle du service MongoLogService

Le service **MongoLogService** est un composant dédié à la communication avec la base de données NoSQL MongoDB. Il permet de centraliser toutes les opérations de lecture et d'écriture vers MongoDB, garantissant ainsi un code plus clair et une maintenance facilitée.

iv. Méthode getTripReservationsWithStats dans le service

La méthode **getTripReservationsWithStats** a pour but de récupérer la liste des réservations de trajets stockées dans MongoDB, avec la possibilité de filtrer les résultats et de calculer des statistiques pertinentes.

```
    public function getTripReservationsWithStats(
        int $limit = 50,
        ?\DateTimeInterface $startDate = null,
        ?\DateTimeInterface $endDate = null,
        ?int $driverId = null
    ): array {
        $filter = ['type' => 'trip.reservation'];

        if ($startDate && $endDate) {
            $start = (new \DateTimeImmutable($startDate->format('Y-m-d')))->setTime(0, 0, 0);
            $end = (new \DateTimeImmutable($endDate->format('Y-m-d')))->setTime(23, 59, 59);
```

```
            $filter['createdAt'] = [
                '$gte' => new \MongoDB\BSON\UTCDateTime($start->getTimestamp() * 1000),
                '$lte' => new \MongoDB\BSON\UTCDateTime($end->getTimestamp() * 1000),
            ];
        } elseif ($startDate) {
            $start = (new \DateTimeImmutable($startDate->format('Y-m-d')))->setTime(0, 0,
0);
            $filter['createdAt'] = [
                '$gte' => new \MongoDB\BSON\UTCDateTime($start->getTimestamp() * 1000),
            ];
        } elseif ($endDate) {
            $end = (new \DateTimeImmutable($endDate->format('Y-m-d')))->setTime(23, 59,
59);
            $filter['createdAt'] = [
                '$lte' => new \MongoDB\BSON\UTCDateTime($end->getTimestamp() * 1000),
            ];
        }

        if ($driverId !== null) {
            $filter['payload.driverId'] = $driverId;
        }

        $cursor = $this->collection->find(
            $filter,
            [
                'sort'  => ['createdAt' => -1],
                'limit' => $limit,
            ]
        );

        $docs = iterator_to_array($cursor);

        $logs = [];
        $totalCredits = 0;
        foreach ($docs as $doc) {
            $price = $doc['payload']['price'] ?? 0;
            $totalCredits += $price;

            $logs[] = [
                'tripId'       => $doc['payload']['tripId'] ?? null,
                'driverId'     => $doc['payload']['driverId'] ?? null,
                'driverPseudo' => $doc['payload']['driverPseudo'] ?? null,
                'price'        => $price,
                'seatsBooked'  => $doc['payload']['seatsBooked'] ?? null,
                'departure'    => $doc['payload']['departure'] ?? null,
                'arrival'      => $doc['payload']['arrival'] ?? null,
                'userId'       => $doc['userId'] ?? null,
                'userEmail'    => $doc['userEmail'] ?? null,
                'createdAt'    => isset($doc['createdAt'])
                    ? $doc['createdAt']->toDateTime()->format('Y-m-d H:i:s')
                    : null,
                '_id'          => (string)($doc['_id'] ?? ''),
```

```
            ];
        }

        return [
            'logs' => $logs,
            'totalCredits' => $totalCredits,
            'platformCredits' => count($logs) * 2
        ];
    }
```

1. **Paramètres**

   - limit (int) : nombre maximum de résultats à afficher (par défaut 50).
   - startDate (DateTimeInterface|null) : date de début de l'intervalle de recherche.
   - endDate (DateTimeInterface|null) : date de fin de l'intervalle de recherche.
   - driverId (int|null) : identifiant du conducteur pour filtrer les résultats.

2. **Construction du filtre MongoDB**

   - En fonction des paramètres fournis, la méthode construit un tableau de filtres compatible avec MongoDB :
   - Si startDate et endDate sont renseignées, elles sont converties en UTCDateTime et utilisées avec les opérateurs $gte et $lte pour filtrer les documents sur cet intervalle.
   - Si seulement startDate ou endDate est fournie, un filtre unilatéral est appliqué.
   - Si driverId est renseigné, un filtre supplémentaire sur payload.driverId est ajouté.

3. **Récupération des données**

Le service exécute la requête avec $this->collection->find(), triée par createdAt en ordre décroissant, et limite le nombre de résultats à la valeur définie.

4. **Calcul des statistiques**

Pour chaque document retourné, la méthode :

   - Ajoute le prix à un compteur totalCredits (somme des crédits réservés).
   - Calcule la part de la plateforme en multipliant le nombre de réservations par 2 crédits.

5. **Résultat retourné**

La méthode retourne un tableau associatif contenant :

   - logs : liste des réservations avec leurs détails (trajet, conducteur, utilisateur, date...). totalCredits : somme totale des crédits réservés.
   - platformCredits : part totale revenant à la plateforme.

v. Contrôleur TripLogController

Le contrôleur **TripLogController** est chargé de récupérer les filtres (dates, conducteur) depuis la requête HTTP, de les convertir au bon format et de les transmettre à **MongoLogService**.

Il appelle la méthode **getTripReservationsWithStats()** pour obtenir la liste filtrée et les statistiques, puis transmet ces données à la vue Twig, où elles sont affichées sous forme de tableau et de résumé statistique.

```php
<?php

namespace App\Controller\Trip;

use App\Service\MongoLogService;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class TripLogController extends AbstractController
{
    #[Route('/admin/trip-logs', name: 'app_trip_logs')]
    public function index(Request $request, MongoLogService $mongoLogService): Response
    {
        $startDateStr = $request->query->get('startDate');
        $endDateStr = $request->query->get('endDate');

        $startDate = $startDateStr ? \DateTimeImmutable::createFromFormat('Y-m-d',
$startDateStr) : null;
        $endDate   = $endDateStr ? \DateTimeImmutable::createFromFormat('Y-m-d',
$endDateStr) : null;

        $driverIdParam = $request->query->get('driverId');
        $driverId = is_numeric($driverIdParam) ? (int)$driverIdParam : null;

        $result = $mongoLogService->getTripReservationsWithStats(50, $startDate, $endDate,
$driverId);

        return $this->render('trip/logs.html.twig', [
            'logs' => $result['logs'],
            'totalCredits' => $result['totalCredits'],
            'platformCredits' => $result['platformCredits'],
            'startDate' => $startDateStr,
            'endDate' => $endDateStr,
            'driverId' => $driverId,
        ]);
    }
}
```

vi. Lien avec la logique métier

Chaque fois qu'une réservation est validée dans l'application, un enregistrement est ajouté dans MongoDB via **MongoLogService->log()** dans le **TripReservationController**. Cela permet de conserver un historique détaillé des événements de réservation.
Ces données peuvent ensuite être consultées via **TripLogController**, filtrées et analysées pour extraire des informations utiles comme le volume d'activité ou les gains de la plateforme.

vii. Schéma simplifié du flux

1. L'utilisateur confirme une réservation.
2. Le contrôleur de réservation appelle **MongoLogService->log()** pour enregistrer l'événement.
3. **TripLogController** interroge MongoDB via **getTripReservationsWithStats()**.
4. Les données et statistiques sont envoyées à la vue Twig.
5. L'interface affiche les résultats et les statistiques à l'utilisateur.

viii. Conclusion

Cette intégration démontre la capacité à utiliser conjointement une base SQL pour le stockage transactionnel et une base NoSQL pour le suivi et l'analyse des événements. Elle répond pleinement aux exigences en matière d'utilisation des deux types de bases de données.