

# Créer une API web avec ASP.NET Core et MongoDB

17/08/2019 • 18 minutes de lecture • 

## Dans cet article

[Configuration requise](#)

[Configurer MongoDB](#)

[Créer le projet d'API web ASP.NET Core](#)

[Ajouter un modèle d'entité](#)

[Ajouter un modèle de configuration](#)

[Ajouter un service d'opérations CRUD](#)

[Ajouter un contrôleur](#)

[Tester l'API web](#)

[Configurer les options de sérialisation JSON](#)

[Ajouter la prise en charge de l'authentification à une API Web](#)

[Étapes suivantes](#)

Par [Pratik Khandelwal](#) et [Scott Addie](#)

Ce didacticiel crée une API web qui effectue des opérations de création, lecture, mise à jour et suppression (CRUD) sur une base de données NoSQL [MongoDB](#).

Dans ce didacticiel, vous apprendrez à :

- ✓ Configurer MongoDB
- ✓ Créer une base de données MongoDB
- ✓ Définir une collection et un schéma MongoDB
- ✓ Effectuer des opérations CRUD MongoDB à partir d'une API web
- ✓ Personnaliser la sérialisation JSON

[Affichez ou téléchargez l'exemple de code](#) ([procédure de téléchargement](#))

## Configuration requise

Visual Studio

Visual Studio Code

Visual Studio pour Mac

- [SDK .NET Core 3.0 ou version ultérieure](#)


- [Visual Studio 2019](#) avec la charge de travail de **développement web et ASP.NET**
- [MongoDB](#)

## Configurer MongoDB


Si vous utilisez Windows, MongoDB est installé par défaut dans *C:\Program Files\MongoDB*. Ajoutez *C:\Program Files\MongoDB\Server\<numéro\_version>\bin* à la variable d'environnement *Path*. Cette modification permet d'accéder à MongoDB depuis n'importe quel emplacement sur votre ordinateur de développement.

Utilisez l'interpréteur de commandes mongo dans les étapes suivantes pour créer une base de données, des collections, et stocker des documents. Pour plus d'informations sur les commandes de l'interpréteur mongo, consultez [Utilisation de l'interpréteur de commandes mongo](#).


1. Choisissez sur votre ordinateur de développement un répertoire pour le stockage des données. Par exemple, *C:\BooksData* sous Windows. Le cas échéant, créez le répertoire. L'interpréteur de commandes mongo ne crée pas nouveaux répertoires.
2. Ouvrez un interpréteur de commandes. Exécutez la commande suivante pour vous connecter à MongoDB sur le port par défaut 27017. N'oubliez pas de remplacer *<data\_directory\_path>* par le répertoire que vous avez choisi à l'étape précédente.

console	
<pre>mongod --dbpath &lt;data_directory_path&gt;</pre>	

3. Ouvrez une autre instance de l'interpréteur de commandes. Connectez-vous à la base de données de test par défaut en exécutant la commande suivante :


console	
<pre>mongo</pre>	

4. Utilisez la ligne suivante dans l'interpréteur de commandes :


console	
<pre>use BookstoreDb</pre>	

Le cas échéant, une base de données nommée *BookstoreDb* est créée. Si la base de données existe déjà, sa connexion est ouverte pour les transactions.


5. Créez une collection `Books` à l'aide de la commande suivante :

console	 Copier
<pre>db.createCollection('Books')</pre>	


Le résultat suivant s'affiche :

console	 Copier
<pre>{ "ok" : 1 }</pre>	

6. Définissez un schéma pour la collection `Books` et insérez deux documents à l'aide de la commande suivante :

console	 Copier
<pre>db.Books.insertMany([{'Name':'Design Patterns','Price':54.93,'Category':'Computers','Author':'Ralph Johnson'}, {'Name':'Clean Code','Price':43.15,'Category':'Computers','Author':'Robert C. Martin'}])</pre>	

Le résultat suivant s'affiche :

console	 Copier
<pre>{   "acknowledged" : true,   "insertedIds" : [     ObjectId("5bfd996f7b8e48dc15ff215d"),     ObjectId("5bfd996f7b8e48dc15ff215e")   ] }</pre>	

#### Notes

Les ID indiqués dans cet article ne correspondent pas aux ID obtenus quand vous exécutez cet exemple.

7. Affichez les documents de la base de données à l'aide de la commande suivante :

console

 Copier

```
db.Books.find({}).pretty()
```

Le résultat suivant s'affiche :

console

 Copier

```
{
  "_id" : ObjectId("5bfd996f7b8e48dc15ff215d"),
  "Name" : "Design Patterns",
  "Price" : 54.93,
  "Category" : "Computers",
  "Author" : "Ralph Johnson"
}
{
  "_id" : ObjectId("5bfd996f7b8e48dc15ff215e"),
  "Name" : "Clean Code",
  "Price" : 43.15,
  "Category" : "Computers",
  "Author" : "Robert C. Martin"
}
```

Le schéma ajoute une propriété `_id` automatiquement générée de type `ObjectId` pour chaque document.

La base de données est en lecture seule. Vous pouvez commencer à créer l'API web ASP.NET Core.

## Créer le projet d'API web ASP.NET Core

Visual Studio

Visual Studio Code

Visual Studio pour Mac

1. Accédez à **Fichier > Nouveau > Projet**.
2. Sélectionnez le type de projet **Application web ASP.NET Core**, puis sélectionnez **Suivant**.
3. Nommez le projet *BooksApi*, puis sélectionnez **Créer**.
4. Sélectionnez le framework cible **.NET Core** et **ASP.NET Core 3.0**. Sélectionnez le modèle de projet **API**, puis sélectionnez **Créer**.
5. Visitez la [galerie NuGet : MongoDB. Driver](https://www.nuget.org/packages/MongoDB.Driver) pour déterminer la dernière version stable du pilote .net pour MongoDB. Dans la fenêtre **Console du**

**Gestionnaire de Package**, accédez à la racine du projet. Exécutez la commande suivante afin d'installer le pilote .NET pour MongoDB :

PowerShell

 Copier

```
Install-Package MongoDB.Driver -Version {VERSION}
```

## Ajouter un modèle d'entité

1. Ajoutez un répertoire *Models* à la racine du projet.
2. Ajoutez une classe *Book* au répertoire *Modèles* avec le code suivant :

C#

 Copier

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace BooksApi.Models
{
    public class Book
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string Id { get; set; }

        [BsonElement("Name")]
        public string BookName { get; set; }

        public decimal Price { get; set; }

        public string Category { get; set; }

        public string Author { get; set; }
    }
}
```

Dans la classe précédente, la propriété *Id* :


- Est requise pour mapper l'objet Common Language Runtime (CLR) à la collection MongoDB.
- Est annotée avec `[BsonId]` pour désigner cette propriété en tant que clé primaire du document.
- Est annotée avec `[BsonRepresentation(BsonType.ObjectId)]` pour autoriser le passage du paramètre en tant que type `string` à la place d'une structure

[ObjectId](#). Mongo gère la conversion de `string` en `ObjectId`.

La propriété `BookName` est annotée avec l'attribut [\[BsonElement\]](#). La valeur de l'attribut de `Name` représente le nom de propriété dans la collection MongoDB.


## Ajouter un modèle de configuration

1. Ajoutez les valeurs de configuration de base de données suivantes à `appsettings.json` :

JSON 

```
{
  "BookstoreDatabaseSettings": {
    "BooksCollectionName": "Books",
    "ConnectionString": "mongodb://localhost:27017",
    "DatabaseName": "BookstoreDb"
  },
  "Logging": {
    "IncludeScopes": false,
    "Debug": {
      "LogLevel": {
        "Default": "Warning"
      }
    },
    "Console": {
      "LogLevel": {
        "Default": "Warning"
      }
    }
  }
}
```

2. Ajoutez un fichier `BookstoreDatabaseSettings.cs` au répertoire `Models` avec le code suivant :

C# 


```
namespace BooksApi.Models
{
    public class BookstoreDatabaseSettings : IBookstoreDatabaseSettings
    {
        public string BooksCollectionName { get; set; }
        public string ConnectionString { get; set; }
        public string DatabaseName { get; set; }
    }

    public interface IBookstoreDatabaseSettings
    {
    }
}
```

```
{
    string BooksCollectionName { get; set; }
    string ConnectionString { get; set; }
    string DatabaseName { get; set; }
}
```

La classe `BookstoreDatabaseSettings` précédente est utilisée pour stocker les valeurs de propriété `BookstoreDatabaseSettings` du fichier `appsettings.json`. Les noms de propriétés JSON et C# sont nommés de manière identique pour faciliter le processus de mappage.

3. Ajoutez le code en surbrillance suivant à `Startup.ConfigureServices` :

C#	
<pre>public void ConfigureServices(IServiceCollection services) {     // requires using Microsoft.Extensions.Options     services.Configure&lt;BookstoreDatabaseSettings&gt;(         Configuration.GetSection(nameof(BookstoreDatabaseSettings)));      services.AddSingleton&lt;IBookstoreDatabaseSettings&gt;(sp =&gt;         sp.GetRequiredService&lt;IOptions&lt;BookstoreDatabaseSettings&gt;&gt;         ().Value);      services.AddControllers(); }</pre>	

Dans le code précédent :

- L'instance de configuration à laquelle la section `BookstoreDatabaseSettings` du fichier `appsettings.json` est liée est inscrite dans le conteneur d'injection de dépendances. Par exemple, la propriété `ConnectionString` d'un objet `BookstoreDatabaseSettings` est peuplée avec la propriété `BookstoreDatabaseSettings:ConnectionString` dans `appsettings.json`.
- L'interface `IBookstoreDatabaseSettings` est inscrite auprès de l'injection de dépendances avec une [durée de vie de service](#) de singleton. Une fois injectée, l'instance d'interface est résolue en objet `BookstoreDatabaseSettings`.

4. Ajoutez le code suivant en haut de `Startup.cs` pour résoudre les références à `BookstoreDatabaseSettings` et `IBookstoreDatabaseSettings` :

C#	

```
using BooksApi.Models;
```

## Ajouter un service d'opérations CRUD

1. Ajoutez un répertoire *Services* à la racine du projet.
2. Ajoutez une classe *BookService* au répertoire *Services* avec le code suivant :


```
C# Copier  
  
using BooksApi.Models;  
using MongoDB.Driver;  
using System.Collections.Generic;  
using System.Linq;  
  
namespace BooksApi.Services  
{  
    public class BookService  
    {  
        private readonly IMongoCollection<Book> _books;  
  
        public BookService(IBookstoreDatabaseSettings settings)  
        {  
            var client = new MongoClient(settings.ConnectionString);  
            var database = client.GetDatabase(settings.DatabaseName);  
  
            _books = database.GetCollection<Book>  
(settings.BooksCollectionName);  
        }  
  
        public List<Book> Get() =>  
            _books.Find(book => true).ToList();  
  
        public Book Get(string id) =>  
            _books.Find<Book>(book => book.Id == id).FirstOrDefault();  
  
        public Book Create(Book book)  
        {  
            _books.InsertOne(book);  
            return book;  
        }  
  
        public void Update(string id, Book bookIn) =>  
            _books.ReplaceOne(book => book.Id == id, bookIn);  
  
        public void Remove(Book bookIn) =>  
            _books.DeleteOne(book => book.Id == bookIn.Id);  
  
        public void Remove(string id) =>  
            _books.DeleteOne(book => book.Id == id);  
    }  
}
```



```
}  
}
```


Dans le code précédent, une instance de `IBookstoreDatabaseSettings` est récupérée à partir de l'injection de dépendances via l'injection de constructeur. Cette technique permet d'accéder aux valeurs de configuration d'`appsettings.json`, qui ont été ajoutées à la section [Ajouter un modèle de configuration](#).

3. Ajoutez le code en surbrillance suivant à `Startup.ConfigureServices` :

C#	
<pre>public void ConfigureServices(IServiceCollection services) {     services.Configure&lt;BookstoreDatabaseSettings&gt;(Configuration.GetSection(nameof(BookstoreDatabaseSettings)));      services.AddSingleton&lt;IBookstoreDatabaseSettings&gt;(sp =&gt;         sp.GetRequiredService&lt;IOptions&lt;BookstoreDatabaseSettings&gt;&gt;().Value);      services.AddSingleton&lt;BookService&gt;();      services.AddControllers(); }</pre>	


Dans le code précédent, la classe `BookService` est inscrite auprès de l'injection de dépendances pour permettre la prise en charge de l'injection de constructeur dans les classes consommatrices. La durée de vie de service de singleton est la plus appropriée, car `BookService` a une dépendance directe sur `MongoClient`. Selon les [recommandations officielles de réutilisation de Mongo Client](#), `MongoClient` doit être inscrit auprès de l'injection de dépendances avec une durée de vie de service de singleton.

4. Ajoutez le code suivant en haut de `Startup.cs` pour résoudre la référence à `BookService` :

C#	
<pre>using BooksApi.Services;</pre>	

La classe `BookService` utilise les membres `MongoDB.Driver` suivants pour effectuer des opérations CRUD dans la base de données :

- [MongoClient](#) – Lit l'instance de serveur qui permet d'effectuer des opérations de base de données. Le constructeur de cette classe reçoit la chaîne de connexion MongoDB :

C#	 Copier
<pre>public BookService(IBookstoreDatabaseSettings settings) {     var client = new MongoClient(settings.ConnectionString);     var database = client.GetDatabase(settings.DatabaseName);      _books = database.GetCollection&lt;Book&gt; (settings.BooksCollectionName); }</pre>	


- [IMongoDatabase](#) – Représente la base de données Mongo qui permet d'effectuer des opérations. Ce tutoriel utilise la méthode générique [GetCollection<TDocument>\(collection\)](#) sur l'interface pour accéder aux données d'une collection spécifique. Effectuez des opérations CRUD sur la collection, après l'appel de cette méthode. Dans l'appel à la méthode `GetCollection<TDocument>(collection)` :
  - `collection` représente le nom de la collection.
  - `TDocument` représente le type d'objet CLR stocké dans la collection.

`GetCollection<TDocument>(collection)` retourne un objet [MongoCollection](#) représentant la collection. Dans ce didacticiel, les méthodes suivantes sont appelées sur la collection :

- [DeleteOne](#) – Supprime un seul document correspondant aux critères de recherche fournis.
- [Find<TDocument>](#) – Retourne tous les documents de la collection correspondant aux critères de recherche fournis.
- [InsertOne](#) – Insère l'objet fourni en tant que nouveau document dans la collection.
- [ReplaceOne](#) – Remplace le seul document correspondant aux critères de recherche fournis par l'objet indiqué.

## Ajouter un contrôleur

Ajoutez une classe `BooksController` au répertoire *Controllers* avec le code suivant :

C#	 Copier
<pre>using BooksApi.Models; using BooksApi.Services;</pre>	

```
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace BooksApi.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class BooksController : ControllerBase
    {
        private readonly BookService _bookService;

        public BooksController(BookService bookService)
        {
            _bookService = bookService;
        }

        [HttpGet]
        public ActionResult<List<Book>> Get() =>
            _bookService.Get();

        [HttpGet("{id:length(24)}", Name = "GetBook")]
        public ActionResult<Book> Get(string id)
        {
            var book = _bookService.Get(id);

            if (book == null)
            {
                return NotFound();
            }

            return book;
        }

        [HttpPost]
        public ActionResult<Book> Create(Book book)
        {
            _bookService.Create(book);

            return CreatedAtRoute("GetBook", new { id = book.Id.ToString()
            }, book);
        }

        [HttpPut("{id:length(24)}")]
        public IActionResult Update(string id, Book bookIn)
        {
            var book = _bookService.Get(id);

            if (book == null)
            {
                return NotFound();
            }

            _bookService.Update(id, bookIn);

            return NoContent();
        }
    }
}
```

```
    }

    [HttpDelete("{id:length(24)}")]
    public IActionResult Delete(string id)
    {
        var book = _bookService.Get(id);

        if (book == null)
        {
            return NotFound();
        }

        _bookService.Remove(book.Id);

        return NoContent();
    }
}
```

Le contrôleur d'API web précédent :

- Utilise la classe `BookService` pour effectuer des opérations CRUD.
- Contient des méthodes d'action pour prendre en charge les requêtes HTTP GET, POST, PUT et DELETE.
- Appelle `CreatedAtRoute` dans la méthode d'action `Create` pour retourner une réponse **HTTP 201**. Le code d'état 201 est la réponse standard d'une méthode HTTP POST qui crée une ressource sur le serveur. `CreatedAtRoute` ajoute également un en-tête `Location` à la réponse. L'en-tête `Location` spécifie l'URI du livre qui vient d'être créé.

## Tester l'API web

1. Générez et exécutez l'application.
2. Accédez à `http://localhost:<port>/api/books` pour tester la méthode d'action `Get` sans paramètre du contrôleur. La réponse JSON suivante apparaît :

JSON	 Copier
<pre>[   {     "id": "5bfd996f7b8e48dc15ff215d",     "bookName": "Design Patterns",     "price": 54.93,     "category": "Computers",     "author": "Ralph Johnson"   },   {</pre>	

```
"id": "5bfd996f7b8e48dc15ff215e",  
"bookName": "Clean Code",  
"price": 43.15,  
"category": "Computers",  
"author": "Robert C. Martin"  
}  
]
```

3. Accédez à `http://localhost:<port>/api/books/{id here}` pour tester la méthode d'action `Get` surchargée du contrôleur. La réponse JSON suivante apparaît :

JSON	 Copier
<pre>{   "id": "{ID}",   "bookName": "Clean Code",   "price": 43.15,   "category": "Computers",   "author": "Robert C. Martin" }</pre>	


## Configurer les options de sérialisation JSON

Vous devez changer deux détails concernant les réponses JSON retournées dans la section [Tester l'API web](#) :

- Vous devez changer la casse mixte par défaut des noms de propriétés pour qu'elle corresponde à la casse Pascal des noms de propriétés de l'objet CLR.
- La propriété `bookName` doit être retournée sous la forme `Name`.

Pour satisfaire les exigences précédentes, apportez les changements suivants :

1. JSON.NET a été supprimé du framework partagé ASP.NET. Ajoutez une référence de package à [Microsoft.AspNetCore.Mvc.NewtonsoftJson](#).
2. Dans `Startup.ConfigureServices`, ajoutez le code en surbrillance suivant à l'appel de méthode `AddMvc` :

C#	 Copier
<pre>public void ConfigureServices(IServiceCollection services) {     services.Configure&lt;BookstoreDatabaseSettings&gt;(         Configuration.GetSection(nameof(BookstoreDatabaseSettings)));      services.AddSingleton&lt;IBookstoreDatabaseSettings&gt;(sp =&gt;         sp.GetRequiredService&lt;IOptions&lt;BookstoreDatabaseSettings&gt;&gt;</pre>	


```
().Value);

services.AddSingleton<BookService>();

services.AddControllers()
    .AddNewtonsoftJson(options => options.UseMemberCasing());
}
```


À la suite du changement effectué, les noms de propriétés de la réponse JSON sérialisée de l'API web correspondent aux noms de propriétés équivalents du type d'objet CLR. Par exemple, la propriété `Author` de la classe `Book` est sérialisée en tant que `Author`.

3. Dans `Models/Book.cs`, annotez la propriété `BookName` avec l'attribut [\[JsonProperty\]](#) suivant :

C#	
<pre>[BsonElement("Name")] [JsonProperty("Name")] public string BookName { get; set; }</pre>	

La valeur de l'attribut `[JsonProperty]` de `Name` représente le nom de propriété dans la réponse JSON sérialisée de l'API web.

4. Ajoutez le code suivant en haut de `Models/Book.cs` pour résoudre la référence d'attribut `[JsonProperty]` :

C#	
<pre>using Newtonsoft.Json;</pre>	

5. Répétez les étapes définies dans la section [Tester l'API web](#). Notez la différence des noms de propriétés JSON.

## Ajouter la prise en charge de l'authentification à une API Web

ASP.NET Core identité ajoute la fonctionnalité de connexion de l'interface utilisateur à ASP.NET Core Web Apps. Pour sécuriser les API Web et la commande SPAs, utilisez l'une des méthodes suivantes :

- [Azure Active Directory](#)
- [Azure Active Directory B2C \(Azure ad B2C\)\]](#)
- [IdentityServer4](#)

IdentityServer4 est un Framework OpenID Connect et OAuth 2,0 pour ASP.NET Core 3,0. IdentityServer4 active les fonctionnalités de sécurité suivantes :

- Authentification en tant que service (AaaS)
- Authentification unique (SSO) sur plusieurs types d'applications
- Contrôle d'accès pour les API
- Passerelle de Fédération

Pour plus d'informations, consultez [Bienvenue dans IdentityServer4](#).

## Étapes suivantes

Pour plus d'informations sur la création d'API web ASP.NET Core, consultez les ressources suivantes :

- [Version YouTube de cet article](#)
- [Créer des API web avec ASP.NET Core](#)
- [Types de retour de l'action du contrôleur dans ASP.NET Core API Web](#)

---

Cette page est-elle utile ?

 Oui  Non

---