Week-2 Web application Assignment

1. Explain in detail about CRUD operation in MongoDB.

Regardless of why you are using a MongoDB server, you'll need to perform CRUD operations on it. The basic methods of interacting with a MongoDB server are called CRUD operations. CRUD stands for Create, Read, Update, and Delete. These CRUD methods are the primary ways you will manage the data in your databases.

In this article, we'll cover the definition of CRUD. We'll then examine how to execute MongoDB CRUD operations using the MongoDB Query Language (MQL).

CRUD in MongoDB

CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.

MongoDB documents are modified by connecting to a server, querying the proper documents, and then changing the setting properties before sending the data back to the database to be updated. CRUD is data-oriented, and it's standardized according to HTTP action verbs.

When it comes to the individual CRUD operations:

- The Create operation is used to insert new documents in the MongoDB database.
- The Read operation is used to query a document in the database.
- The Update operation is used to modify existing documents in the database.
- The Delete operation is used to remove documents in the database.

Create Operations

For MongoDB CRUD, if the specified collection doesn't exist, the create operation will create the collection when it's executed. Create operations in MongoDB target a single collection, not multiple collections. Insert operations in MongoDB are atomic on a single document level.

MongoDB provides two different create operations that you can use to insert documents into a collection:

- db.collection.insertOne()
- db.collection.insertMany()

insertOne()

As the namesake, insertOne() allows you to insert one document into the collection. For this example, we're going to work with a collection called RecordsDB. We can insert a single entry into our collection by calling the insertOne() method on RecordsDB. We then provide the information we want to insert in the form of key-value pairs, establishing the schema.

insertMany()

It's possible to insert multiple items at one time by calling the *insertMany()* method on the desired collection. In this case, we pass multiple items into our chosen collection (*RecordsDB*) and separate them by commas. Within the parentheses, we use brackets to indicate that we are passing in a list of multiple entries. This is commonly referred to as a nested method.

Operations

The read operations allow you to supply special query filters and criteria that let you specify which documents you want. The MongoDB documentation contains more information on the available query filters. Query modifiers may also be used to change how many results are returned.

MongoDB has two methods of reading documents from a collection:

- db.collection.find()
- db.collection.findOne()

find()

In order to get all the documents from a collection, we can simply use the *find()* method on our chosen collection. Executing just the *find()* method with no arguments will return all records currently in the collection.

findOne()

In order to get one document that satisfies the search criteria, we can simply use the *findOne()* method on our chosen collection. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of

documents on the disk. If no documents satisfy the search criteria, the function returns null. The function takes the following form of syntax.

Operations

Like create operations, update operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.

You should be careful when updating documents, as updates are permanent and can't be rolled back. This applies to delete operations as well.

For MongoDB CRUD, there are three different methods of updating documents:

- db.collection.updateOne()
- db.collection.updateMany()
- db.collection.replaceOne()

updateOne()

We can update a currently existing record and change a single document with an update operation. To do this, we use the *updateOne()* method on a chosen collection, which here is "RecordsDB." To update a document, we provide the method with two arguments: an update filter and an update action.

updateMany()

updateMany() allows us to update multiple items by passing in a list of items, just as we did when inserting multiple items. This update operation uses the same syntax for updating a single document.

replaceOne()

The *replaceOne()* method is used to replace a single document in the specified collection. *replaceOne()*replaces the entire document, meaning fields in the old document not contained in the new will be lost.

Delete Operations

Delete operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document. You can provide delete operations with

filters and criteria in order to specify which documents you would like to delete from a collection. The filter options rely on the same syntax that read operations utilize.

MongoDB has two different methods of deleting records from a collection:

- db.collection.deleteOne()
- db.collection.deleteMany()

deleteOne()

deleteOne() is used to remove a document from a specified collection on the MongoDB server. A filter criteria is used to specify the item to delete. It deletes the first record that matches the provided filter.

deleteMany()

deleteMany() is a method used to delete multiple documents from a desired collection with a single delete operation. A list is passed into the method and the individual items are defined with filter criteria as in *deleteOne()*.

2. What is a namespace in MongoDB?

The official definition of "namespace" is here: The canonical name for a collection or index in MongoDB. The namespace is a combination of the database name and the name of the collection or index, like so: [database-name]. [collection-or-index-name]. All documents belong to a namespace.

Namespace MongoDB.Entities

Classes

AsObjectIdAttribute

Use this attribute to mark a string property to store the value in MongoDB as ObjectID if it is a valid ObjectId string. If it is not a valid ObjectId string, it will be stored as string. This is useful when using custom formats for the ID field.

AsyncEventHandlerExtensions

CollectionAttribute

Specifies a custom MongoDB collection name for an entity type.

Coordinates2D

Represents a 2D geographical coordinate consisting of longitude and latitude

DataStreamer

Provides the interface for uploading and downloading data chunks for file entities.

Date

A custom date/time type for precision datetime handling

DB

The main entrypoint for all data access methods of the library

DBContext

This db context class can be used as an alternative entry point instead of the DB static class.

Distinct<T, TProperty>

Represents a MongoDB Distinct command where you can get back distinct values for a given property of a given Entity.

DontPreserveAttribute

Properties that don't have this attribute will be omitted when using SavePreserving() TIP: These attribute decorations are only effective if you do not specify a preservation expression when calling SavePreserving()

Entity

Inherit this class for all entities you want to store in their own collection.

Extensions

Extension methods for entities

FieldAttribute

Specifies the field name and/or the order of the persisted document.

FileEntity

Inherit this base class in order to create your own File Entities

Find<T>

Represents a MongoDB Find command.

TIP: Specify your criteria using .Match() .Sort() .Skip() .Take() .Project() .Option() methods and finally call .Execute()

Note: For building queries, use the DB.Fluent* interfaces

Find<T, TProjection>

Represents a MongoDB Find command with the ability to project to a different result type.

TIP: Specify your criteria using .Match() .Sort() .Skip() .Take() .Project() .Option() methods and finally call .Execute()

FuzzyString

Use this type to store strings if you need fuzzy text searching with MongoDB

TIP: There's a default limit of 250 characters for ensuring best performance. If you exceed the default limit, an exception will be thrown. You can increase the limit by sacrificing performance/resource utilization by setting the static property `FuzzyString.CharacterLimit = 500` at startup.

GeoNear<T>

Fluent aggregation pipeline builder for GeoNear

IgnoreAttribute

Use this attribute to ignore a property when persisting an entity to the database.

IgnoreDefaultAttribute

Use this attribute to ignore a property when persisting an entity to the database if the value is null/default.

Index<T>

Represents an index creation command

TIP: Define the keys first with .Key() method and finally call the .Create() method.

InverseSideAttribute

Indicates that this property is the inverse side of a many-to-many relationship

JoinRecord

Represents a parent-child relationship between two entities.

TIP: The ParentID and ChildID switches around for many-to-many relationships depending on the side of the relationship you're accessing.

Many<TChild>

Represents a one-to-many/many-to-many relationship between two Entities.

WARNING: You have to initialize all instances of this class before accessing any of it's members.

Initialize from the constructor of the parent entity as follows:

```
this.InitOneToMany(() => Property);

this.InitManyToMany(() => Property, x => x.OtherProperty);
```

ManyBase

Base class providing shared state for Many'1 classes

Migration

Represents a migration history item in the database

ModifiedBy

NameAttribute

ObjectIdAttribute

Use this attribute to mark a property in order to save it in MongoDB server as ObjectId

One<T>

Represents a one-to-one relationship with an IEntity.

OwnerSideAttribute

Indicates that this property is the owner side of a many-to-many relationship

PagedSearch<T>

Represents an aggregation query that retrieves results with easy paging support.

PagedSearch<T, TProjection>

Represents an aggregation query that retrieves results with easy paging support.

PreserveAttribute

Use this attribute on properties that you want to omit when using SavePreserving() instead of supplying an expression. TIP: These attribute decorations are only effective if you do not specify a preservation expression when calling SavePreserving()

Prop

This class provides methods to generate property path strings from lambda expression.

Replace<T>

Represents an UpdateOne command, which can replace the first matched document with a given entity

TIP: Specify a filter first with the .Match(). Then set entity with .WithEntity() and finally call .Execute() to run the command.

Template

A helper class to build a JSON command from a string with tag replacement

Template<T>

A helper class to build a JSON command from a string with tag replacement

Template<TInput, TResult>

A helper class to build a JSON command from a string with tag replacement

Transaction

Represents a transaction used to carry out inter-related write operations.

TIP: Remember to always call .Dispose() after use or enclose in a 'Using' statement.

IMPORTANT: Use the methods on this transaction to perform operations and not the methods on the DB class.

Update<T>

Represents an update command

TIP: Specify a filter first with the .Match(). Then set property values with .Modify() and finally call .Execute() to run the command.

UpdateAndGet<T>

Update and retrieve the first document that was updated.

TIP: Specify a filter first with the .Match(). Then set property values with .Modify() and finally call .Execute() to run the command.

UpdateAndGet<T, TProjection>

Update and retrieve the first document that was updated.

TIP: Specify a filter first with the .Match(). Then set property values with .Modify() and finally call .Execute() to run the command.

UpdateBase<T>

Watcher<T>

Watcher for subscribing to mongodb change streams.

3. How do I handle 404 responses?

ASP.NET Core MVC is the .NET Core counterpart of the ASP.NET MVC framework for building cross-platform, scalable, high-performance web applications and APIs using the Model-View-Controller design pattern. Surprisingly, although ASP.NET Core provides plenty of options for handling 404 errors gracefully, the ASP.NET Core MVC runtime doesn't take advantage of them by default.

As a result, when a web page is not found and a 404 error is returned by the application, ASP.NET Core MVC presents only a generic browser error page (as shown in Figure 1 below). This article discusses three options in ASP.NET Core we can use to handle 404 errors more gracefully.

Create an ASP.NET Core MVC project

First off, let's create an ASP.NET Core project in Visual Studio. Assuming Visual Studio 2019 is installed in your system, follow the steps outlined below to create a new ASP.NET Core project in Visual Studio.

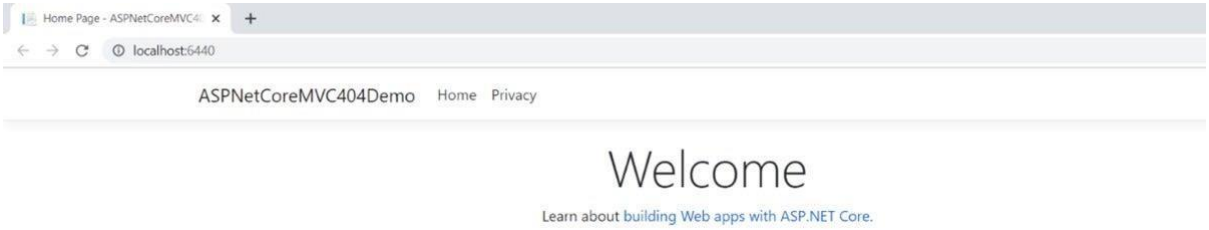
1. Launch the Visual Studio IDE.
2. on "Create new project."

1. In the "Create new project" window, select "ASP.NET Core Web Application" from the list of templates displayed.
2. Click Next.
3. In the "Configure your new project" window shown next, specify the name and location for the new project.
4. Click Create.
5. In the "Create a New ASP.NET Core Web Application" window, select .NET Core as the runtime and ASP.NET Core 3.1 (or later) from the drop-down list at the top.

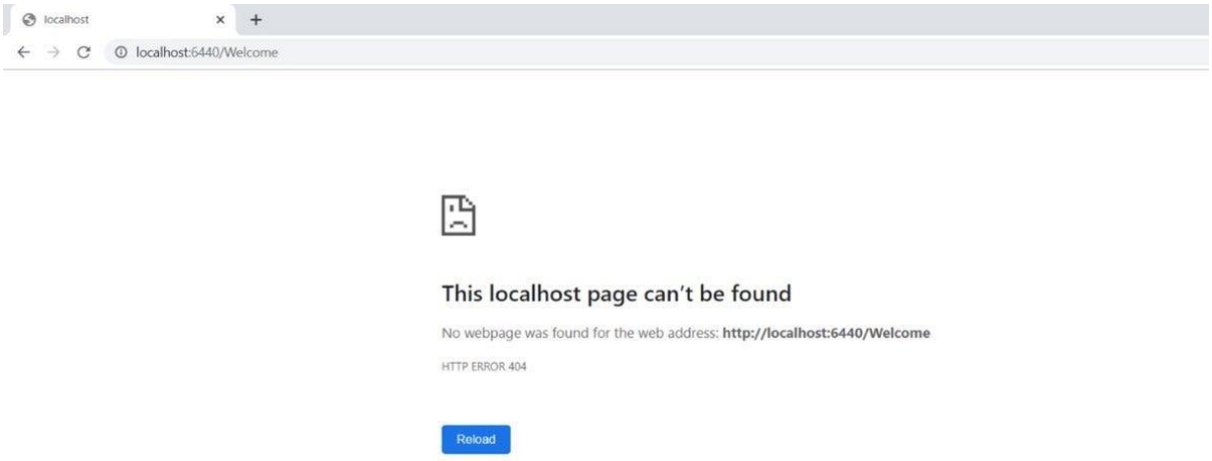
6. Select “Web Application (Model-View-Controller)” as the project template to create a new ASP.NET Core MVC application.
7. Ensure that the check boxes “Enable Docker Support” and “Configure for HTTPS” are unchecked as we won’t be using those features here.
8. Ensure that Authentication is set to “No Authentication” as we won’t be using authentication either.
9. Click Create.

Following these steps will create a new ASP.NET Core MVC project in Visual Studio 2019. We’ll use this project to illustrate our 404 error handling options in the subsequent sections of this article.

When you execute the ASP.NET Core MVC project we’ve created in the preceding section, you will see the home page of the application together with the welcome message as shown in Figure 1 below.



IDG



IDG

Check Response.StatusCode in ASP.NET Core MVC

There are several ways in which you can improve on this generic error page. A simple solution is to check for the HTTP status code 404 in the response. If found, you can redirect the control to a page that exists. The following code snippet illustrates how you can write the necessary code in the Configure method of the Startup class to redirect to the home page if a 404 error has occurred.

```
app.Use(async (context, next) =>
  {
     await next();
     if (context.Response.StatusCode == 404)
     {
        context.Request.Path = "/Home";
        await next();
     }
  });
```

Now if you execute the application and try to browse the URL http://localhost:6440/welcome, you will be redirected to the home page of the application.

The complete code of the Configure method is given below for your reference.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
     {
        if (env.IsDevelopment())
        {
           app.UseDeveloperExceptionPage();
        }
        else
        {
           app.UseExceptionHandler("/Home/Error");
        }
        app.Use(async (context, next) =>
        {
           await next();
           if (context.Response.StatusCode == 404)
           {
              context.Request.Path = "/Home";
              await next();
           }
        });
        app.UseStaticFiles();
        app.UseRouting();
```

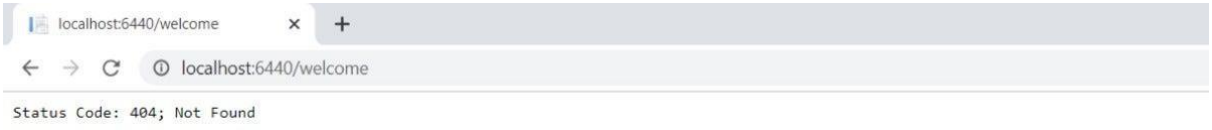
```
        app.UseAuthorization();
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller=Home}/{action=Index}/{id?}");
        });
    }
```

Use UseStatusCodePages middleware in ASP.NET Core MVC

A second solution for handling 404 errors in ASP.NET Core is by using the built-in UseStatusCodePages middleware. The following code snippet shows how you can implement StatusCodePages in the Configure method of the Startup class.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        app.UseStatusCodePages();
        //Other code
    }
```

Now when you execute the application and browse to the non-existent resource, the output will be similar to Figure 3.



IDG

Figure 3: The ASP.NET Core StatusCodePages middleware in action.

Use UseStatusCodePagesWithReExecute middleware in ASP.NET Core MVC

You can take advantage of the UseStatusCodePagesWithReExecute middleware to handle non-success status codes in cases where the process of generating the response has not been started. Hence this middleware will not handle HTTP 404 status code errors — rather, when a 404 error occurs the control will be passed to another controller action to handle the error.

The following code snippet illustrates how you can use this middleware to redirect to another action method.

```
app.UseStatusCodePagesWithReExecute("/Home/HandleError/{0}");
```

Here's what the action method would look like.

```
[Route("/Home/HandleError/{code:int}")]
public IActionResult HandleError(int code)
{
  ViewData["ErrorMessage"] = $"Error occurred. The ErrorCode is: {code}";
  return View("~/Views/Shared/HandleError.cshtml");
}
```

I leave it to you to create the HandleError view to display the error message.

Finally, you might want to create views specifically for an error code. For example, you might create views such as Home/Error/500.cshtml or Home/Error/404.cshtml. You could then check the HTTP error code and redirect to the appropriate error page.

Yet another way of handling page not found errors is by using a custom view and setting the error code appropriately. When an error occurs in your application, you could redirect the user to the appropriate error page and display your custom error message describing the error.

4.Connect to a running mongo instance, use a database named mongo_practice.

MongoDB Practice

MongoDB Exercise in mongo shell

Connect to a running mongo instance, use a database named mongo_practice.

Document all your queries in a javascript file to use as a reference.

Insert Documents

Insert the following documents into a movies collection.

```
title : Fight Club
writer : Chuck Palahniuk
year : 1999
actors : [
```

  Brad Pitt
  Edward Norton
]
title : Pulp Fiction
writer : Quentin Tarantino
year : 1994
actors : [
  John Travolta
  Uma Thurman
]
title : Inglorious Basterds
writer : Quentin Tarantino
year : 2009
actors : [
  Brad Pitt
  Diane Kruger
  Eli Roth
]
title : The Hobbit: An Unexpected Journey
writer : J.R.R. Tolkein
year : 2012
franchise : The Hobbit
title : The Hobbit: The Desolation of Smaug
writer : J.R.R. Tolkein
year : 2013
franchise : The Hobbit
title : The Hobbit: The Battle of the Five Armies
writer : J.R.R. Tolkein
year : 2012
franchise : The Hobbit
synopsis : Bilbo and Company are forced to engage in a war against an array of combatants and keep the Lonely Mountain from falling into the hands of a rising darkness.
title : Pee Wee Herman's Big Adventure
title : Avatar

Query / Find Documents

query the movies collection to

10. get all documents
11. get all documents with writer set to "Quentin Tarantino"
12. get all documents where actors include "Brad Pitt"
13. get all documents with franchise set to "The Hobbit"
14. get all movies released in the 90s
15. get all movies released before the year 2000 or after 2010

Update Documents

16. add a synopsis to "The Hobbit: An Unexpected Journey" : "A reluctant hobbit, Bilbo Baggins, sets out to the Lonely Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold within it - from the dragon Smaug."

17. add a synopsis to "The Hobbit: The Desolation of Smaug" : "The dwarves, along with Bilbo Baggins and Gandalf the Grey, continue their quest to reclaim Erebor, their homeland, from Smaug. Bilbo Baggins is in possession of a mysterious and magical ring."
18. add an actor named "Samuel L. Jackson" to the movie "Pulp Fiction"

Text Search

19. find all movies that have a synopsis that contains the word "Bilbo"
20. find all movies that have a synopsis that contains the word "Gandalf"
21. find all movies that have a synopsis that contains the word "Bilbo" and not the word "Gandalf"
22. find all movies that have a synopsis that contains the word "dwarves" or "hobbit"
23. find all movies that have a synopsis that contains the word "gold" and "dragon"

Delete Documents

24. delete the movie "Pee Wee Herman's Big Adventure"
25. delete the movie "Avatar"

Relationships

Insert the following documents into a users collection

username : GoodGuyGreg
first_name : "Good Guy"
last_name : "Greg"
username : ScumbagSteve
full_name :
  first : "Scumbag"
  last : "Steve"

Insert the following documents into a posts collection

username : GoodGuyGreg
title : Passes out at party
body : Wakes up early and cleans house
username : GoodGuyGreg
title : Steals your identity
body : Raises your credit score
username : GoodGuyGreg
title : Reports a bug in your code
body : Sends you a Pull Request
username : ScumbagSteve
title : Borrows something
body : Sells it
username : ScumbagSteve
title : Borrows everything
body : The end
username : ScumbagSteve
title : Forks your repo on github
body : Sets to private

Insert the following documents into a comments collection

username : GoodGuyGreg
comment : Hope you got a good deal!
post : [post_obj_id]

where [post_obj_id] is the ObjectId of the posts document: "Borrows something"

username : GoodGuyGreg
comment : What's mine is yours!
post : [post_obj_id]

where [post_obj_id] is the ObjectId of the posts document: "Borrows everything"

username : GoodGuyGreg
comment : Don't violate the licensing agreement!
post : [post_obj_id]

where [post_obj_id] is the ObjectId of the posts document: "Forks your repo on github

username : ScumbagSteve
comment : It still isn't clean
post : [post_obj_id]

where [post_obj_id] is the ObjectId of the posts document: "Passes out at party"

username : ScumbagSteve
comment : Denied your PR cause I found a hack
post : [post_obj_id]

where [post_obj_id] is the ObjectId of the posts document: "Reports a bug in your code"