Rest API NOTES WITH ASP.NET

Part 1

1. Create the dotnet web api by typing [dotnet new webapi]
2. Install the c-sharp extension in visual studio code
3. Inside the root folder, create a model folder
4. Right click on the model folder and with the help of the c-sharp extension, create a class file called [MallModel.cs]

Paste this code in the MallModel.cs file

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace DotNetApi.Models
{
public class MallModel
{
public int Id { get; set; }
public string Name { get; set; } = "Name";
public DateTime created { get; set; }
}
}
```

Inside the Controller Folder, right click on it and create a ApiController called ModelController.cs paste the following code in it

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DotNetApi.Models;
using Microsoft.AspNetCore.Mvc;

namespace DotNetApi.Controllers
{
[ApiController]
[Route("api/MallController")]
public class MallController:ControllerBase{
[HttpGet]
public IEnumerable<MallModel>GetModel(){
return new List<MallModel>{
new MallModel{Id=1, Name="Kofi"},
new MallModel{Id=2, Name="Ama"},
new MallModel{Id=3, Name="Yaa"},
new MallModel{Id=4, Name="Akos"},
new MallModel{Id=5, Name="Kwame"},
new MallModel{Id=6, Name="Kojo"},
};
}
}
}
```

PART 2

We may have some field in the model that we don't want to expose it to the user. So we can just create a DTO which will function as a wrapper between the model and the controller

in the model folder, create a folder named DTO and add this code to it

# MallDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace DotNetApi.Models.Dto
{
public class MallDto
{
public int Id { get; set; }
public string Name { get; set; } = "Name";
}
}
```

# The new controller class will be this
# MallController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DotNetApi.Models.Dto;
using Microsoft.AspNetCore.Mvc;

namespace DotNetApi.Controllers
{
[ApiController]
[Route("api/Mall")]
public class MallController : ControllerBase
{
[HttpGet]
public IEnumerable<MallDTO> GetMall(){
```

```
return new List<MallDTO>{
new MallDTO{Id = 1, Name = "Solomon"},
new MallDTO{Id = 2, Name = "Whiskey"},
new MallDTO{Id = 3, Name = "Jack"},
new MallDTO{Id = 4, Name = "Jill"},
new MallDTO{Id = 5, Name = "Max"},
new MallDTO{Id = 6, Name = "Charlie"},

};

}
}
}
```

PART 3

# Create a Data folder in the root project directory and add a class MallStore

# Inside the MallStore class, add the following code

# MallStore.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DotNetApi.Models.Dto;

namespace DotNetApi.Data
{
public static class MallStore
{
public static List<MallDTO> MallList = new List<MallDTO>{
new MallDTO{Id = 1, Name = "Solomon"},
new MallDTO{Id = 2, Name = "Whiskey"},
new MallDTO{Id = 3, Name = "Jack"},
new MallDTO{Id = 4, Name = "Jill"},
new MallDTO{Id = 5, Name = "Max"},
new MallDTO{Id = 6, Name = "Charlie"},

};
}
}
```

# The mall controller will look like this:

MallController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DotNetApi.Data;
using DotNetApi.Models.Dto;
using Microsoft.AspNetCore.Mvc;

namespace DotNetApi.Controllers
{
[ApiController]
[Route("api/Mall")]
public class MallController : ControllerBase
{
[HttpGet]
public IEnumerable<MallDTO> GetMall(){
return MallStore.MallList;

}
}
}
```

PART 4

In this section, we are going to perform CRUD operation, which is Create, Read, Update, Delete

For the Read Section, This will be the Code inside the MallController

```
[HttpGet]
public IEnumerable<MallDTO> GetMall(){
return MallStore.MallList;

}

[HttpGet("Id")]
public MallDTO GetOneMall(int Id){

return MallStore.MallList.FirstOrDefault(x=>x.Id == Id);

}

The HTTP REQUEST MUST HAVE A RESPONSE CODE WHICH WILL BE
[HttpGet]
public ActionResult<IEnumerable<MallDTO>> GetMall(){
return Ok(MallStore.MallList);

}

[HttpGet("Id")]
public ActionResult<MallDTO> GetOneMall(int Id){
```

```
if (Id <= 0){
return BadRequest();
}
var Mall = MallStore.MallList.FirstOrDefault(x=>x.Id == Id);
if (Mall == null){
return NotFound();
}
return Ok(Mall);

}
```

We can further add dynamic status code to the Http Get so that it will return the various Status codes

```
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK)]
public ActionResult<IEnumerable<MallDTO>> GetMall(){
return Ok(MallStore.MallList);

}



[HttpGet("Id")]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<MallDTO> GetOneMall(int Id){
if (Id <= 0){
return BadRequest();
}
var Mall = MallStore.MallList.FirstOrDefault(x=>x.Id == Id);
if (Mall == null){
return NotFound();
}
return Ok(Mall);
}
```

For the Create Section, This will be the Code inside the MallController

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public ActionResult<MallDTO> CreateMall([FromBody]MallDTO mallDTO){
```

```
if(mallDTO == null){
return BadRequest(mallDTO);
}

if(mallDTO.Id>0){
return StatusCode(StatusCodes.Status500InternalServerError);
}
mallDTO.Id = MallStore.MallList.OrderByDescending(u=>u.Id).FirstOrDefault().Id+1;
MallStore.MallList.Add(mallDTO);

return Ok(mallDTO);


}
```

# We can also give a route to the new created Mall object by setting an explicit name to the mall get method which has a specific Id

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DotNetApi.Data;
using DotNetApi.Models.Dto;
using Microsoft.AspNetCore.Mvc;

namespace DotNetApi.Controllers
{
[ApiController]
[Route("api/Mall")]
public class MallController : ControllerBase
{
/*

GET REQUEST AND ID CODES



*/
[HttpGet]
[ProducesResponseType(StatusCodes.Status200OK)]
public ActionResult<IEnumerable<MallDTO>> GetMall(){
return Ok(MallStore.MallList);

}


[HttpGet("{Id:int}",Name="GetMall")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<MallDTO> GetOneMall(int Id){
if (Id <= 0){
return BadRequest();
}
var Mall = MallStore.MallList.FirstOrDefault(x=>x.Id == Id);
if (Mall == null){
return NotFound();
}
return Ok(Mall);
}



/*
HTTP POST REQUEST
*/

[HttpPost]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public ActionResult<MallDTO> CreateMall([FromBody]MallDTO mallDTO){

if(mallDTO == null){
return BadRequest(mallDTO);
}

if(mallDTO.Id>0){
return StatusCode(StatusCodes.Status500InternalServerError);
}
mallDTO.Id = MallStore.MallList.OrderByDescending(u=>u.Id).FirstOrDefault().Id+1;
MallStore.MallList.Add(mallDTO);

return CreatedAtRoute("GetMall", new{Id = mallDTO.Id}, mallDTO);


}



}
}
```

# We can also set a required and max length for the Model Dto so as to ensure that our users do the right thing

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace DotNetApi.Models.Dto
{
public class MallDTO
{
public int Id { get; set; }

[Required ]
[MaxLength(50)]
public string Name { get; set; } = "Name";
}
}
```

# Since we use the [ApiConroller] in the controller class, it give us a basic feature of the Controller.

# If we disable it, we will need to explicitly check things

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public ActionResult<MallDTO> CreateMall([FromBody]MallDTO mallDTO){

if(!ModelState.IsValid){
return BadRequest(ModelState);
}

if(mallDTO == null){
return BadRequest(mallDTO);
}

if(mallDTO.Id>0){
return StatusCode(StatusCodes.Status500InternalServerError);
}
mallDTO.Id = MallStore.MallList.OrderByDescending(u=>u.Id).FirstOrDefault().Id+1;
MallStore.MallList.Add(mallDTO);
return CreatedAtRoute("GetMall", new{Id = mallDTO.Id}, mallDTO);


}
```

# To make the Name field unique we need to do this manually

```
[HttpPost]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public ActionResult<MallDTO> CreateMall([FromBody]MallDTO mallDTO){


if(MallStore.MallList.FirstOrDefault(u=>u.Name.ToLower()==mallDTO.Name.ToLower())!
=null)
{
ModelState.AddModelError("CustomError","Name already exists");
return BadRequest(ModelState);
}


if(mallDTO == null){
return BadRequest(mallDTO);
}

if(mallDTO.Id>0){
return StatusCode(StatusCodes.Status500InternalServerError);
}
mallDTO.Id = MallStore.MallList.OrderByDescending(u=>u.Id).FirstOrDefault().Id+1;
MallStore.MallList.Add(mallDTO);

return CreatedAtRoute("GetMall", new{Id = mallDTO.Id}, mallDTO);


}
```

# We can also Delete A Mall Objects

```
[HttpDelete("{Id:int}", Name="DeleteMall")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult DeleteMall(int Id){

if(Id ==0){
return BadRequest();
}

var mall = MallStore.MallList.FirstOrDefault(m => m.Id == Id);
if(mall == null){
return NotFound();
}
MallStore.MallList.Remove(mall);
return NoContent();



}
```

# We can update all the fields of a model using the Http Put Method and this is the code for that

```
[HttpPut("{Id:int}",Name="UpdateMall")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]


public IActionResult UpdateMall(int Id,[FromBody]MallDTO mallDTO){

if(Id==null || Id != mallDTO.Id){
return BadRequest();
}

var mall = MallStore.MallList.FirstOrDefault(m => m.Id == Id);
mall.Name = mallDTO.Name;
mall.Age = mallDTO.Age;
mall.Contact = mallDTO.Contact;
return NoContent();


}
```

# If we want to update a single field using the Http Patch

```
[HttpPatch("{Id:int}",Name ="Patch")]

public IActionResult PartialUpdate(int Id,JsonPatchDocument<MallDTO> patchDto){

if(patchDto==null || Id==0){
return BadRequest();
}
var mall = MallStore.MallList.FirstOrDefault(x => x.Id==Id);
if (mall==null){
return BadRequest();
}

patchDto.ApplyTo(mall,ModelState);
if(!ModelState.IsValid){
return BadRequest();
}
return NoContent();



}
```

Logging

The Loggers helps us to add our custom information to the command line of the dotnet application.
This is the full MallController.cs Code with Loggers

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using DotNetApi.Data;
using DotNetApi.Models.Dto;
using Microsoft.AspNetCore.Mvc;

namespace DotNetApi.Controllers
{
[ApiController]
[Route("api/Mall")]
public class MallController : ControllerBase
{
//Logging
private readonly ILogger<MallController> _logger;
public MallController(ILogger<MallController> logger)
{
_logger = logger;
}



/*

GET REQUEST AND ID CODES


*/
[HttpGet]
```

```
[ProducesResponseType(StatusCodes.Status200OK)]
public ActionResult<IEnumerable<MallDTO>> GetMall(){
_logger.LogInformation("Getting All Mall Objects");
return Ok(MallStore.MallList);

}



[HttpGet("{Id:int}",Name="GetMall")]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<MallDTO> GetOneMall(int Id){
if (Id <= 0){
_logger.LogError("The Id is "+Id);
return BadRequest();
}
var Mall = MallStore.MallList.FirstOrDefault(x=>x.Id == Id);
if (Mall == null){
return NotFound();
}
return Ok(Mall);
}




/*
HTTP POST REQUEST
*/

[HttpPost]
[ProducesResponseType(StatusCodes.Status200OK)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status500InternalServerError)]
public ActionResult<MallDTO> CreateMall([FromBody]MallDTO mallDTO){


if(MallStore.MallList.FirstOrDefault(u=>u.Name.ToLower()==mallDTO.Name.ToLower())!
=null)
{
ModelState.AddModelError("CustomError","Name already exists");
return BadRequest(ModelState);
}


if(mallDTO == null){
return BadRequest(mallDTO);
}
```

```
if(mallDTO.Id>0){
return StatusCode(StatusCodes.Status500InternalServerError);
}
mallDTO.Id = MallStore.MallList.OrderByDescending(u=>u.Id).FirstOrDefault().Id+1;
MallStore.MallList.Add(mallDTO);

return CreatedAtRoute("GetMall", new{Id = mallDTO.Id}, mallDTO);


}

/*

HTTP DELETE REQUEST
*/

[HttpDelete("{Id:int}", Name="DeleteMall")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult DeleteMall(int Id){

if(Id ==0){
return BadRequest();
}

var mall = MallStore.MallList.FirstOrDefault(m => m.Id == Id);
if(mall == null){
return NotFound();
}
MallStore.MallList.Remove(mall);
return NoContent();


}

/*

HTTP PUT Request

*/

[HttpPut("{Id:int}",Name="UpdateMall")]
[ProducesResponseType(StatusCodes.Status204NoContent)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]


public IActionResult UpdateMall(int Id,[FromBody]MallDTO mallDTO){
```

```
if(Id==null || Id != mallDTO.Id){
return BadRequest();
}

var mall = MallStore.MallList.FirstOrDefault(m => m.Id == Id);
mall.Name = mallDTO.Name;
mall.Age = mallDTO.Age;
mall.Contact = mallDTO.Contact;
return NoContent();


}





}
}
```

Adding Database

Follow this link to create a database sql server using docker in Mac Os
https://builtin.com/software-engineering-perspectives/sql-server-management-studio-mac

Authentication

Step 1

create a user model with this code and name is User.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace FlexiSecure.Models
{
public class User
{
public int Id { get; set; }

public string Name { get; set; } = string.Empty;
public string Email { get; set; } = string.Empty;
public byte[] PasswordHash { get; set; } = new byte[32];
public byte[] PasswordSalt { get; set; } = new byte[32];
public string? VerificationToken{get; set; }
public DateTime? VerifiedAt { get; set; }
public string? PasswordResetToken { get; set; }
public DateTime? ResetTokenExpires{get; set; }
}
}
```

step 2
install entity framework core, dotnet core design and microsoft sql server

step 3
Create a data context file and a data folder

DataContext.cs

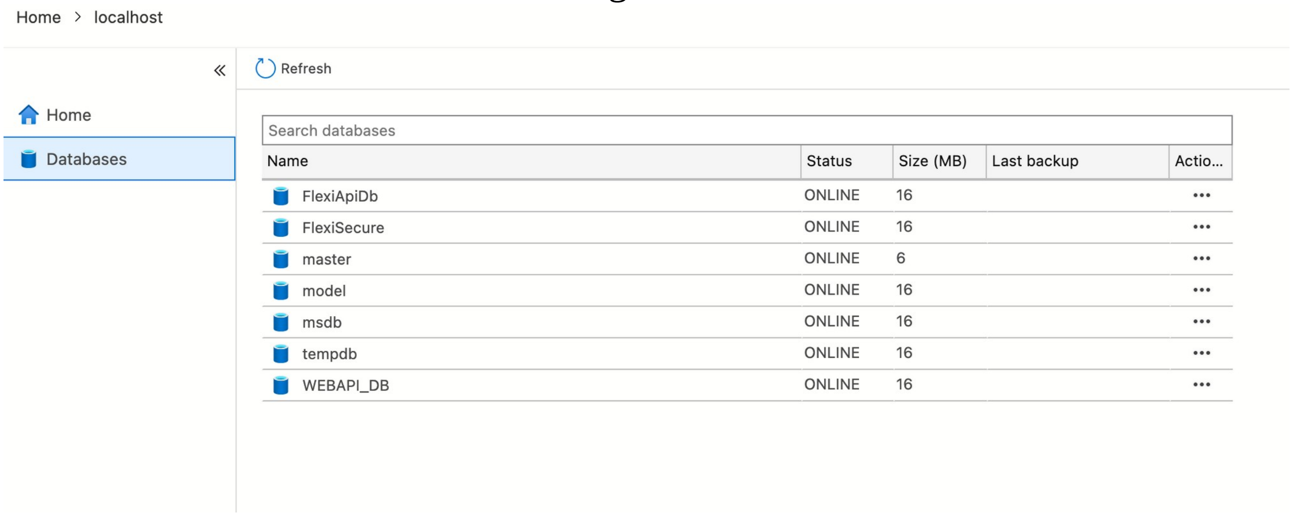
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;


namespace FlexiSecure.Data
{
public class DataContext : DbContext
{

//Construtor
public DataContext(DbContextOptions<DataContext> options) : base(options){}

//Connection Strings
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder){
base.OnConfiguring(optionsBuilder);

optionsBuilder.UseSqlServer("Server=localhost,1433;Database=FlexiSecure;User=sa;Pa
ssword=HydotTech;TrustServerCertificate=true;"
);
}



//Data sets
public DbSet<User> Users => Set<User>();


}
}
```

Step 4
In the terminal, install dotnet-ef tool by running this command
[dotnet tool install --global dotnet-ef]

Migrate the DbContext by running the command
[dotnet ef migrations add Initial]
This will create a migration folder in the root folder

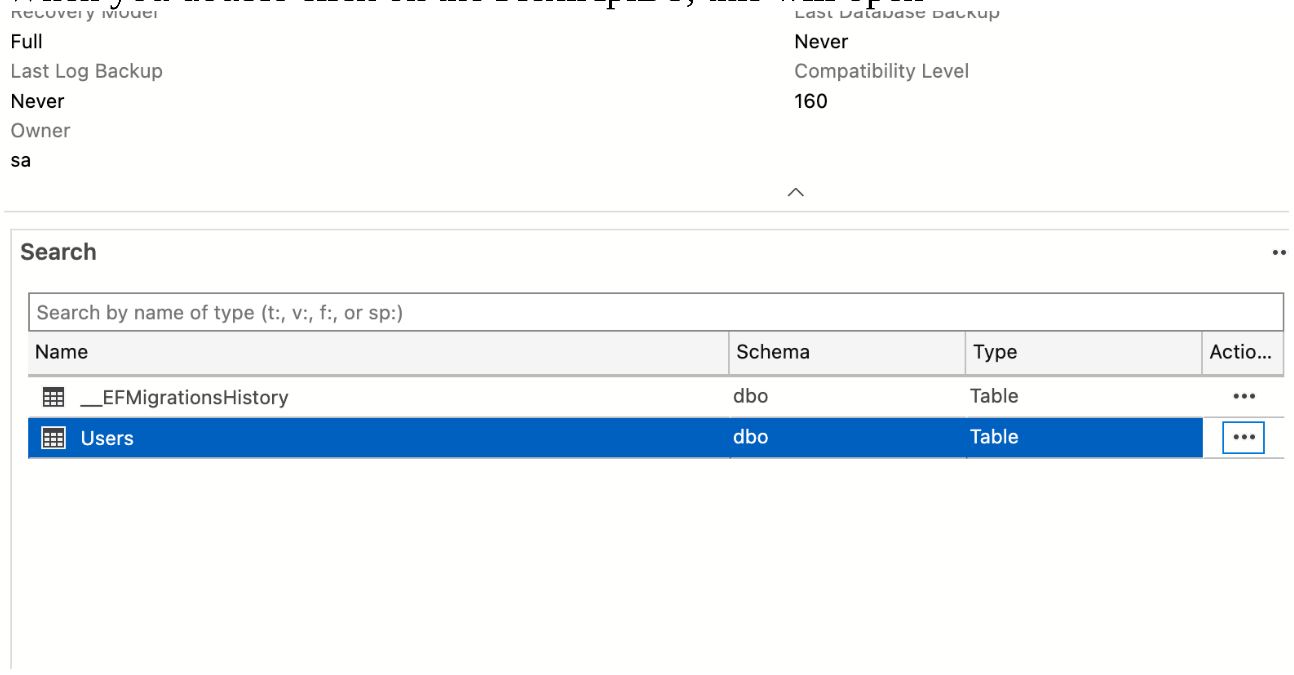
Also run the command to push the migration to the database
[dotnet ef database update]

This is how the Azure Database manager will look like



Name	Status	Size (MB)	Last backup	Actio...
FlexiApiDb	ONLINE	16		•••
FlexiSecure	ONLINE	16		•••
master	ONLINE	6		•••
model	ONLINE	16		•••
msdb	ONLINE	16		•••
tempdb	ONLINE	16		•••
WEBAPI_DB	ONLINE	16		•••

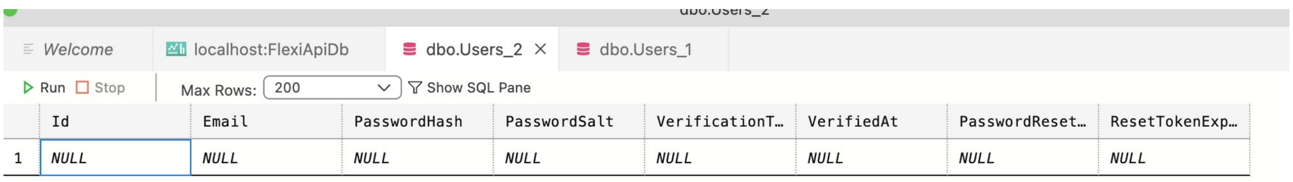
When you double click on the FlexiApiDb, this will open



Full	Never
Last Log Backup	Compatibility Level
Never	160
Owner	
sa	

Name	Schema	Type	Actio...
__EFMigrationsHistory	dbo	Table	•••
Users	dbo	Table	•••

When you click on the three dot, this will open (...)



	Id	Email	PasswordHash	PasswordSalt	VerificationT…	VerifiedAt	PasswordReset…	ResetTokenExp…
1	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Step 5
Create a UserRegister.cs file in the model folder and paste the code in it

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace FlexiSecure.Models
{
public class UserRegisterRequest
{
[Required,MinLength(3, ErrorMessage ="Enter a valid name")]
public string Name { get; set; } = string.Empty;
[Required, EmailAddress]
public string Email { get; set; } = string.Empty;
[Required, MinLength(6,ErrorMessage ="Please enter at least 6 characters ")]
public string Password { get; set; } = string.Empty;
[Required, Compare("Password")]
public string ConfirmPassword{get; set; } = string.Empty;


}
}
```

Step 6
The UserController.cs for the User Registration Will be like this

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace FlexiSecure.Controllers
```

```
{
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
private readonly DataContext _context;
public UserController(DataContext context)
{
_context = context;
}

[HttpPost("register")]
public async Task<IActionResult>Register(UserRegisterRequest request)
{

//Check if Email is already registered
if(_context.Users.Any(u=> u.Email == request.Email)){
return BadRequest();
};

//Create a Password Harsh
CreatePasswordHash(request.Password, out byte[]passwordHash,out
byte[]passwordSalt);

//Call the user model from User.cs, create a new objects
var user = new User{
Name = request.Name,
Email = request.Email,
PasswordHash=passwordHash,
PasswordSalt = passwordSalt,
VerificationToken = CreateRandomToken(),
};

// The _context is called from the DataContext file and insiidde that file, we
have the Users data set
//The Users dataset is using the User model
//We will the add the user object we created to the Users dataset
_context.Users.Add(user);
await _context.SaveChangesAsync();

return Ok("Hello "+request.Name+", successfully created");



}

private static void CreatePasswordHash(string password, out byte[] passwordHash,
out byte[] passwordSalt)
{
using (var hmac = new HMACSHA512())
{
passwordSalt = hmac.Key;
passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));


}

}

private string CreateRandomToken(){
return Convert.ToHexString(RandomNumberGenerator.GetBytes(64));
}

}
}
```

Step 7

Create a UserLoginRequest.cs model and add the following codes to it

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace FlexiSecure.Models
{
public class UserLoginRequest
{

[Required, EmailAddress]
public string Email { get; set; } = string.Empty;
[Required]
public string Password { get; set; } = string.Empty;

}
}
```

Step 8

The UserController.cs at this stage will look like this

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace FlexiSecure.Controllers
{
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
private readonly DataContext _context;
public UserController(DataContext context)
{
_context = context;
}

[HttpPost("register")]
public async Task<IActionResult>Register(UserRegisterRequest request)
{

//Check if Email is already registered
if(_context.Users.Any(u=> u.Email == request.Email)){
return BadRequest();
};

//Create a Password Harsh
CreatePasswordHash(request.Password, out byte[]passwordHash,out
byte[]passwordSalt);

//Call the user model from User.cs, create a new objects
var user = new User{
Name = request.Name,
Email = request.Email,
PasswordHash=passwordHash,
PasswordSalt = passwordSalt,
VerificationToken = CreateRandomToken(),
};

// The _context is called from the DataContext file and insiidde that file, we
have the Users data set
//The Users dataset is using the User model
//We will the add the user object we created to the Users dataset
_context.Users.Add(user);
await _context.SaveChangesAsync();

return Ok("Hello "+request.Name+", successfully created");



}
```

```
[HttpPost("login")]
public async Task<IActionResult>Login(UserLoginRequest request)
{

var user = await _context.Users.FirstOrDefaultAsync(u => u.Email ==
request.Email);
if(user==null){
return BadRequest("User not found");
}

if(!VerifyPasswordHash(request.Password, user.PasswordHash, user.PasswordSalt))
{
return BadRequest("Password is incorrect");
}

if(user.VerifiedAt == null){
return BadRequest("User is not verified");
}
return Ok($"Welcome Back, {user.Email}! ");


}



private static bool VerifyPasswordHash(string password, byte[] passwordHash,
byte[] passwordSalt)
{
using (var hmac = new HMACSHA512(passwordSalt))
{

var computedHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
return computedHash.SequenceEqual(passwordHash);


}

}


private static void CreatePasswordHash(string password, out byte[] passwordHash,
out byte[] passwordSalt)
{
using (var hmac = new HMACSHA512())
{
passwordSalt = hmac.Key;
passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
}

}

private string CreateRandomToken(){
return Convert.ToHexString(RandomNumberGenerator.GetBytes(64));
}

}
}
```

Step 9

After creating our users, we can verify them by sending a token to their account.
The verify user code in the controller will be

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;

namespace FlexiSecure.Controllers
{
[ApiController]
[Route("api/[controller]")]
public class UserController : ControllerBase
{
private readonly DataContext _context;
public UserController(DataContext context)
{
_context = context;
}

[HttpPost("register")]
public async Task<IActionResult>Register(UserRegisterRequest request)
{

//Check if Email is already registered
if(_context.Users.Any(u=> u.Email == request.Email)){
return BadRequest();
};

//Create a Password Harsh
CreatePasswordHash(request.Password, out byte[]passwordHash,out
byte[]passwordSalt);
//Call the user model from User.cs, create a new objects
var user = new User{
Name = request.Name,
Email = request.Email,
PasswordHash=passwordHash,
PasswordSalt = passwordSalt,
VerificationToken = CreateRandomToken(),
};

// The _context is called from the DataContext file and insiidde that file, we
have the Users data set
//The Users dataset is using the User model
//We will the add the user object we created to the Users dataset
_context.Users.Add(user);
await _context.SaveChangesAsync();

return Ok("Hello "+request.Name+", successfully created");



}






[HttpPost("login")]
public async Task<IActionResult>Login(UserLoginRequest request)
{

var user = await _context.Users.FirstOrDefaultAsync(u => u.Email ==
request.Email);
if(user==null){
return BadRequest("User not found");
}

if(!VerifyPasswordHash(request.Password, user.PasswordHash, user.PasswordSalt))
{
return BadRequest("Password is incorrect");
}

if(user.VerifiedAt == null){
return BadRequest("User is not verified");
}


return Ok($"Welcome Back, {user.Email}! ");

}
```

```
[HttpPost("verify")]
public async Task<IActionResult>Verify(string token)
{

var user = await _context.Users.FirstOrDefaultAsync(u => u.VerificationToken ==
token);
if(user==null){
return BadRequest("Invalid token");
}

user.VerifiedAt = DateTime.Now;
await _context.SaveChangesAsync();


return Ok("Congratulations! You have been verified");




}







private static bool VerifyPasswordHash(string password, byte[] passwordHash,
byte[] passwordSalt)
{
using (var hmac = new HMACSHA512(passwordSalt))
{

var computedHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));
return computedHash.SequenceEqual(passwordHash);


}

}
```

```
private static void CreatePasswordHash(string password, out byte[] passwordHash,
out byte[] passwordSalt)
{
using (var hmac = new HMACSHA512())
{
passwordSalt = hmac.Key;
passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));


}

}

private string CreateRandomToken(){
return Convert.ToHexString(RandomNumberGenerator.GetBytes(64));
}

}
}
```