​ DATABASE PROOF IT2J

​

**​Olagoke Ayodeji**

**Rares Matei Husarescu**

**Version 1.1**

**Version Control**

|  |  |  |
| --- | --- | --- |
| **Version number** | **Changes** | **Member** |
| 1.0 | Initial version of the document | Olagoke Ayodeji |
| 1.1 | Filled document | Olagoke Ayodeji |

Contents

[Authorization and Authentication proof 4](#_Toc195017918)

[Stored Procedure 7](#_Toc195017919)

[Views 7](#_Toc195017920)

[Triggers 8](#_Toc195017921)

[User Permissions 9](#_Toc195017922)

[Backup Strategy 20](#_Toc195017923)

[Issue with ASP . Net 23](#_Toc195017924)

[References 30](#_Toc195017925)

# Authorization and Authentication proof

***N.b change the connection strings in the appsettings.cs then click run and the Api should be configured correctly***

A screenshot of a computer

AI-generated content may be incorrect.

To use the api you need to login with premade user to get access token.

The process is below and all calls to the api can be made in both XML or JSON.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# Stored Procedure

The following are the considered stored procedures for the application

A screenshot of a computer

AI-generated content may be incorrect.

# Views

The following are the considered views for the application

A screenshot of a computer

AI-generated content may be incorrect.

# Triggers

A screenshot of a computer

AI-generated content may be incorrect.

# User Permissions

To login via a role on the database, use the following credentials and passwords:

api\_medior\_login WITH PASSWORD = 'MediorApiPass!'

api\_senior\_login WITH PASSWORD = 'SeniorApiPass!'

api\_user\_login WITH PASSWORD = ‘GeneralApiPass!’

api\_junior\_login WITH PASSWORD = 'JuniorApiPass!'

For example a junior user will not be able to view the user or subscription table A screenshot of a computer

AI-generated content may be incorrect.

whilst an Api user can’t even see any tables at all.A white screen with a blue border

AI-generated content may be incorrect.

The accomplish this, both store procedures below were used:

CREATE PROCEDURE [dbo].[sp\_CreateAPIUsers]

AS

BEGIN

SET NOCOUNT ON;

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

BEGIN TRY

-- Create Logins

IF NOT EXISTS (SELECT \* FROM sys.server\_principals WHERE name = 'api\_junior\_login')

CREATE LOGIN api\_junior\_login WITH PASSWORD = 'JuniorApiPass!';

IF NOT EXISTS (SELECT \* FROM sys.server\_principals WHERE name = 'api\_medior\_login')

CREATE LOGIN api\_medior\_login WITH PASSWORD = 'MediorApiPass!';

IF NOT EXISTS (SELECT \* FROM sys.server\_principals WHERE name = 'api\_senior\_login')

CREATE LOGIN api\_senior\_login WITH PASSWORD = 'SeniorApiPass!';

IF NOT EXISTS (SELECT \* FROM sys.server\_principals WHERE name = 'api\_user\_login')

CREATE LOGIN api\_user\_login WITH PASSWORD = 'GeneralApiPass!';

-- Create Users

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'api\_junior')

CREATE USER api\_junior FOR LOGIN api\_junior\_login;

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'api\_medior')

CREATE USER api\_medior FOR LOGIN api\_medior\_login;

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'api\_senior')

CREATE USER api\_senior FOR LOGIN api\_senior\_login;

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'api\_user')

CREATE USER api\_user FOR LOGIN api\_user\_login;

-- Create Roles if they don't exist

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'JuniorRole')

CREATE ROLE JuniorRole;

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'MediorRole')

CREATE ROLE MediorRole;

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'SeniorRole')

CREATE ROLE SeniorRole;

IF NOT EXISTS (SELECT \* FROM sys.database\_principals WHERE name = 'APIUserRole')

CREATE ROLE APIUserRole;

-- Assign users to roles

ALTER ROLE JuniorRole ADD MEMBER api\_junior;

ALTER ROLE MediorRole ADD MEMBER api\_medior;

ALTER ROLE SeniorRole ADD MEMBER api\_senior;

ALTER ROLE APIUserRole ADD MEMBER api\_user;

COMMIT TRANSACTION;

PRINT 'API Users created and assigned to roles successfully.';

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

PRINT 'Error occurred: ' + ERROR\_MESSAGE();

END CATCH;

END;

This uses isolation levels to ensure that this stored procedure is currectly run and errors are accounted for.

-- Permissions for SeniorRole (full access)

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.[User] TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Profile TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Genre TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Subscription TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Referral TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Episode TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Film TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Preferences TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.[Role] TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Watch\_History TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Role\_Change\_Log TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Profile\_Genre TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Series TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Subtitles TO SeniorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Watch\_List TO SeniorRole;

-- Grant access to all views for SeniorRole

GRANT SELECT ON dbo.api\_recommended\_content TO SeniorRole;

GRANT SELECT ON dbo.api\_user\_roles\_view TO SeniorRole;

GRANT SELECT ON dbo.api\_user\_view TO SeniorRole;

GRANT SELECT ON dbo.api\_watch\_history TO SeniorRole;

GRANT SELECT ON dbo.vw\_ContentDetails TO SeniorRole;

-- Grant all stored procedures to SeniorRole

GRANT EXECUTE ON SCHEMA::dbo TO SeniorRole;

-- Permissions for MediorRole

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.[User] TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Profile TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Genre TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Referral TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Episode TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Film TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Preferences TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.[Role] TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Watch\_History TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Role\_Change\_Log TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Profile\_Genre TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Series TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Subtitles TO MediorRole;

GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Watch\_List TO MediorRole;

DENY SELECT, INSERT, UPDATE, DELETE ON dbo.Subscription TO MediorRole;

-- View permissions for MediorRole (deny api\_user\_roles\_view)

GRANT SELECT ON dbo.api\_recommended\_content TO MediorRole;

DENY SELECT ON dbo.api\_user\_roles\_view TO MediorRole;

GRANT SELECT ON dbo.api\_user\_view TO MediorRole;

GRANT SELECT ON dbo.api\_watch\_history TO MediorRole;

GRANT SELECT ON dbo.vw\_ContentDetails TO MediorRole;

-- Stored procedure permissions for MediorRole

GRANT EXECUTE ON OBJECT::[dbo].[AddUser] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[GetUserProfileDetails] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[SelectUser] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].DeleteUserById TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_AddToWatchList] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_CreateEpisode] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_CreateGenre] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_CreateProfile] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_CreateWatchHistory] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteContent] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteEpisode] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteGenre] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteProfile] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteWatchHistory] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetAllGenres] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].sp\_GetEpisodeById TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].sp\_GetProfileById TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetWatchHistoryByProfile] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetWatchListByProfile] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_RemoveFromWatchList] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateContent] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateEpisode] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateGenre] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateProfile] TO MediorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateWatchHistory] TO MediorRole;

-- Deny sensitive procedures for MediorRole

DENY EXECUTE ON OBJECT::[dbo].[CheckAndInsertDummyUsers] TO MediorRole;

DENY EXECUTE ON OBJECT::[dbo].[InsertDummyUsers] TO MediorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateUsersAndAssignRoles] TO MediorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateAPIUsers] TO MediorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateSubscription] TO MediorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_UpdateSubscription] TO MediorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_GetAllSubscriptions] TO MediorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_RemoveAPIUsersAndRoles] TO MediorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_SetupRolesAndPermission] TO MediorRole;

-- Permissions for JuniorRole

GRANT SELECT ON dbo.Genre TO JuniorRole;

GRANT SELECT ON dbo.Referral TO JuniorRole;

GRANT SELECT ON dbo.Episode TO JuniorRole;

GRANT SELECT ON dbo.Film TO JuniorRole;

GRANT SELECT ON dbo.Preferences TO JuniorRole;

GRANT SELECT ON dbo.[Role] TO JuniorRole;

GRANT SELECT ON dbo.Watch\_History TO JuniorRole;

GRANT SELECT ON dbo.Role\_Change\_Log TO JuniorRole;

GRANT SELECT ON dbo.Profile\_Genre TO JuniorRole;

GRANT SELECT ON dbo.Series TO JuniorRole;

GRANT SELECT ON dbo.Subtitles TO JuniorRole;

GRANT SELECT ON dbo.Watch\_List TO JuniorRole;

DENY SELECT, INSERT, UPDATE, DELETE ON dbo.[User] TO JuniorRole;

DENY SELECT, INSERT, UPDATE, DELETE ON dbo.Subscription TO JuniorRole;

DENY SELECT, INSERT, UPDATE, DELETE ON dbo.Profile TO JuniorRole;

-- View permissions for JuniorRole (deny api\_user\_roles\_view)

GRANT SELECT ON dbo.api\_recommended\_content TO JuniorRole;

DENY SELECT ON dbo.api\_user\_roles\_view TO JuniorRole;

GRANT SELECT ON dbo.api\_user\_view TO JuniorRole;

GRANT SELECT ON dbo.api\_watch\_history TO JuniorRole;

GRANT SELECT ON dbo.vw\_ContentDetails TO JuniorRole;

-- Stored procedure permissions for JuniorRole

GRANT EXECUTE ON OBJECT::[dbo].[sp\_AddToWatchList] TO JuniorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetAllGenres] TO JuniorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetEpisodeById] TO JuniorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetWatchHistoryByProfile] TO JuniorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetWatchListByProfile] TO JuniorRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_RemoveFromWatchList] TO JuniorRole;

-- Deny all user and subscription related procedures for JuniorRole

DENY EXECUTE ON OBJECT::[dbo].[AddUser] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[CheckAndInsertDummyUsers] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].DeleteUserById TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[GetUserProfileDetails] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[InsertDummyUsers] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[SelectUser] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateAPIUsers] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateProfile] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateSubscription] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateUsersAndAssignRoles] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateWatchHistory] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_DeleteProfile] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_DeleteWatchHistory] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_GetAllSubscriptions] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].sp\_GetProfileById TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_RemoveAPIUsersAndRoles] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_UpdateProfile] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_UpdateSubscription] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_UpdateWatchHistory] TO JuniorRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_SetupRolesAndPermission] TO JuniorRole;

-- Permissions for APIUserRole

GRANT EXECUTE ON SCHEMA::dbo TO APIUserRole;

GRANT SELECT ON dbo.api\_user\_view TO APIUserRole;

-- View permissions for APIUserRole (deny api\_user\_roles\_view)

GRANT SELECT ON dbo.api\_recommended\_content TO APIUserRole;

DENY SELECT ON dbo.api\_user\_roles\_view TO APIUserRole;

GRANT SELECT ON dbo.api\_watch\_history TO APIUserRole;

GRANT SELECT ON dbo.vw\_ContentDetails TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[AddUser] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].DeleteUserById TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[GetUserProfileDetails] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[SelectUser] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_AddToWatchList] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_CreateEpisode] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_CreateGenre] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_CreateProfile] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_CreateWatchHistory] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteContent] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteEpisode] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteGenre] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteProfile] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_DeleteWatchHistory] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetAllGenres] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetEpisodeById] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].sp\_GetProfileById TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetWatchHistoryByProfile] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_GetWatchListByProfile] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_RemoveFromWatchList] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateContent] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateEpisode] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateGenre] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateProfile] TO APIUserRole;

GRANT EXECUTE ON OBJECT::[dbo].[sp\_UpdateWatchHistory] TO APIUserRole;

-- Deny restricted procedures for APIUserRole

DENY EXECUTE ON OBJECT::[dbo].[CheckAndInsertDummyUsers] TO APIUserRole;

DENY EXECUTE ON OBJECT::[dbo].[InsertDummyUsers] TO APIUserRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateAPIUsers] TO APIUserRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateSubscription] TO APIUserRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_CreateUsersAndAssignRoles] TO APIUserRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_GetAllSubscriptions] TO APIUserRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_RemoveAPIUsersAndRoles] TO APIUserRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_UpdateSubscription] TO APIUserRole;

DENY EXECUTE ON OBJECT::[dbo].[sp\_SetupRolesAndPermission] TO APIUserRole;

DENY SELECT, INSERT, UPDATE, DELETE ON SCHEMA::dbo TO APIUserRole;

DECLARE @SQL NVARCHAR(MAX);

SET @SQL = 'ALTER ROLE ' + QUOTENAME(@Role) + ' ADD MEMBER ' + QUOTENAME(@UserName);

EXEC sp\_executesql @SQL;

COMMIT TRANSACTION;

PRINT 'Roles, permissions, and user assignment completed successfully.';

END TRY

BEGIN CATCH

ROLLBACK TRANSACTION;

PRINT 'Error: ' + ERROR\_MESSAGE();

END CATCH;

END;

# Backup Strategy

While our databases are hosted locally, the data storage process runs over a backup set up in Microsoft Azure. This choice has been made based on the 2 backup options of SSMS. The team had the choice to back up the database either on a local disk or online through URL.

Since this is an external project (external in: client outside of the NHL Stenden university), the local storage option was not up for discussion. Therefore, the group set up and used an Azure container, where the database is saved iteratively, every 7 days.

The group opted for a Mirror Backup on an Azure container, which occurs iteratively every 7 days instead of a differential backup. This choice was ideal for several reasons:

1. **Small File Size:** Mirror backups create an exact copy of the source data at the time of backup, resulting in smaller file sizes compared to full backups. This is particularly advantageous when dealing with limited storage resources.
2. **Efficiency:** Mirror backups provide an up-to-date copy of the data, ensuring that the most recent state is always available. This can simplify the recovery process, as there is no need to sift through multiple incremental or differential backups to piece together the current state of the data.
3. **Security:** Azure was the best match due to its simplicity to implement

Below are the steps to backup a database on an azure container.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Issue with ASP . Net

The framework did not allow for user without the sysadmin permission to call any stored procedure via:

ALTER SERVER ROLE sysadmin ADD MEMBER api\_user\_login;

The code below is the alternate version of the UserController which uses the stored procedures to function but only when logged in as the db owner or a role is edited to have the afore mentioned sever role; hence coding in such manner was redundant because it prevents the roles to have actual value since sysadmin gives them all rights.

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using project6.\_1Api.Entities;

using project6.\_1Api.Model;

using System.Collections.Generic;

using System.Data;

using Microsoft.Data.SqlClient;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using Microsoft.AspNetCore.Authorization;

namespace project6.\_1Api.Controllers

{

[Route("api/[controller]")]

[Authorize]

[ApiController]

[Produces("application/json", "application/xml")]

[Consumes("application/json", "application/xml")]

public class UserController : ControllerBase

{

private readonly databaseContext \_dbContext;

public UserController(databaseContext dbContext)

{

\_dbContext = dbContext;

}

// GET: api/User

[HttpGet("")]

public IActionResult GetAll()

{

var users = \_dbContext.User

.FromSqlRaw("GetAllUsers").ToList()

.Select(u => new Model.User

{

user\_id = u.User\_id, // Ensure lowercase

email = u.Email,

password = u.Password,

account\_status = u.Account\_status,

subscription\_id = u.Subscription\_id,

role\_id = u.Role\_id,

referred\_by = u.Referred\_by,

})

.ToList();

return users.Any() ? Ok(users) : NotFound();

}

// GET: api/User/{id}

[HttpGet("{id}")]

public IActionResult GetById(int id)

{

// Using FromSqlRaw to call stored procedure

var user = \_dbContext.User

.FromSqlRaw($"GetUserById {id}")

.AsEnumerable()

.FirstOrDefault();

if (user == null)

return NotFound();

return Ok(new Model.User

{

user\_id = user.User\_id,

email = user.Email,

password = user.Password,

account\_status = user.Account\_status,

subscription\_id = user.Subscription\_id,

role\_id = user.Role\_id,

referred\_by = user.Referred\_by

});

}

// POST: api/User

[HttpPost("")]

public IActionResult Create([FromBody] Model.User userModel)

{

if (!ModelState.IsValid)

return BadRequest(ModelState);

try

{

// Hash the password before sending to stored procedure

var hashedPassword = HashString(userModel.password);

// Using ExecuteSqlRaw to call stored procedure

var userIdParam = new SqlParameter("@User\_id", SqlDbType.Int) { Direction = ParameterDirection.Output };

\_dbContext.Database.ExecuteSqlRaw($"sp\_CreateUser @Email, @Password, @Account\_status, @Subscription\_id, @Role\_id, @Referred\_by",

new SqlParameter("@Email", userModel.email),

new SqlParameter("@Password", hashedPassword),

new SqlParameter("@Account\_status", userModel.account\_status ?? "active"),

new SqlParameter("@Subscription\_id", userModel.subscription\_id ?? (object)DBNull.Value),

new SqlParameter("@Role\_id", userModel.role\_id ?? (object)DBNull.Value),

new SqlParameter("@Referred\_by", userModel.referred\_by ?? (object)DBNull.Value));

return StatusCode(201, "User created successfully.");

}

catch (DbUpdateException ex) when (ex.InnerException is SqlException sqlEx && sqlEx.Number == 50001)

{

return Conflict($"Email '{userModel.email}' already exists.");

}

catch (DbUpdateException ex) when (ex.InnerException is SqlException sqlEx && sqlEx.Number == 50002)

{

return BadRequest("Invalid reference ID provided");

}

catch (Exception)

{

return StatusCode(500, "Error creating user. Please try again later.");

}

}

// PUT: api/User/{id}

[HttpPut("{id}")]

public IActionResult Update(int id, [FromBody] Model.User userModel)

{

if (!ModelState.IsValid)

return BadRequest(ModelState);

try

{

// Hash the password before sending to stored procedure

var hashedPassword = HashString(userModel.password);

// Using ExecuteSqlRaw to call stored procedure

\_dbContext.Database.ExecuteSqlRaw(

$"sp\_UpdateUser @User\_id, @Email, @Password, @Account\_status, @Subscription\_id, @Role\_id, @Referred\_by",

new SqlParameter("@User\_id", id),

new SqlParameter("@Email", userModel.email),

new SqlParameter("@Password", hashedPassword),

new SqlParameter("@Account\_status", userModel.account\_status),

new SqlParameter("@Subscription\_id", userModel.subscription\_id ?? (object)DBNull.Value),

new SqlParameter("@Role\_id", userModel.role\_id ?? (object)DBNull.Value),

new SqlParameter("@Referred\_by", userModel.referred\_by ?? (object)DBNull.Value));

return StatusCode(201, "User edited successfully.");

}

catch (DbUpdateException ex) when (ex.InnerException is SqlException sqlEx && sqlEx.Number == 50001)

{

return Conflict($"Email '{userModel.email}' already exists.");

}

catch (DbUpdateException ex) when (ex.InnerException is SqlException sqlEx && sqlEx.Number == 50002)

{

return BadRequest("Invalid reference ID provided");

}

catch (DbUpdateException ex) when (ex.InnerException is SqlException sqlEx && sqlEx.Number == 50003)

{

return NotFound("User not found");

}

catch (Exception)

{

return StatusCode(500, "Error updating user. Please try again later.");

}

}

// DELETE: api/User/{id}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

var user = \_dbContext.User

.FromSqlRaw($"GetUserById {id}")

.AsEnumerable()

.FirstOrDefault();

if (user != null)

{

\_dbContext.Database.ExecuteSqlRaw(

$"sp\_DeleteUser {id}");

return StatusCode(201, "User deleted successfully.");

}

else

{

return NotFound();

}

}

public static string HashString(string input)

{

using (SHA256 sha256 = SHA256.Create())

{

byte[] inputBytes = Encoding.UTF8.GetBytes(input);

byte[] hashBytes = sha256.ComputeHash(inputBytes);

return Convert.ToBase64String(hashBytes);

}

}

}

}

# References

Crocetti, R. F. a. P. (2022, December 14). *Types of backup explained: Full, incremental, differential, etc.* Data Backup. <https://www.techtarget.com/searchdatabackup/feature/Full-incremental-or-differential-How-to-choose-the-correct-backup-type>