To create a console application using .Net 6.0 or .Net 7.0

Follow the steps below:

Install the .NET 6.0 or .Net 7.0 SDK.

Create a new folder for your project.

Navigate to the project folder in a terminal or command prompt.

Run the following command:

dotnet new console

Select the .NET 6.0 or .Net 7.0 framework.

Press Enter to create the project.

This will create a new console application project with a single file, Program.cs.

To create DTOs and models based on the Figma designs

Follow these steps:

Review the Figma designs and identify the different entities that need to be modeled.

Create a new class for each entity.

Add properties to the classes to represent the different attributes of the entities.

Use the appropriate data types for the properties, based on the Figma designs.

For example, if the Figma designs show a Program entity with the following attributes:

Name

Description

Start Date

End Date

You would create a Program class with the following properties:

public class Program

{

public string Name { get; set; }

public string Description { get; set; }

public DateTime StartDate { get; set; }

public DateTime EndDate { get; set; }

}

To implement CRUD APIs for the 4 tabs

To implement CRUD APIs for the 4 tabs, you can use the following steps:

Create a new ASP.NET Core Web API project.

Add the following NuGet packages:

Microsoft.Azure.Cosmos

Autofac

xUnit

Create a new class for each API controller.

Add the appropriate HTTP methods to the controller methods to implement the CRUD operations.

Inject the necessary dependencies into the controller constructors.

Write unit tests for the API controllers.

For example, to implement a CRUD API for the Program entity, you would create a ProgramController class with the following methods:

public class ProgramController : ControllerBase

{

private readonly IProgramRepository \_programRepository;

public ProgramController(IProgramRepository programRepository)

{

\_programRepository = programRepository;

}

[HttpGet]

public async Task<ActionResult<IEnumerable<Program>>> GetAllPrograms()

{

var programs = await \_programRepository.GetAllPrograms();

return Ok(programs);

}

[HttpGet("{id}")]

public async Task<ActionResult<Program>> GetProgramById(int id)

{

var program = await \_programRepository.GetProgramById(id);

if (program == null)

{

return NotFound();

}

return Ok(program);

}

[HttpPost]

public async Task<ActionResult<Program>> CreateProgram([FromBody] Program program)

{

await \_programRepository.CreateProgram(program);

return CreatedAtAction(nameof(GetProgramById), new { id = program.Id }, program);

}

[HttpPut("{id}")]

public async Task<ActionResult> UpdateProgram(int id, [FromBody] Program program)

{

if (id != program.Id)

{

return BadRequest();

}

await \_programRepository.UpdateProgram(program);

return NoContent();

}

[HttpDelete("{id}")]

public async Task<ActionResult> DeleteProgram(int id)

{

await \_programRepository.DeleteProgram(id);

return NoContent();

}

}

To store the data in Azure Cosmos DB

To store the data in Azure Cosmos DB, you can use the following steps:

Create an Azure Cosmos DB account.

Create a new container in the Azure Cosmos DB account.

Install the Microsoft.Azure.Cosmos NuGet package.

Inject the CosmosClient class into the API controllers.

Use the CosmosClient class to perform the CRUD operations on the Azure Cosmos DB container.

For example, to implement the GetAllPrograms() method using Azure Cosmos DB, you would use the following code:

public async Task<ActionResult<IEnumerable<Program>>> GetAllPrograms()

{

var container = \_cosmosClient.GetContainer("MyDatabase