Technical Documentation for Viva Assessment

Developed by Yiorgos Dimitriadis | .NET 8.0 | C# | SQL Server | Clean Architecture

The project consists of four incremental tasks:

1. Compute the second largest integer from an input array.
2. Retrieve country data from the public RestCountries API.
3. Persist the retrieved data into a SQL Server database.
4. Implement a caching layer to improve response performance.

A brief description of the architectural style

The solution follows the Clean Architecture pattern, ensuring a clear separation of concerns, high testability, and maintainability.

Layers overview:

VivaAssessment.Api – Presentation Layer (controllers, Swagger, DI setup).

VivaAssessment.Application – Application Layer (services, Abstractions, DTOs).

VivaAssessment.Domain – Core domain models (Country, RequestObj).

VivaAssessment.Infrastructure – Infrastructure Layer (persistence, caching, and HTTP Clients.

This design allows the system to easily replace external dependencies (e.g., caching or data source) without affecting business logic.

Functional Description

Q1. Second Largest Number

Endpoint: /api/Numbers/second-largest

Accepting a JSON body, i.e.

```
{
 "requestArrayObj": [10, 3, 25, 7, 25, 18]
}
```

Response body

```
{
 "message": "the second largest number:",
 "value": 18
}
```

Logic:

- Uses LINQ (Distinct, OrderByDescending)
- Handles edge cases (null input, fewer than two elements)

Q2. Retrieve Countries from external source (RestCountries API)

Endpoint: GET /api/Countries

The source: https://restcountries.com/v3.1/all?fields=name,capital,borders

Maps results into:

```
public sealed record RestCountryDto(
    string Name,
    IReadOnlyList<string> Capital,
    IReadOnlyList<string> Borders
);
```

Uses a lightweight HttpClient-based abstraction (IRestCountriesClient) to ensure separation from the API.

Q3. Database Integration

Adds persistence layer using Dapper and SQL Server.

Two tables are created:

- Countries
- CountryBorders (1-to-many relationship with Countries)

Data flow:

1. Retrieve from API
2. Map DTOs → Entities
3. Save to SQL Server via repository layer

Q4. Caching Layer

Introduces an in-memory caching mechanism using IMemoryCache.

Retrieval order:

1. Check cache first.
2. If not found → check database.
3. If still not found → call RestCountries API, then save results to both DB and cache.

This improves performance and reduces redundant external API calls.

Dependency Injection Setup > Program.cs

```
builder.Services.AddApplication();//registers mappings and core services.
builder.Services.AddInfrastructure(builder.Configuration); //registers persistence, caching, and HTTP client.
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
```

Database

The database schema uses proper foreign keys and unique constraints to ensure referential integrity and prevent duplicate records.

Connection: Configurable via appsettings.json.

The connection string:

```
"ConnectionStrings": {
  "SqlServer":
"Server=YiorgosThnkCntr;Database=Northwind;Trusted_Connection=True;Multiple
ActiveResultSets=true;Encrypt=False"
}
```

The two SQL server tables:

```
CREATE TABLE dbo.Countries(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    CommonName NVARCHAR(200) NOT NULL CONSTRAINT
UQ_Countries_CommonName UNIQUE,
    Capital NVARCHAR(200) NULL
);

CREATE TABLE dbo.CountryBorders(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    CountryId INT NOT NULL
        CONSTRAINT FK_CountryBorders_Countries
        REFERENCES dbo.Countries(Id) ON DELETE CASCADE,
    BorderCode NVARCHAR(10) NOT NULL,
    CONSTRAINT UQ_CountryBorders UNIQUE (CountryId, BorderCode)
);

-- optional indexes, faster joins.
CREATE INDEX IX_CountryBorders_CountryId ON
dbo.CountryBorders(CountryId);
```

The select query:

```
SELECT
```

```
    t.CommonName,
    t.Capital,
    t.Id,
    u.BorderCode
FROM dbo.Countries AS t
INNER JOIN dbo.CountryBorders AS u
    ON t.Id = u.CountryId
ORDER BY
    t.CommonName ASC, u.BorderCode ASC;
```

Unit Testing (Optional)

Includes a basic unit testing for the numeric logic (SecondLargestFinder) using xUnit to demonstrate correctness and edge-case handling.

Test coverage:

- Normal case (distinct values)
- Duplicate values
- Negative numbers
- Null or empty input validation

How to Run

- Clone the repository.
- Update the SQL connection string in appsettings.json.
- Apply the SQL script to create tables.
- Run the API using Visual Studio (F5) or:

```
dotnet run --project VivaAssessment.Api
```

Opens Swagger UI: https://localhost:7187/swagger/index.html
or https://localhost:7187/index.html

Technology Stack

Category	Tool

Language	C# (.NET 8.0)
Framework	ASP.NET Core Web API
ORM	Dapper
Database	SQL Server
Caching	IMemoryCache
API Docs	Swagger / OpenAPI
Testing	xUnit

This documentation accompanies the Viva Assessment submission and describes the implementation details, architectural decisions, and setup instructions.