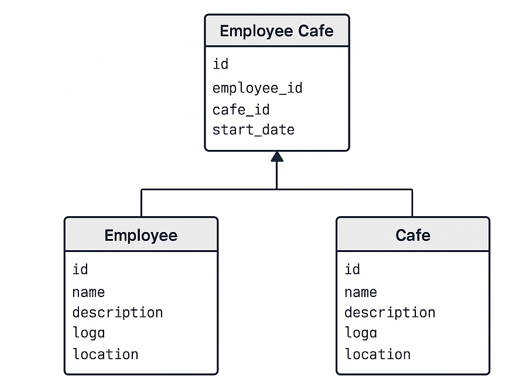
**Detail Design Document**

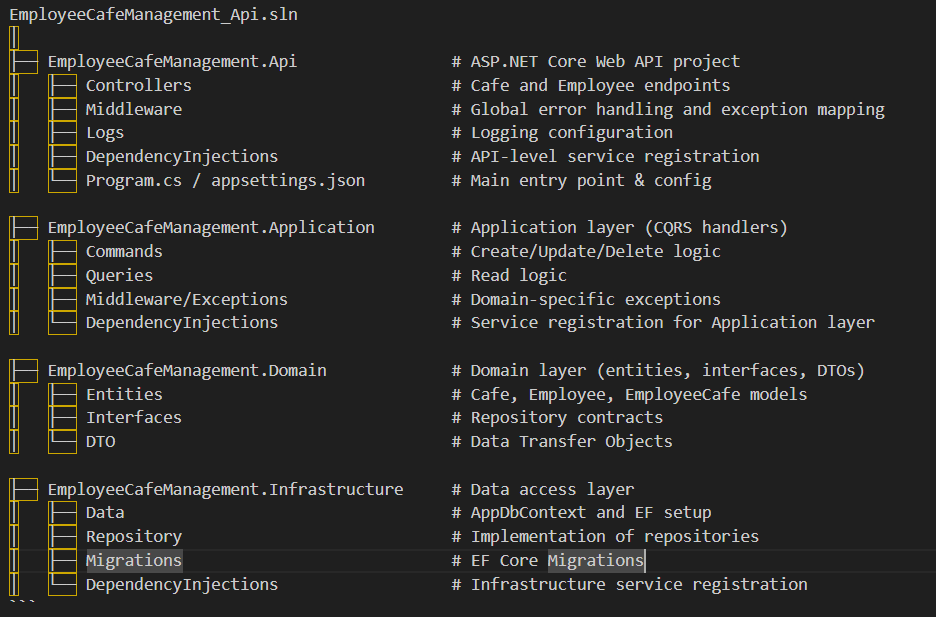
### Employee Cafe Management API

### 1. Overview

A RESTful API built with ASP.NET Core 8, applying Clean Architecture, CQRS, Entity Framework Core, and Dependency Injection to manage employee and café data.



### 2. Project Structure



### 3. Features

* CRUD operations for Cafés and Employees
* Assign employee to a single café (enforced constraint)
* Calculate `days worked` from `start date`
* Filter cafés by location, sorted by number of employees
* Filter employees by café, sorted by days worked
* Proper exception handling and logging
* Clean separation of concerns with CQRS and Domain-Driven Design

**4. Technologies**

* .NET 8
* ASP.NET Core Web API
* Entity Framework Core
* SQL Server
* MediatR (for CQRS)
* Dto’s
* XUnit
* FluentValidation
* Serilog (logging)

**5. API Endpoints**

#### Cafés

**GET** **/cafes?location=<location>**`

* Returns list of cafés filtered by location (optional), sorted by employee count

**POST /cafe**`

* Create a new café

**PUT /cafe**`

* Update café details

**DELETE /cafe**`

* Delete café and all assigned employees

#### Employees

**GET** **/employees?cafe=<cafe\_name>**`

* Returns list of employees (filtered by café), sorted by days worked

**POST /employee**`

* Create employee and assign to a café

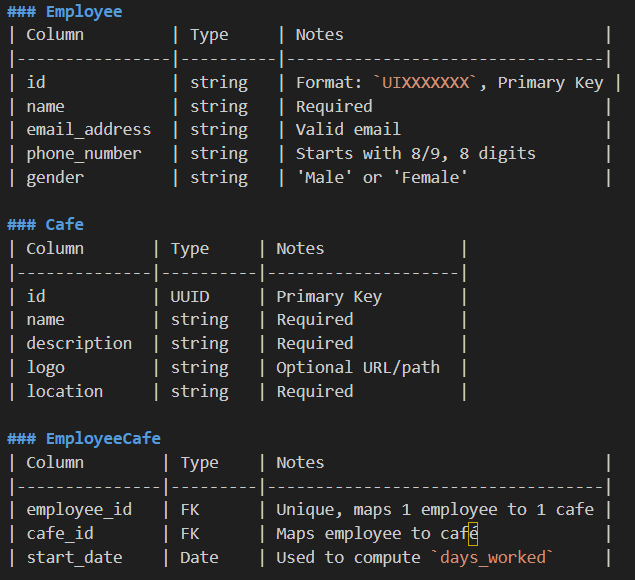
**PUT** **/employee**`

* Update employee details and café assignment

**DELETE** **/employee**`

* Delete employee

**6. Database Schema Summary**



**7.Seed Data**

Seed data is available in EF Core `AppDbContext` or you can manually populate the database as below.

Example Café SQL:

INSERT INTO Cafes (id, name, description, location) VALUES

(gen\_random\_uuid(), 'Cafe Mocha', 'Brewed delights', 'Orchard');

```

Example Employee SQL:

INSERT INTO Employees (id, name, email\_address, phone\_number, gender) VALUES

('UI1234ABC', 'Jane Tan', 'jane@example.com', '91234567', 'Female');

```

Employee–Café Assignment SQL:

INSERT INTO EmployeeCafe (employee\_id, cafe\_id, start\_date) VALUES

('UI1234ABC', '<cafe-uuid>', '2024-12-01');

```

**8. Dependencies**

* **AppDbContext**  
  Used for data access to Employees, Cafes, and EmployeesCafe.
* **RepositoryException**  
  Custom exception for wrapping lower-level exceptions with user-friendly messages.
* **Regex**  
  For enforcing strict validation rules on employee data (ID, email, phone, gender).

**9. Validation Rules**

| **Field** | **Rule** |
| --- | --- |
| Employee ID | Must start with UI followed by 7 alphanumeric characters |
| Phone Number | Must be 8 digits and start with 8 or 9 |
| Email Address | Must follow standard email pattern |
| Gender | Must be either "Male" or "Female" |
| Start Date | Must not be in the future |

**10. Error Handling**

* All methods wrap their logic in try-catch blocks (except for validation failures)
* If a failure occurs, a RepositoryException is thrown with a descriptive error message.
* The RepositoryException is caught by a global middleware (GlobalExceptionHandler) that translates it into HTTP error responses.

**11. Integration with Middleware**

This repository works in tandem with:

**GlobalExceptionMiddleware**

* Catches RepositoryException
* Logs the message and inner exception if available
* Returns standardized error responses

**Sample Output**

{

"message": "Failed to delete cafe with ID: 123",

"details": "System.InvalidOperationException: Cannot delete..."

}

**12. Testability**

* Interface-driven (ICafeRepository& IEmployeeRepository) enables mocking and testing
* Suggest using xUnit with Moq and EF Core InMemory for repository tests

**13. Key Design Decisions**

* **Manual Validation**: Validation is done inside repository using Regex, keeping logic self-contained.
* **DTO Usage**: Accepts EmployeeDto for insert operations to decouple input models from EF entities.
* **Separation of Concerns**: Repository does not deal with HTTP or UI concerns—only data logic.
* **Atomic Save**: All insert and update operations complete in a single SaveChangesAsync() call.