AUTODESK®

Overview

Senior Developer Advocate position.

Luis Nino

Agenda

01 Dot NET Architectural Overview

02 Dot NET Design Patterns & Principles

03 Infrastructure & Integrations

04 Scalability & Dot NET Best Practices

05 Scalability & .NET Best Practices

06 Questions

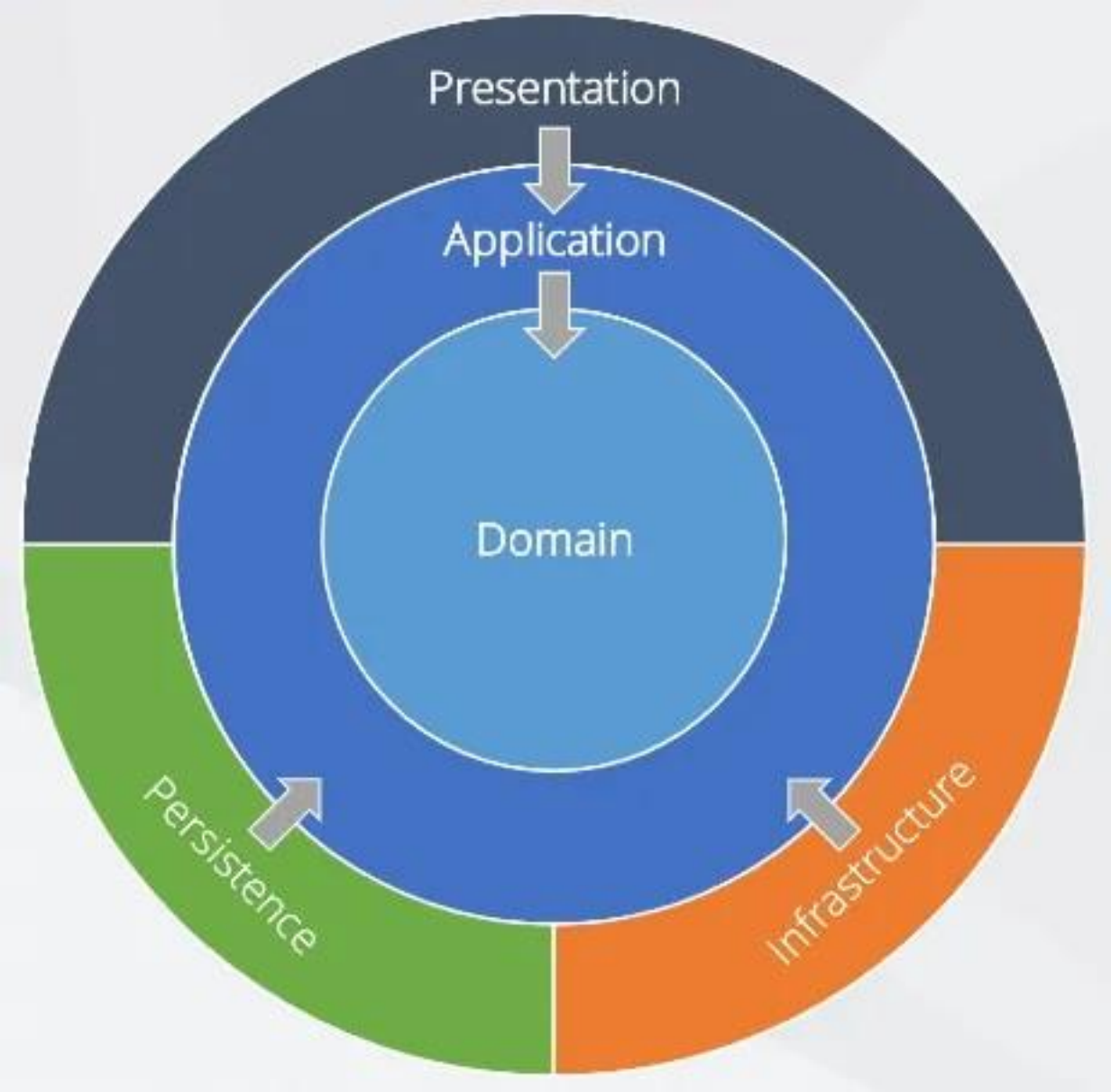
Dot NET Architectural Overview

Dot NET Architectural Overview

This solution follows a classic Clean Architecture (Onion) approach, decoupling layers by responsibility and direction of dependencies.

- Domain: core business entities and invariants
- Core: application use-cases and business rules
- Persistence: data access (EF Core repositories)
- Infrastructure: error strategies, interceptors, shared implementations
- Api: ASP.NET Core controllers, DI setup, middleware
- Frontend: Blazor WASM UI consuming the API
- Test: unit/integration tests spanning every layer

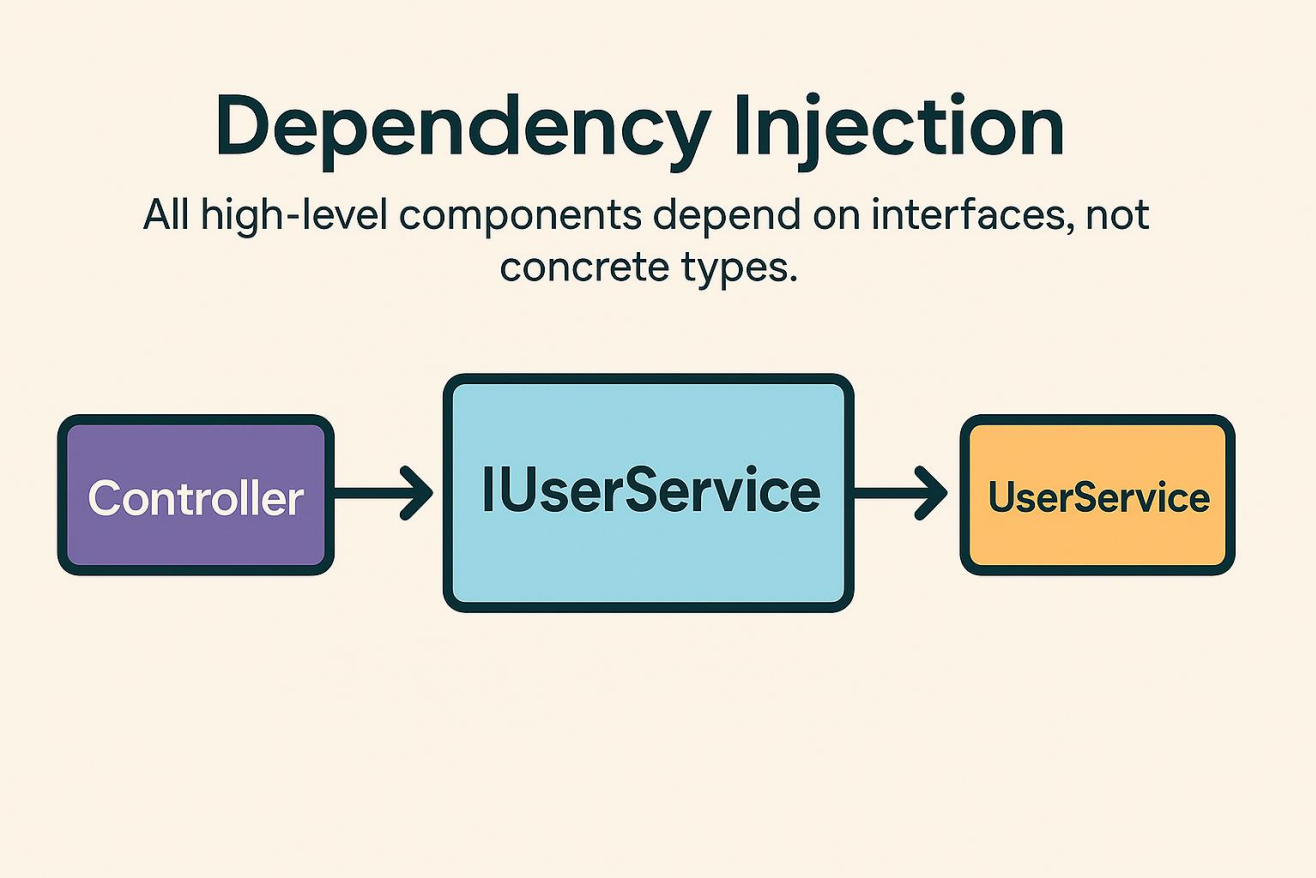
Dot NET Architectural Overview



Dot NET Design Patterns & Principles

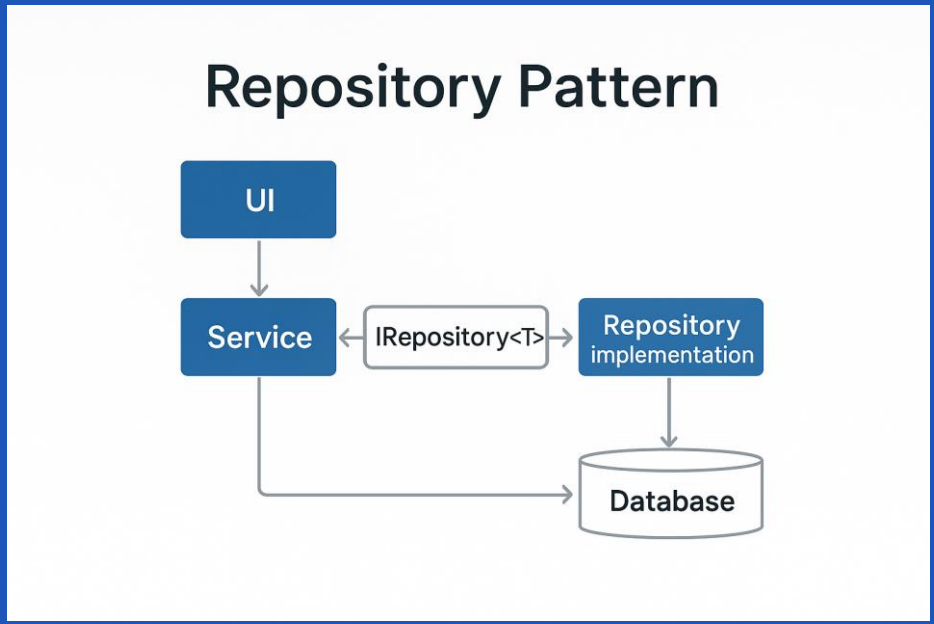
Dependency Injection

All high-level components depend on interfaces, not concrete types.



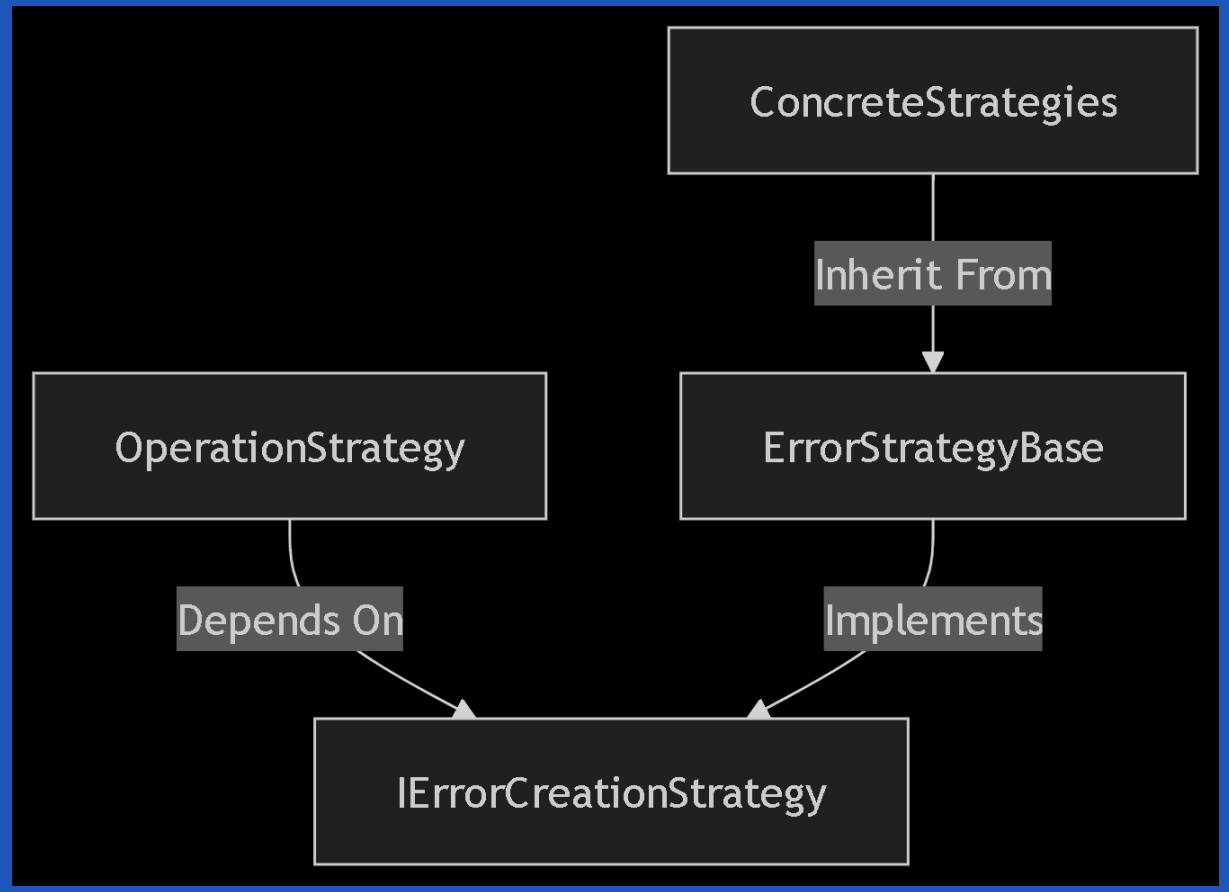
Repository Pattern

```
public class UserRead : IReadUsers
{
    // uses EF Core underneath but
    // Application layer only sees IUserRead
}
```



Strategy Pattern

All high-level components depend on interfaces, not concrete types.

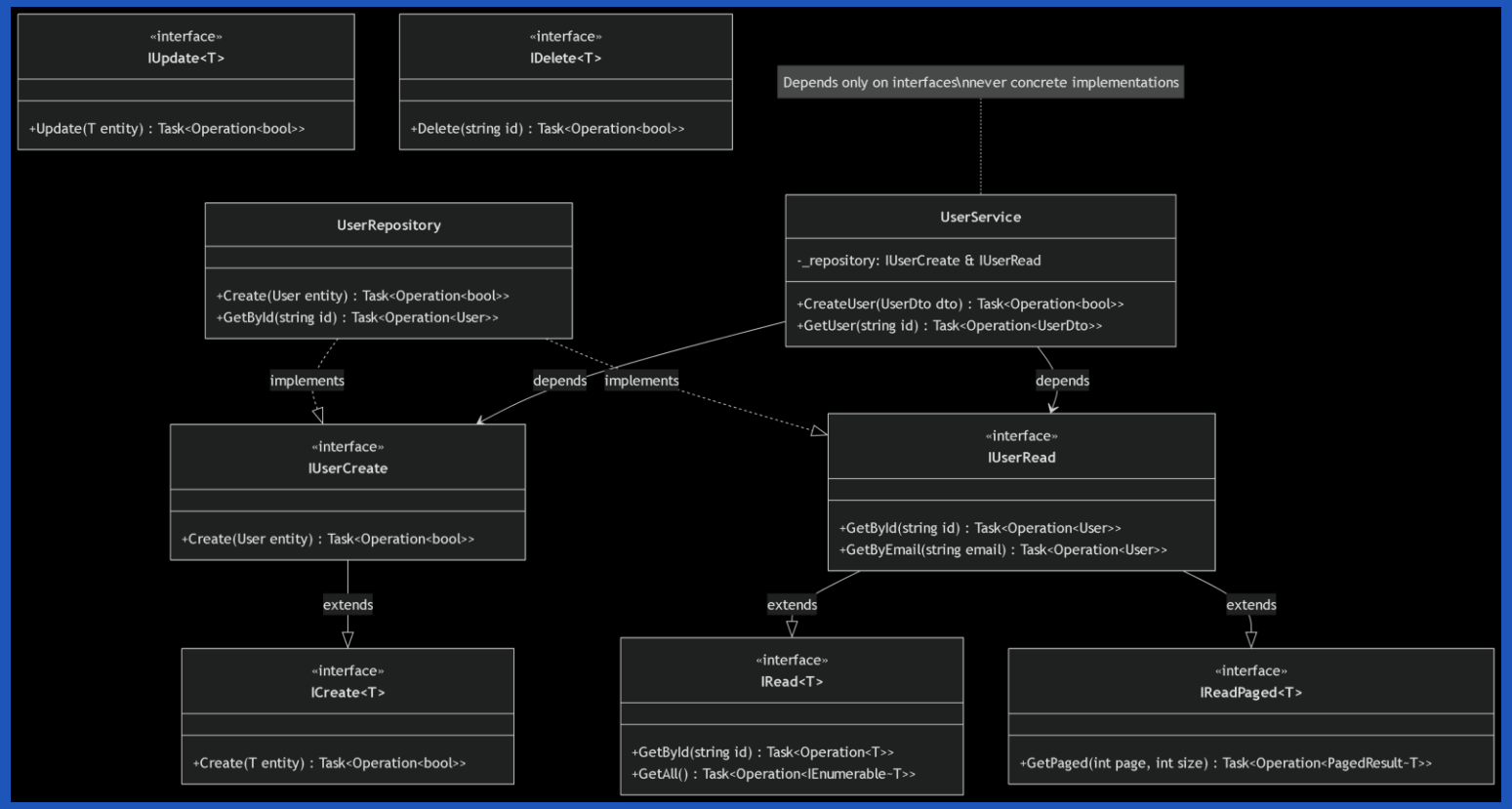


Template Method

All high-level components depend on interfaces, not concrete types.
in abstract repositories (CreateRepository<T>, UpdateRepository<T>) to wrap common flow and allow subclasses to override only CreateEntity / UpdateEntity.

CQRS-lite

Separate interfaces for Create/Read/Update/Delete (IUserCreate, IUserRead, etc.)

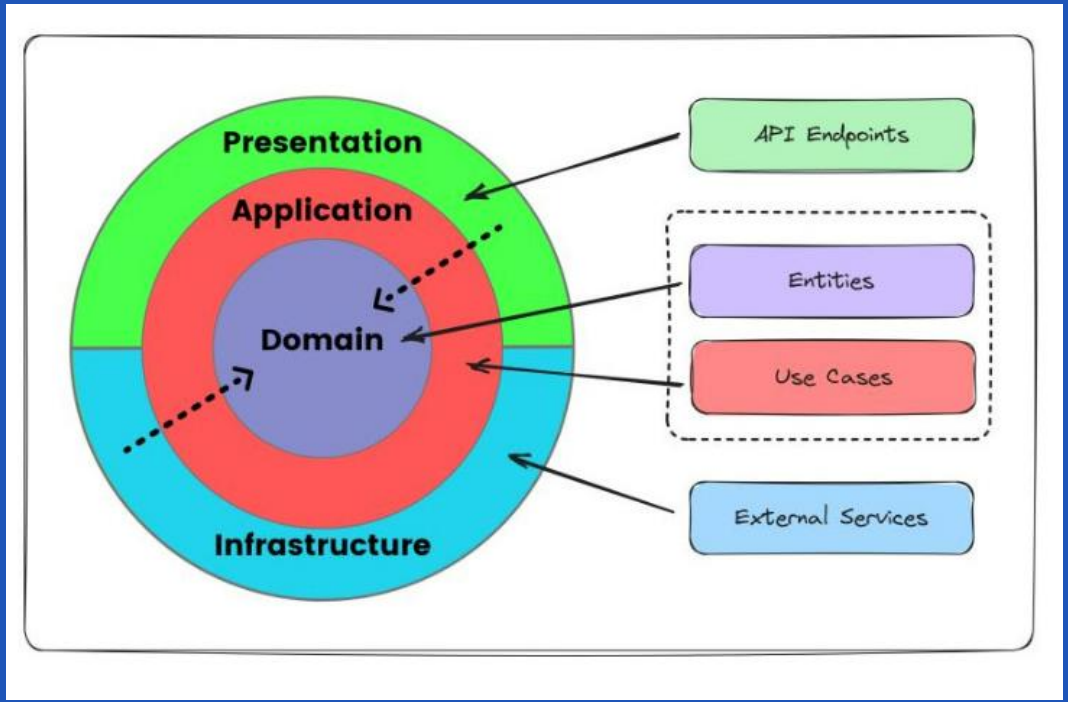


SOLID

- SRP: Each class has a clear responsibility (e.g. SqliteFunctionInterceptor only registers DB functions).
- OCP: Easy to extend new use-cases or error strategies without modifying existing code.
- LSP: Derived repository classes honor the base contracts.
- ISP: Interfaces are narrow (CRUD-specific).
- DIP: High-level modules (Controllers) rely on abstractions.

Domain-Driven Design (DDD)

- Entities with identity (Id) and invariants (via DataAnnotations).
- No anemic domain is present—most logic lives in Application/Infrastructure, so you could push more behavior into Domain if desired.



Infrastructure & Integrations

ORM & Persistence

- **Entities with identity (Id) and invariants (via DataAnnotations).**
- **DbFunction interceptor** registers an ordinal string comparison for cursor-based paging.
- **Migrations in Autodesk.Api.Migrations for schema evolution.**

Caching

- **IMemoryCache in UserRead.GetUsersPage(...) to cache paged results for 5 minutes.**

External Services

- **HttpClientFactory for outbound HTTP calls (builder.Services.AddHttpClient()).**

- **Swagger** for API documentation.

External Services

- **HttpClientFactory for outbound HTTP calls (builder.Services.AddHttpClient()).**

- **Swagger** for API documentation.

Scalability & .NET Best Practices

Asynchronous All the Way

- All I/O (DB, cache, HTTP) uses async/await, preventing thread-pool starvation.

Performance Recommendations

- Batching: Consider IAsyncEnumerable<T> for very large result sets.
- Distributed Cache: Migrate MemoryCache → RedisCache for load-balanced environments.
- Health Checks & Metrics: Add Microsoft.Extensions.Diagnostics.HealthChecks and Prometheus/App Metrics.
- Logging: Use structured logging (ILogger<T>) rather than Console.WriteLine.

Questions

Thank You