**Onion Architecture**

**Architecture**

Bir project’in (web, mobile, desktop və s.) əsas önəmli hissəsi onun back-end’dir. Biz elə bir back-end yazmalı, onu inkişaf etdirməliyik ki, flexible olmalı, yeniliklərə və ehtiyyac doğrultusunda dəyişikliklərə açıq olmalıdır. Hər hansısa dəyişiklik ediləcəyi zaman bu prosesi çətinləşdirən bir back-end olmamalıdır. Günümüzdə bir çox proyektlər sırf bir client’a özəl olaraq inkişaf etdirildiyi üçün onlar üzərində hər hansısa bir dəyişikliyi etmək çox resurs xərcinə səbəb olur. Müəyyən stukturlara (structure,architecture) uyğun olaraq və ya bir başa controller’in daxilində də bu project yazılıb inkişaf etdirilə bilər.

**N-Layer Architecture**

Bunu izzah etmək üçün ilk olaraq n-layer(n-tier və ya multi layer olaraq da bilinən) architecture haqqında danışaq. N olaraq adlandırılsa da əsasən 3 əsas layer’dən ibarət olur: PL - presentation(user interface) layer,

BL - business(logic) layer,

DAL - data access layer.

3 əsas layer daxilində daha da genişləndirilə bilər. Günümüzdə, keçmişdə yazılmış, inkşaf etdirilən və ya etdirilməyən bir sıra project’lər hələ də bu architecture’da davam edir. Yeni yazılan project’lər və ya keçmişdə yazılıb daha sonra başqa architecture’da yazılan project’lər artıq bu architecture’da yazılmır. Çünki bu architecture’ın istifadə etməməyimizin səbəbi, dəyişikliyə çox müqavimətli olması və bu şəkildə dizayn edilməsidir.

**Bəs nə üçün N-layer Architecture kifayət etmir?**

Bunun səbəbini tam anlamaq üçün yuxarı hissədə bəhs edilən müqavimətin səbəblərinə göz gəzdirək.

* Layer’lar arasında sıx əlaqə, asılılıq mövcuddur. Hər bir layer bir alt layer’dən asılı halda olur. Təbii ki, Onion architecture’da da bu asılılıq olacaq lakin Onion architecture’da olan asılılıq N-layer’dəki kimi dəyişikliyə müqvimət göstərəcək qədər güclü deyil.
* Kompleks olan project’lərdə yetərsizdir.
* Data Access layer N-layer architecture’da mərkəzi layer’dır. Beləliklə, application dizaynında data’ın gəlmə tərzinə bir asılılıq yaranır. Bu səbəbdən dataya əl çatanlıqla bağlı hər hansısa dəyişiklik yaranarsa bunun üçün çox böyük resurs ehtiyacı yaranır.

**Bəs nə üçün Onion Architecture(Clean Architecture)?**

**Onion Architecture istifadə edərək bu problemlər aradan necə qaldırılır?**

**Onion Architecture**

Bu arxitekturanın Onion(soğan) adlandırılmasının səbəbi, bu arxitektura izah edilən zaman çəkilən şəkil soğan halqasını anımsatmasıdır. Bu səbəbdən də bu arxitekturanın adı Onion olaraq adlandırılıb.

Mərkəzdən kənara doğru konstruksiya olunan və əlaqələri çox gevşək olan asılılığa sahib olan bir structure’dır.

Mərkəzdə Domain layer yer alır. Daha sonra bir üstdə yerləşən Application(Repository Interface & Service Interface də adlanır) layer yer alır. Ardı ilə eynigüclü olan Persistence və Infrastructure layer’lər və sonda Persistence və Infrastructure ilə eynigüclü olan Presentation layer yer alır.

Domain Layer - Onion Architecture’ın mərkəzi olan layer’dir. Burada domain object’lər, yəni Entity’lər, Value Object’lər, Enum’lar və Exception’lar(Entity’lərlə bağlı olan Exception’lar) yer alır.

Application Layer(Repository Interface & Service Interface) - Domain Layer ilə Persistence & Infrastructure Layer’lər arasındaki Abstraction’ı təmin edir. Burada Repository’lər və Service’lərin Abstract halları yəni Interface’ləri yer alır. Bu Layer’ın əsas məqsədi

A: Tight coupling refers to a situation where two or more software components are closely connected and depend on each other to function properly. Loose coupling, on the other hand, means that the components are less dependent on each other and can operate more independently.