Architecture Decision Record (ADR)

Pemilihan Struktur dan Pola Desain pada Sistem Akademik

1. Konteks dan Masalah

Sistem KRS awal dirancang menggunakan satu class utama bernama KRSManager yang menangani berbagai tanggung jawab: validasi SKS, pengecekan prasyarat, dan pendaftaran kelas. Pendekatan ini cepat dibuat, namun seiring pertumbuhan fitur, muncul masalah:

- Kode sulit diuji dan di-*maintain*.
- Ketergantungan antar class sangat tinggi.
- Sulit menambahkan validator atau aturan bisnis baru tanpa mengubah class inti.

Untuk mengatasi ini, tim melakukan refaktor menuju arsitektur yang mematuhi **prinsip SOLID** dan menghindari **God Object**.

2. Keputusan Arsitektur

Struktur class diubah dengan pendekatan **Service-Oriented + Interface-Based Design**, dengan inti keputusan sebagai berikut:

- Memisahkan logika bisnis menjadi class khusus seperti MaxSKSValidation, PrerequisiteValidation, dan DuplicateEnrollmentValidation.
- Menggunakan **interface** (IValidationRule, IValidationService, ICourseRepository, IEnrollmentRepository) agar ketergantungan dibalik sesuai **Dependency Inversion Principle**.
- Menambahkan KRSService sebagai *facade layer* yang mengorkestrasi proses validasi dan penyimpanan data, menggantikan KRSManager yang sebelumnya berperan sebagai *God Object*.

3. Alasan Pemilihan Struktur

Struktur ini dipilih karena:

- **Modularitas Tinggi:** Setiap class memiliki satu tanggung jawab spesifik (*Single Responsibility*).
- **Kemudahan Pengujian:** Dengan interface, setiap komponen bisa diuji secara terpisah menggunakan *mock dependency*.
- **Ekstensibilitas:** Validator baru dapat ditambahkan tanpa memodifikasi class lain.
- **Keterbacaan Kode:** Aliran bisnis lebih mudah dipahami melalui KRSService dan ValidationService.

4. Trade-off yang Dipertimbangkan

Aspek	Keuntungan	Kekurangan
Keterpisahan Komponen	Lebih mudah maintenance dan testing	Struktur class lebih kompleks
Penggunaan Interface	Meningkatkan fleksibilitas dan DIP	Membutuhkan boilerplate code tambahan
Refactor Anti-God Object	Menurunkan coupling dan risiko bug sistemik	Membutuhkan adaptasi developer baru
Pola Service-Oriented	Memungkinkan reuse & substitusi implementasi	Perlu dokumentasi arsitektur yang baik

5. Pola Desain yang Diterapkan

- **Factory Pattern** — digunakan secara implisit untuk membuat objek validator melalui ValidationService, tanpa ketergantungan langsung pada implementasi.
- **Strategy Pattern** — setiap validator (MaxSKSValidation, PrerequisiteValidation, dll.) adalah strategi berbeda untuk memvalidasi *enrollment*.
- **Dependency Injection (DIP)** — seluruh repository dan policy disuntikkan ke dalam service untuk menghindari *tight coupling*.
- **Facade Pattern** — KRSService menjadi pintu utama yang menyederhanakan interaksi antara modul mahasiswa, validasi, dan data repository.

6. Dampak Keputusan

Keputusan ini menghasilkan desain sistem yang lebih fleksibel, mudah diuji, dan siap dikembangkan untuk skenario lain (misalnya validasi lintas jurusan atau pembayaran online). Waktu pengembangan awal meningkat, namun biaya perawatan jangka panjang berkurang signifikan.