



Design Pattern

Gold - chapter 6 - Topic 1

**Selamat datang di Chapter 6 Topik 1 online
course Fullstack Web dari Binar Academy!**

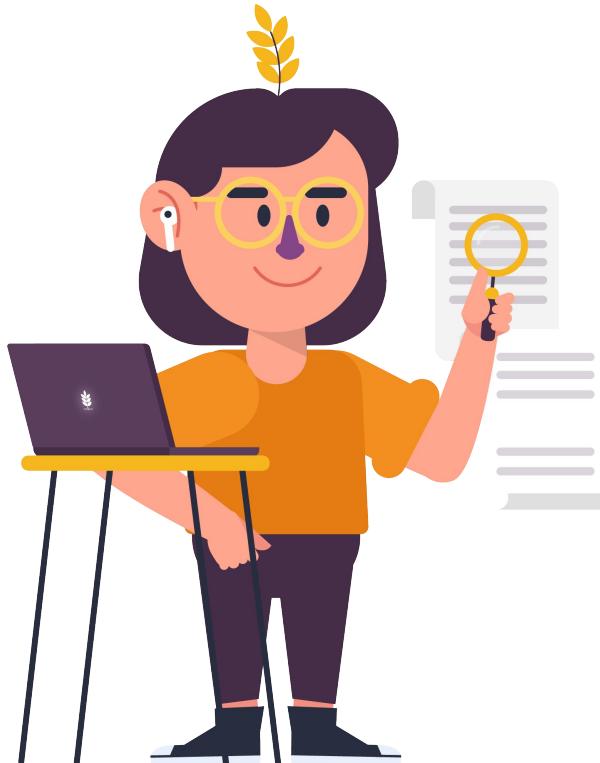




Sebagai pembuka course, chapter 6 bakal ngajak kamu buat **memahami cara merancang arsitektur dan dokumentasi API**, mulai dari pengenalan Design Pattern, Asynchronous process, membuat Authentication dan open API.

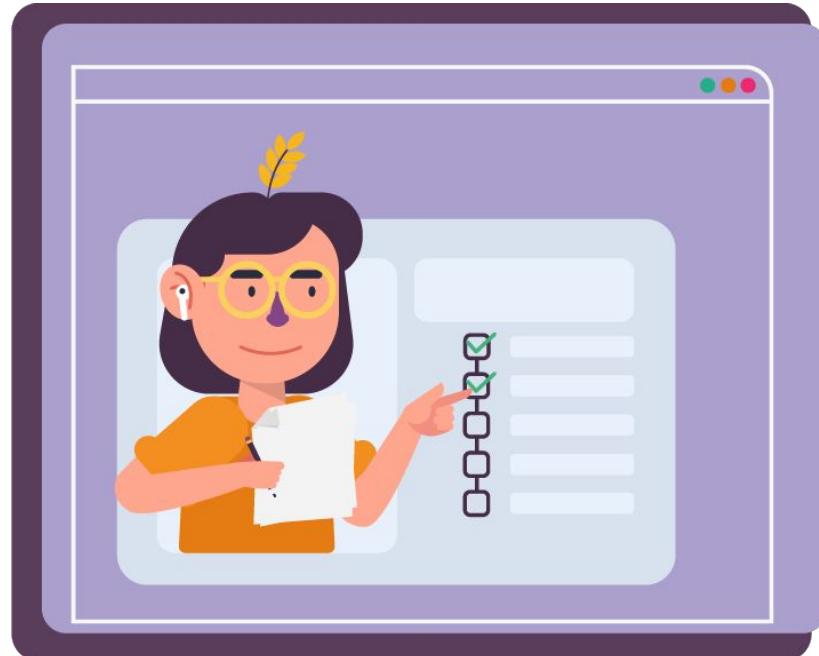
Di topik pertama ini, kita bakal bahas dulu tentang **apa itu Design Pattern, beserta macam-macamnya**.

Letsgowww~



Detailnya, kita bakal bahas hal-hal berikut ini:

- Mengenal apa itu Design Pattern
- Design pattern : MVC (Model View Controller)
- Design pattern : Service Repository Pattern
- Microservice Pattern





Apa itu **design pattern**?
Mungkinkah ada kaitannya dengan
aturan mendesain pola seperti saat
kita membatik gitu?



Pernah ga sih, kalian bikin sebuah aplikasi entah menggunakan Express.js maupun framework lain, dan kalian mikir,

“Kapan sih kita harus buat file baru untuk menulis kode kita? Ada nggak sih struktur folder yang bikin kita enak saat ngoding? Bisa gak sih kita mengelompokkan beberapa kode yang memiliki kemiripan dalam fungsinya? Gimana bikin codingan yang lebih konsisten?”

Yes, kalau bertanya-tanya hal itu maka jawabannya adalah dengan menggunakan **Design Pattern**





Apa itu Design Pattern?

Jadi design pattern itu adalah sebuah **pola dalam menulis dan membuat aplikasi**. Dimana dalam pola tersebut terdapat objek yang merepresentasikan sesuatu dengan fungsi / tanggung jawab yang spesifik.

Jadi ketika kita ingin mencari sebuah kode untuk suatu aksi tertentu, kita akan dengan mudah mencari dimana kode itu ditulis.





Kita bisa analogikan desain pattern itu kayak tata letak buku di Toko Buku.

Kalo kamu berkunjung ke Toko Buku, pasti kamu bisa lihat banyak rak-rak buku yang sudah ada labelnya masing-masing. Ada rak buku khusus Sains, ada rak buku khusus Sastra, dan sebagainya.

Nah label kayak gitu yang akan mempermudah pengunjung untuk mencari buku yang sesuai. Kebayang kalau enggak ada labelnya, pengunjung harus muter-muter satu gedung Toko Buku, untuk cari satu buku.

TOKO BUKU SAY





Design pattern ini sudah terbukti ampuh lho untuk memecahkan masalah-masalah yang umum. Jadi penting banget harus banget pake design pattern ini dalam membuat aplikasi.

Dalam sebuah repository atau source code, design pattern ini akan berbentuk folder dan file. Jadi output dari penggunaan design pattern ini adalah folder dan file dari source code kita akan lebih tertata dan berpola.



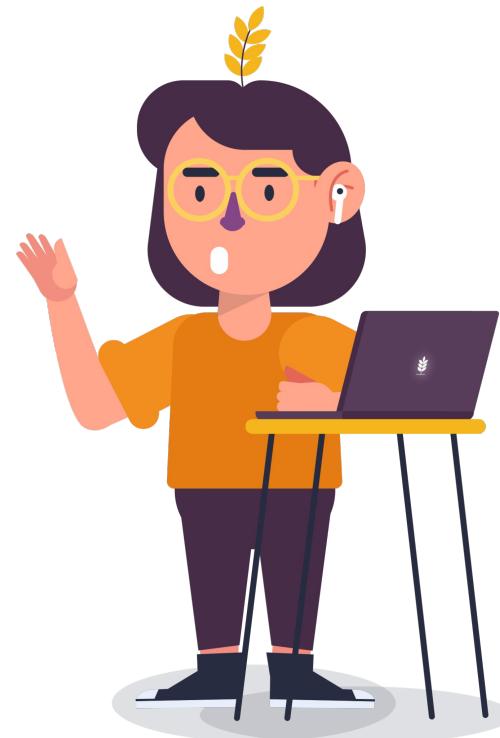
```
• • •  
.  
├── liberal-arts  
│   └── book-01.txt  
│   └── book-02.txt  
├── religion  
│   └── book-03.txt  
│   └── book-04.txt  
└── science  
    └── book-05.txt  
    └── book-06.txt  
  
3 directories, 6 files
```



Oh iya, di kesempatan yang sekarang, kita hanya akan banyak bahas penerapan Design Pattern di dalam Express.Js (back end), untuk sisanya kamu explore sendiri ya! 😊

Penjelasan design pattern pada Express.Js bukanlah pakem atau suatu keharusan, melainkan insight dari apa yang bisa kamu bisa lakukan dengan konsep design pattern ini.

Untuk penerapan yang baik dan benar seperti apa, akan sangat tergantung dari case setiap projek dan dan keputusan dari Principal Engineer kamu nantinya. Hehehe.





Oke, kita sudah tau apa itu Design Pattern ya.

Mungkin makin banyak pertanyaan juga, emang ada **arsitektur design pattern** apa aja sih yang bisa digunakan?

Kalau muncul pertanyaan itu dipikiran kamu, artinya kita sama! Yuk eksplor bareng-bareng





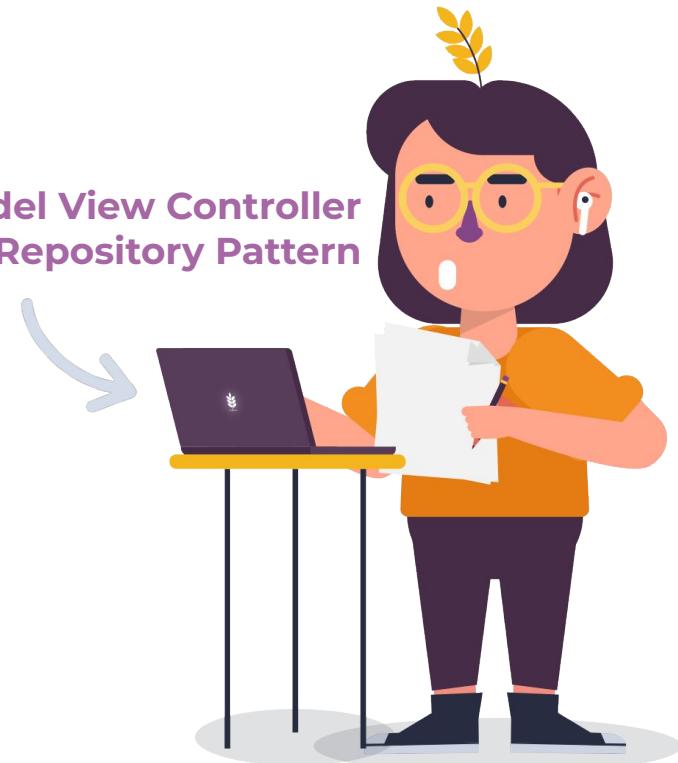
Jenis Arsitektur Design Pattern

Di dalam backend, ada beberapa design pattern yang umum digunakan di dalam industri, antara lain:

1. **Model View Controller (MVC)**
2. **Service Repository Pattern**

Langsung aja, kita lihat gimana sih kedua design pattern tersebut.

Model View Controller Service Repository Pattern





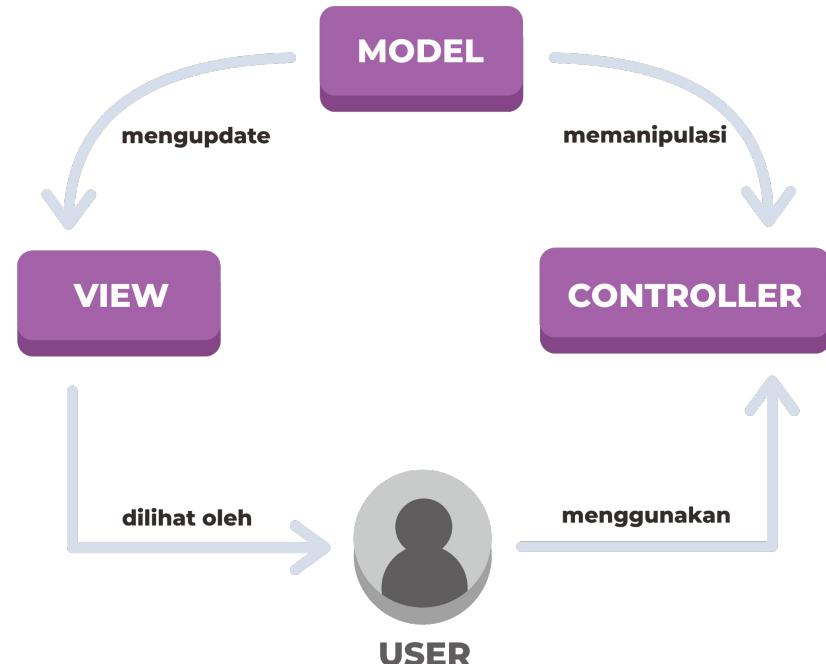
Model View Controller (MVC)

MVC adalah salah satu contoh design pattern yang dapat diimplementasikan di dalam Express.Js. Pattern ini memiliki 3 komponen utama :

Model → file yang digunakan untuk menyimpan kode yang berfungsi untuk mengolah data sesuai dengan nama modelnya.

View → file yang digunakan untuk mendefinisikan sebuah tampilan, baik dalam skema JSON, maupun HTML.

Controller → file yang digunakan untuk menyimpan kode yang digunakan sebagai penerima request yang masuk dan memberikan response.





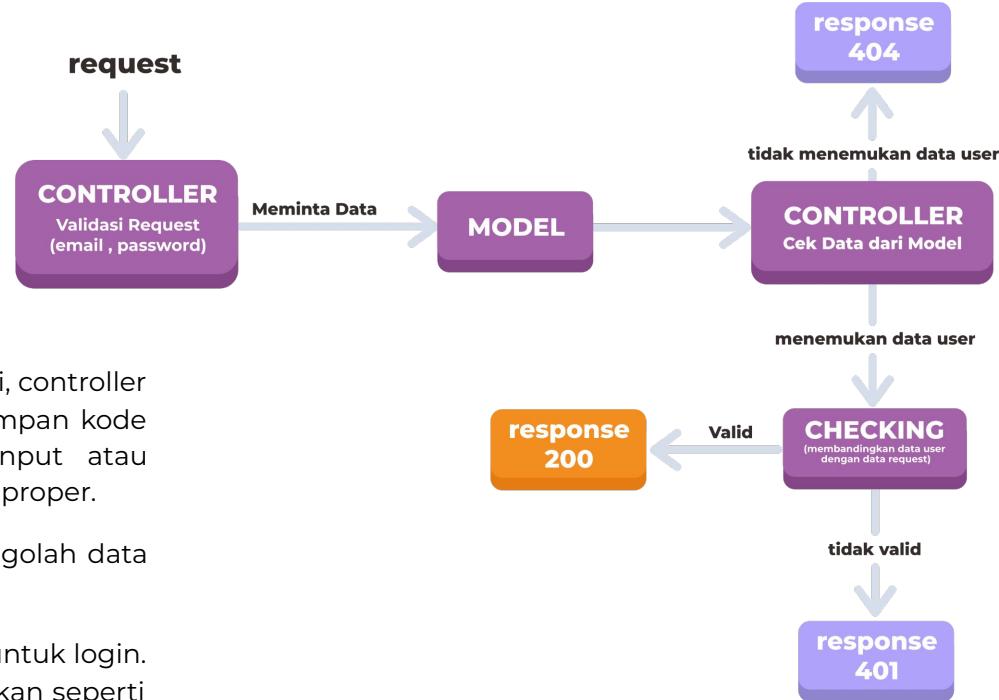
Ketika kita menggunakan pattern MVC maka struktur folder project kita kurang lebih akan kayak gini

Untuk mulai menggunakan design pattern ini, kamu bisa clone [repository berikut](#) sebagai boilerplate dari pattern ini, biar ga perlu bikin kode-kodenya dari nol banget xixixi~



```
controller
└── homeController.js
models
└── index.js
    └── user.js
package.json
views
└── index.ejs
yarn.lock

3 directories, 6 files
```



Controller

Seperti yang dijelaskan pada pengenalan MVC tadi, controller adalah sebuah file yang digunakan untuk menyimpan kode yang berfungsi untuk menghandle sebuah input atau request dari user, dan memberikan response yang proper.

Selain itu, controller ini juga bertugas untuk mengolah data sesuai dengan use case dari sebuah endpoint.

Sebagai contoh kita akan ambil contoh use case untuk login. Proses login kalau kita gambarkan kurang lebih akan seperti gambar disamping



Berikut contoh implementasi dari controller di dalam MVC yang memberikan response berupa HTML dari view engine.

```
module.exports = {
  index(req, res) {
    res.status(200).render("index");
  },
  onLost(req, res) {
    res.status(404).render("404", {
      url: req.url,
    });
  },
  onError(err, req, res, next) {
    res.status(500).render("500", {
      name: err.name,
      message: err.message,
    });
  },
};
```



```
● ● ●  
  
const { Post } =  
require("../..../models");  
  
module.exports = {  
  list(req, res) {  
    Post.findAll()  
      .then((posts) => {  
        res.status(200).json({  
          status: "OK",  
          data: {  
            posts,  
          },  
        });  
      })  
      .catch((err) => {  
        res.status(400).json({  
          status: "FAIL",  
          message: err.message,  
        });  
      });  
  },  
};
```

Dan berikut contoh implementasi controller di dalam MVC untuk menghandle request yang masuk ke dalam REST API



Di dalam pattern MVC ini, biasanya yang memiliki LOC (Line of Code) paling banyak adalah controller, karena semua business logic dari aplikasi ditulis disini.

Controller itu mirip lah sama controller PS, menerima input user dan ngasih feedback yang sesuai.

Jadi kalau kamu ingin mencari implementasi lebih lanjut dari sebuah endpoint itu gimana, dan logikanya gimana, kamu bisa langsung cek ke folder controllers yang tadi sudah dibuat.

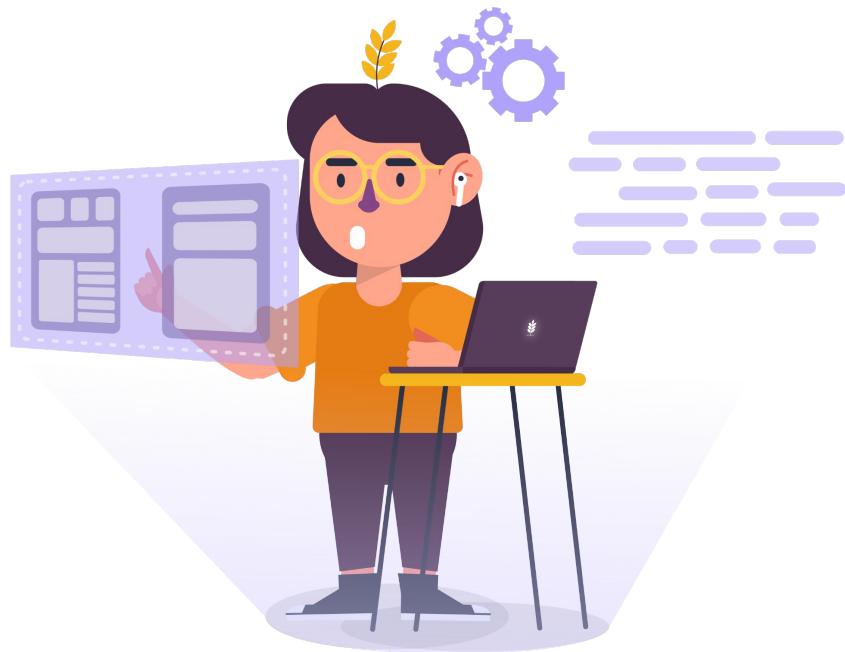




Model

Model adalah sebuah modul yang kita gunakan dalam **mendefinisikan entitas dari logika bisnis** kita.

Simpelnya adalah, dia ini representasi dari tabel di dalam database kita. Sebagai contoh, aplikasi kita memiliki resource **users**, **posts**, dan lain-lain. Maka dalam source code kita, pasti akan terdapat model **User**, dan model **Post**. Dimana kedua model ini akan digunakan untuk mengolah data terkait user dan post.





Beruntunglah kita udah belajar sequelize sebelumnya.
Kenapa?

Karena **sequelize sudah bisa handle model kita, jadi yang kita hanya perlu generate dan panggil modelnya**, tanpa harus pusing-pusing mikirin gimana caranya untuk mengimplementasikan koneksi ke database dan mengambil data secara spesifik.

Jadi, most of the time, kita cuma perlu mikirin controlernya aja.





Disamping ini adalah contoh model hasil generate dari sequelize



```
● ● ●

"use strict";
const { Model } = require("sequelize");
module.exports = (sequelize, DataTypes) => {
  class Post extends Model {
    static associate(models) {
      // define association here
    }
  }
  Post.init(
    {
      title: DataTypes.STRING,
      body: DataTypes.TEXT,
    },
    {
      sequelize,
      modelName: "Post",
    }
  );
  return Post;
};
```



View

View ini dapat digunakan untuk **menampilkan data dan memberikan interface untuk input data yang akan dikirim ke controller.**

Dari penjelasan tadi, kita tarik kesimpulan kegunaan dari view adalah:

1. Membuat tampilan HTML
2. Membuat form untuk diisi user
3. Membuat button agar user bisa berinteraksi dan disambungkan ke controller





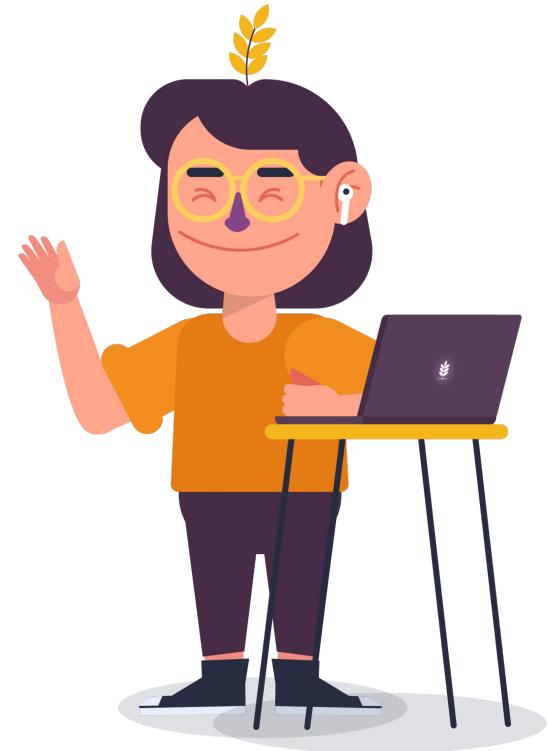
Mungkin kalian bertanya, "Berarti MVC ini hanya bisa dipake untuk Express.Js yang pake view engine aja ya?"

Wah, tidak bestie ~

View itu ga cuma sekedar tampilan HTML aja, tapi inti dari view adalah gimana caranya kita merepresentasikan sebuah data kepada pengguna HTTP Server kita. Kalau di dalam RESTful API, kita bisa membuat object view yang digunakan untuk mendefinisikan bentuk JSON yang akan diberikan kepada pengguna API kita. Object ini biasanya disebut sebagai Serializer, yang mana ya intinya dia itu View.

Tau kan gimana caranya ngolah data JSON? 😊

Ngomong-ngomon soal REST API, ada lho design pattern yang cocok banget untuk dipakai membuat REST API, nama patternnya adalah **Service Repository Pattern**.





Setelah bahas tentang MVC pada Express.Js, sekarang kita akan lanjut bahas Architecture Design Pattern lainnya , yaitu Service Repository Pattern

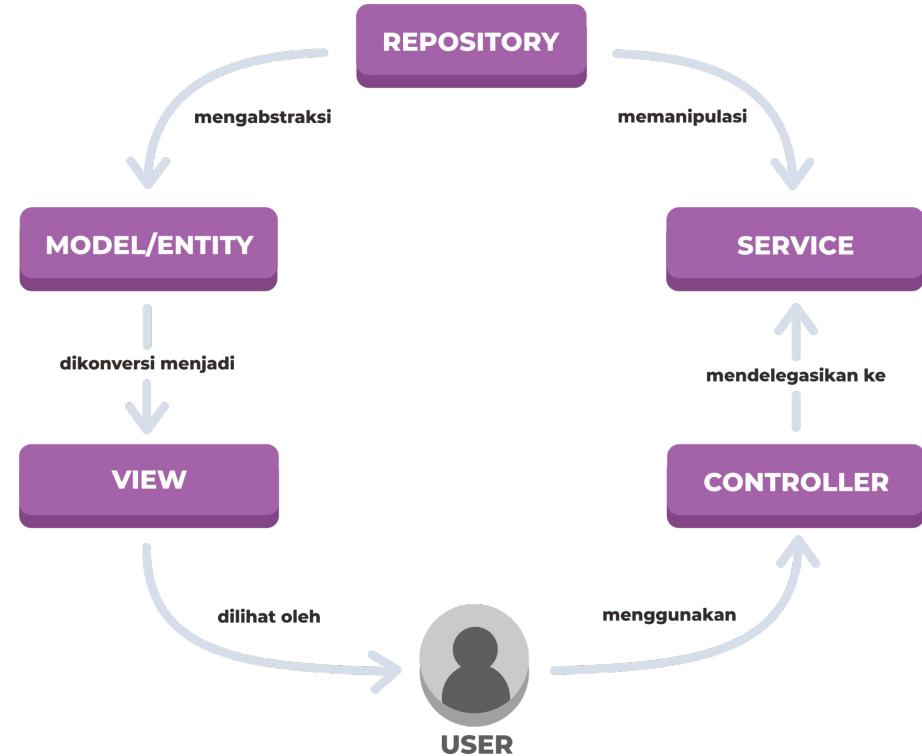
Yuk yuk yuk, bestie!



Service Repository Pattern

Pattern ini merupakan **ekstensi dari MVC**, yang sangat baik **digunakan ketika proses business logic dan use case suatu apps semakin banyak.**

Pada pattern ini kita menambahkan 2 layer tambahan di dalam aplikasi kita, yaitu Service dan Repository untuk membagi bobot kerja yang seharusnya dilakukan oleh controller



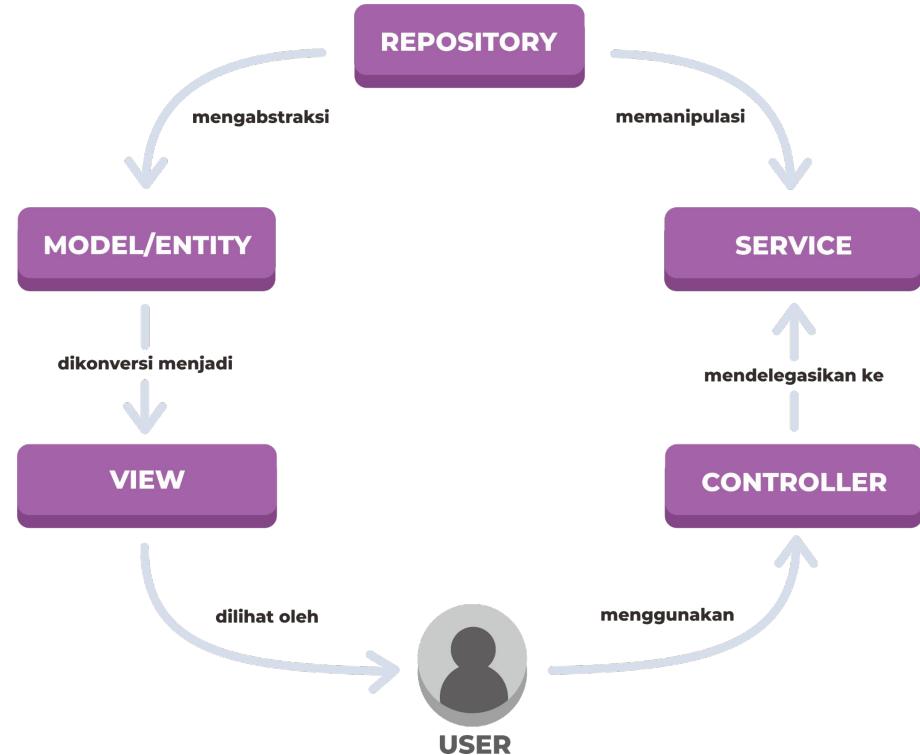


Service → file atau komponen ini digunakan untuk menyimpan logika bisnis dari sebuah aplikasi. Jadi di pattern ini bukan controller lagi yang handle.

Repository → file atau komponen ini, dipake buat mengabstraksi query ke dalam database, ya kita udah punya model, tapi model itu masih generic.

Oh iya, ga selamanya Service Repository Pattern ini dipakai bersamaan dengan MVC, bisa aja pattern ini dipake dengan kombinasi lainnya.

Untuk implementasinya kamu bisa cek link [disini](#)





Controller

Karena kita mengimplementasikan Service Repository Pattern, pastinya akan ada adjustment di dalam controller kita. Anggap saja kita mempunyai sebuah endpoint yang digunakan untuk mengambil semua data post, dan mendapatkan jumlah post yang ada di dalam database.

Maka dari itu kita perlu melakukan dua kali query, yaitu query untuk mengambil data post, dan query untuk mengambil jumlah post yang ada di dalam database.

Maka dari itu, controller kita akan terlihat sebagai berikut

```
function list(req, res) {
  postService
    .list()
    .then(({ data, count }) => {
      res.status(200).json({
        status: "OK",
        data: { posts: data },
        meta: { total: count,
      });
    })
    .catch((err) => {
      res.status(400).json({
        status: "FAIL",
        message: err.message,
      });
    });
}
```



Kalau kita bandingkan dengan controller kita sebelumnya, **controller kita saat ini tidak memanggil model secara langsung, melainkan ia hanya memanggil service.** Jadi controller tidak perlu memusingkan business logic lagi, ia hanya perlu menerima request, dan menyajikan response.

```
function list(req, res) {
  postService
    .list()
    .then(({ data, count }) => {
      res.status(200).json({
        status: "OK",
        data: { posts: data },
        meta: { total: count,
      });
    })
    .catch((err) => {
      res.status(400).json({
        status: "FAIL",
        message: err.message,
      });
    });
}
```



Service

Oke, untuk service, seperti yang sudah dijelaskan diatas, service digunakan untuk **menyimpan business logic**. Jadi logika untuk ngambil data post, dan ngambil total post yang ada di dalam database akan dihandle disini. Service ini biasanya mempunyai dependency ke Repository, tapi bisa aja dia berdiri sendiri tanpa repository.

Seperti yang bisa kita lihat, **hanya service saja yang memiliki akses untuk manggil repository**. Karena kalau kta panggil repository dari controller, akan membingungkan dan tidak menaati pola yang disepakati.



```
async function list() {
  try {
    const posts = await
postRepository.findAll();
    const postCount = await
postRepository.getTotalPost();

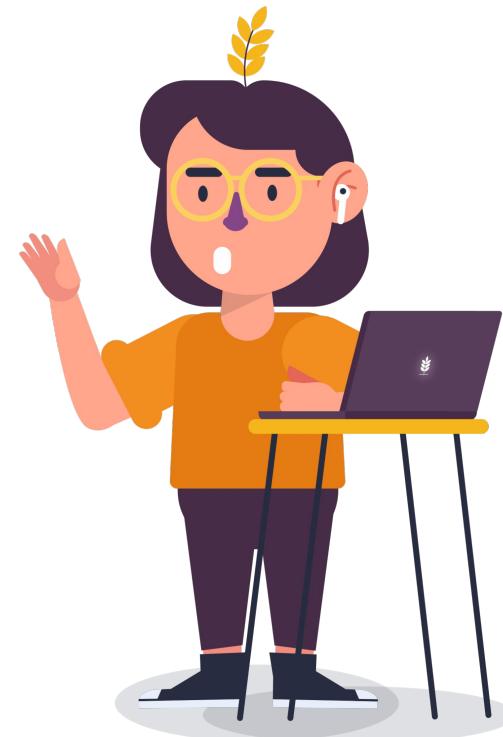
    return {
      data: posts,
      count: postCount,
    };
  } catch (err) {
    throw err;
  }
}
```



Repository

Jangan bingung dengan istilah repository, ini bukan repository Git ya!

Repository itu sebuah komponen yang **berguna untuk mengabstraksi query ke dalam database**, atau bisa dibilang juga repository ini dipakai untuk mengabstraksi penggunaan Entity Framework, yang dalam konteks Bootcamp kita ini, entity framework kita adalah sequelize.





Service tidak boleh mengakses sequelize atau model. Ketika service perlu mengakses model maka dia harus melalui Repository terlebih dahulu.

Jadi ketika kita ingin mencari sebuah query, kita hanya perlu lihat folder repository, dan dipastikan tidak terpencar-pencar ke service. Ini akan mengurangi dependency juga, jadi dari **semua kode yang ada di dalam source code, hanya repository aja yang memiliki dependency ke model**, (service sama controller ga perlu tau hehe)



```
function findAll() {  
    return Post.findAll();  
}  
  
function getTotalPost() {  
    return Post.count();  
}
```



Apa yang akan kita bahas selanjutnya, gak ada hubungan erat dengan apa yang kita bahas sebelumnya, yaitu design pattern.

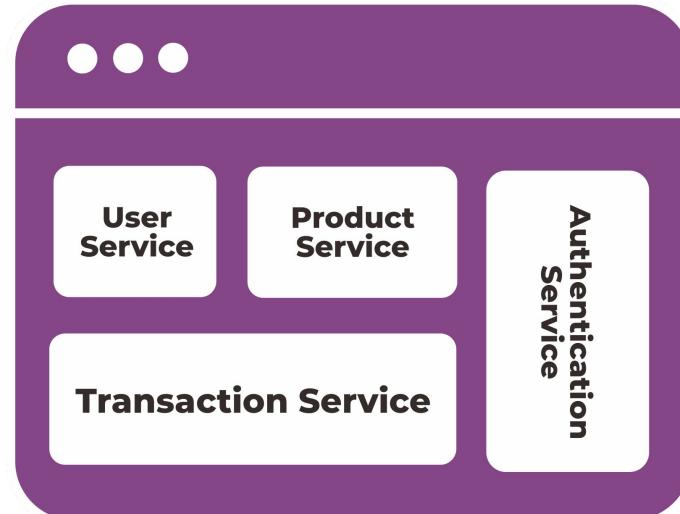
Meskipun sama-sama pattern, microservice pattern ini berbeda dengan design pattern.



Design pattern digunakan sebagai panduan dalam menulis kode, sedangkan microservice pattern ini lebih mengatur dari sisi architectural.

Microservice pattern akan sangat berguna ketika kita ingin membuat sebuah aplikasi yang terdiri dari beberapa bagian.

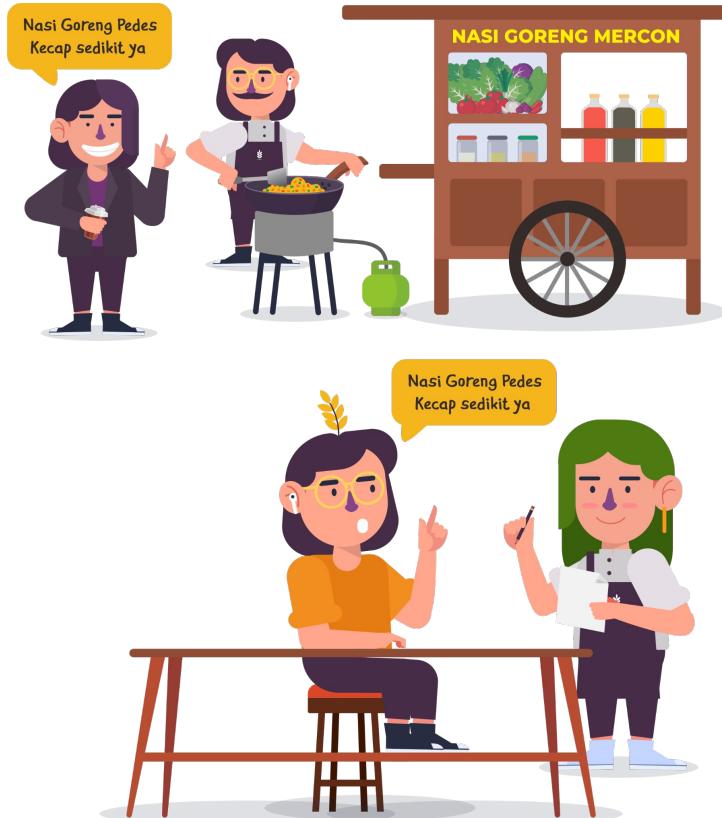
Toko Ungu Dot Com





Kalian masih inget abang nasi Goreng yang kita kenal di materi sebelumnya?

Bayangin usaha abang itu makin gede, makin banyak yang beli, tentu saja si abang nasi goreng ini ga bakal ngehandle semuanya sendirian. Dia butuh asisten yang bantu nerima pesenan, nyiapin tempat duduk dan sebagainya.





Nah... Dari gambaran Warung Abang Nasgor tadi berubah dari yang semula Monolithic **menjadi Microservices**. Dimana di warung tersebut **terdapat dua service yang berjalan**, yaitu abang nasgor sebagai tukang masak, sama asisten-asisten sebagai pelayan dari pembeli.

Meskipun ada 2 servis, tapi endingnya kan tetep aja dua servis itu bekerja untuk warung yang sama, yaitu Warung Nasgor si Abang.

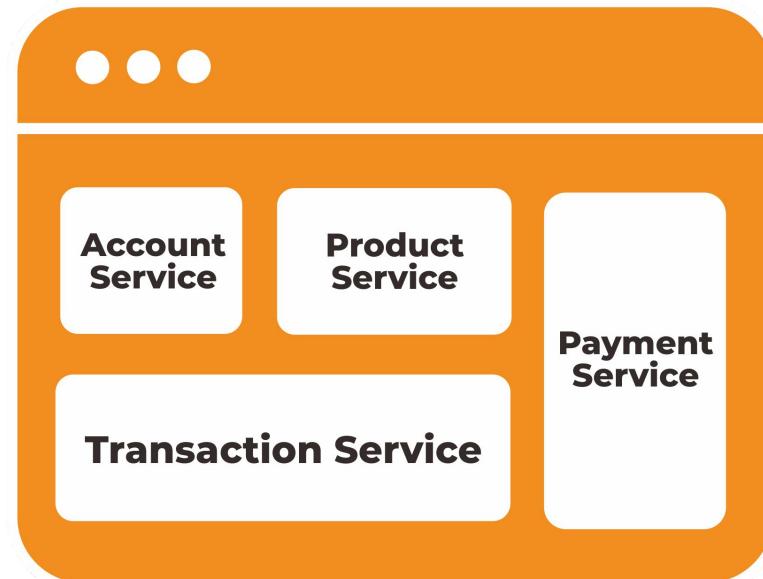




Kalau kamu penasaran sama implementasi microservice tuh kaya gimana, jadi tiap aplikasi yang kamu buat akan punya repositorynya masing-masing.

Sebagai contoh, kita punya situs belanja, biasanya situs belanja kalo menerapkan microservice pattern, akan terdapat beberapa servis, diantaranya:

- Account Service
- Product Service
- Transaction Service
- Payment Service

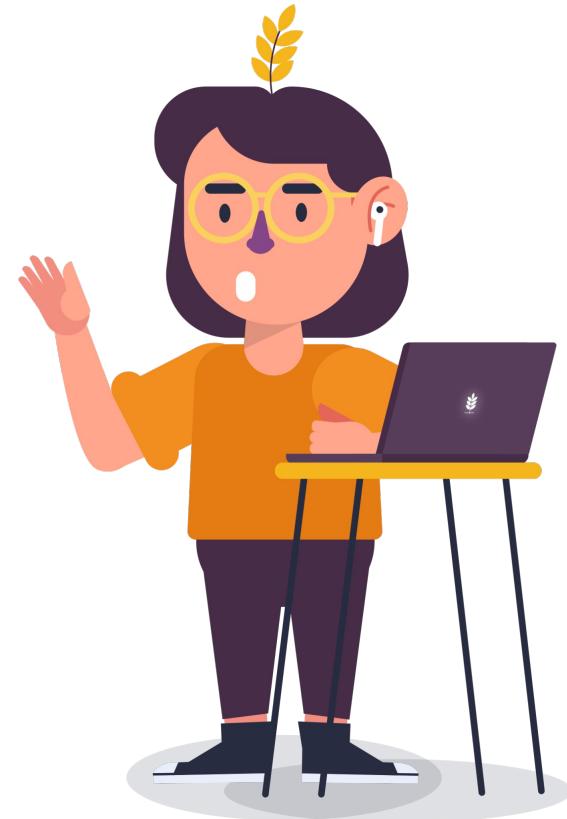


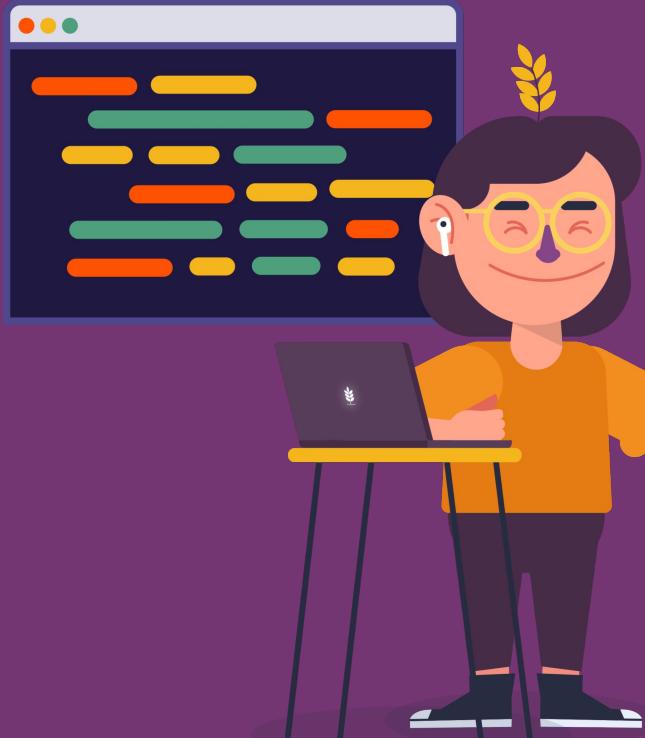


Karena tiap servis akan punya repositorynya masing-masing, bisa saja kode-kode yang ditulis akan menggunakan bahasa yang berbeda-beda. Tapi, nantinya tiap servis tersebut akan berkomunikasi menggunakan protocol yang sama, sebagai contoh HTTP.

Jadi kesimpulannya microservice ini enggak menuntut untuk menggunakan bahasa pemrograman yang spesifik, namun kita akan menggunakan suatu protokol dalam berkomunikasi.

Kalian bisa lihat [repository ini](#) kalo kepo sama pattern ini





Karena microservice ini tergolong sebagai materi yang advance, jadi kalian bisa eksplorasi secara mandiri ya untuk mengulik lebih dalam.

Referensi buat tambahan kamu belajar ~

- <https://medium.com/@ipenywis/what-is-the-mvc-cr-eating-a-node-js-express-mvc-application-da10625a4eda>
- <https://refactoring.guru/design-patterns>



Gimana nih materi pembuka di chapter ini?

Kamu merasa bisa menguasainya?

Kalau ada kesulitan untuk memahami, kamu bisa diskusi lagi di kelas bareng temen dan facil kamu ya!

See you di topik berikutnya ~



Terima Kasih!



Next Topic

loading...