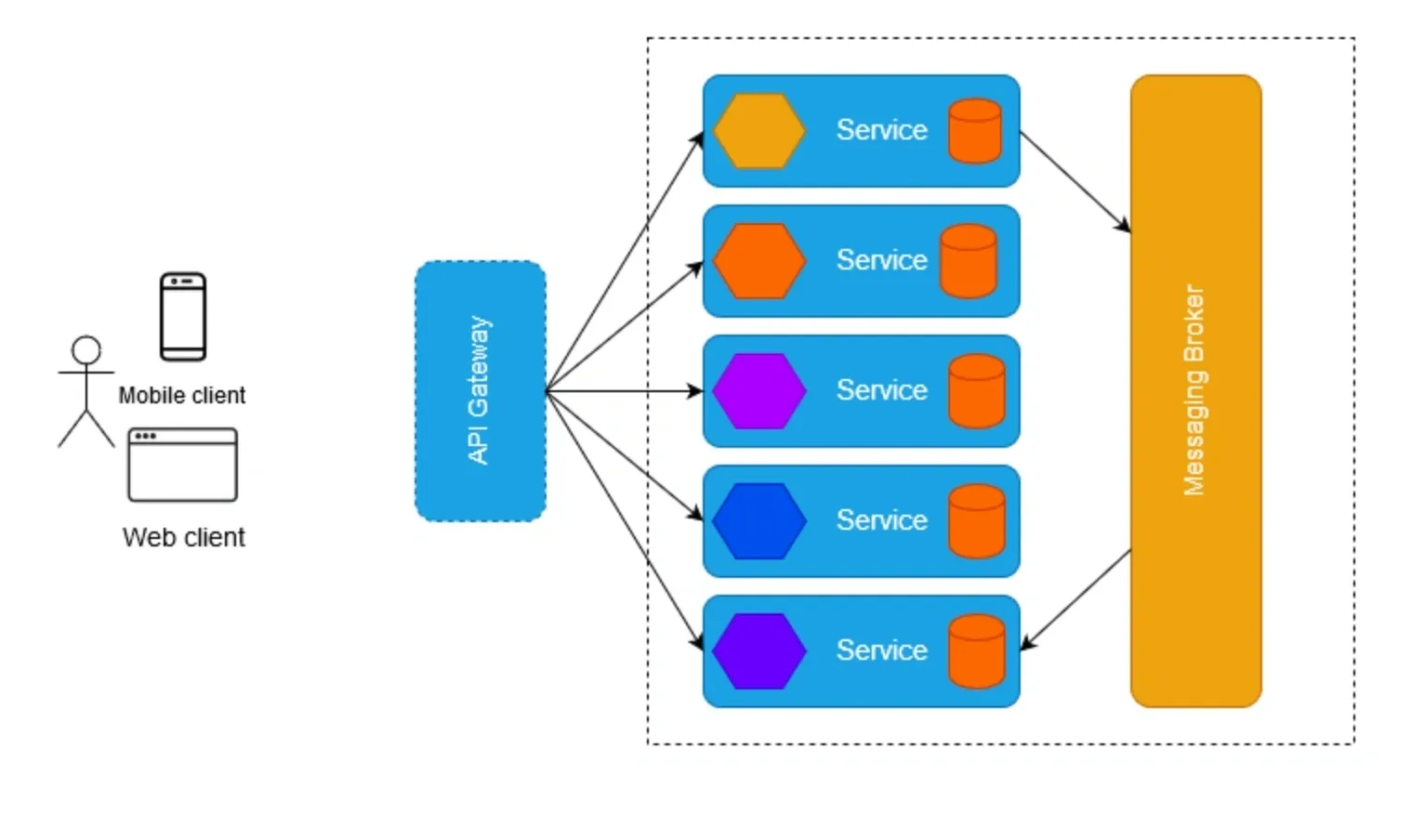
**BAB IV**

**MICROSERVICES**

1. **Microservices**

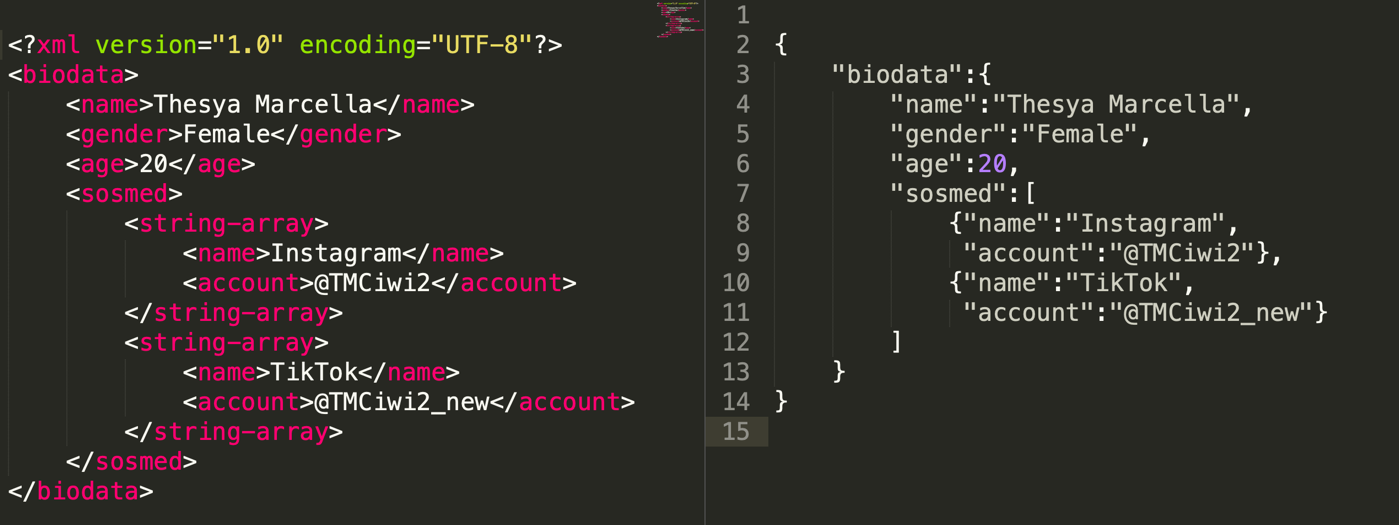
Microservices merupakan sebuah konsep arsitektur perangkat lunak yang dapat menjadi solusi untuk dapat membuat perangkat lunak yang besar lebih terorganisasi dan dapat dengan cepat beradaptasi terhadap perkembangan kebutuhan bisnis yang ada. Ada banyak solusi yang diusulkan oleh para peneliti untuk dapat meningkatkan kualitas *software*/sistem informasi dari sisi *nonfunctional requirements* (*scalability, reliability, maintainability* dan *availability*). Mulai dari penggunaan teknik *mirrorring*/DRC/*cloud*/sampai dengan pendekatan algoritma untuk dapat mengembalikan performa sistem setelah terjadi serangan. Microservice yang belum lama ini mulai populer di kalangan praktisi Software Engineering menawarkan pendekatan yang sedikit berbeda. Sistem Informasi enterprise yang pada umumnya dibangun dengan pendekatan monolitik (aplikasi terbungkus dalam satu package besar, dimana perubahan pada salah satu bagian kode program akan besar pengaruhnya terhadap kode program yang lainnya) digeser menjadi pendekatan terdisitribusi. Aplikasi dibagi menjadi bagian-bagian kecil yang berfungsi spesifik (high cohesion) dan tidak bergantung pada komponen program lainnya (loose coupling), dengan antarmuka API (Application Programming Interface).



***Gambar 1.a.*** *Microservices Arsitektur*

Dari sudut pandang paradigma *microservices*, sebuah konsep kunci dalam *resilient engineering* adalah penyekatan. Jika salah satu komponen dari sistem gagal, kegagalan tersebut tidak akan *cascade* / memberikan pengaruh ke kinerja komponen yang lain. Dengan demikian, masalah dapat terisolasi dan sisanya dari komponen-komponen sistem yang lain dapat terus bekerja. Dalam sistem monolitik, jika sebuah layanan gagal, maka semuanya berhenti bekerja. Sistem dapat dijalankan pada beberapa mesin yang redundan untuk mengurangi kesempatan kegagalan/*system failure*. Sebaliknya, dengan microservices dapat dibangun sistem yang bisa menangani kegagalan total layanan karena layanan fungsionalitas sistem telah tersekat dalam batas yang jelas.

Konsep microservices ini sedikit banyak berbeda dengan paradigma pendahulunya, yaitu *System Oriented Architecture* (SOA). Mulai dari penggunaan protokol komunikasi pesan yang lebih ringkas (REST, dll) dibandingkan dengan SOA yang banyak menggunakan XML SOAP, sampai dengan proses desain dan pembagian fungsional yang mengedepankan pembagian berdasarkan domain fungsionalitas pada organisasi.



***Gambar 1.b.*** *Perbedaan SOAP dan Rest API*

* 1. **SOAP**

*Simple Object Access Protocol* (SOAP) adalah protokol pesan yang memungkinkan elemen terdistribusi aplikasi untuk berkomunikasi, SOAP dapat dijalankan melalui berbagai jenis protokol tingkat rendah, termasuk HTTP (*Hypertext Transfer Protocol*) yang terkait dengan web. SOAP mendefinisikan struktur header yang mengidentifikasi tindakan-tindakan yang diharapkan dilakukan oleh berbagai node SOAP pada message, di samping struktur payload untuk membawa informasi. Konsep pemberian rute pesan melalui serangkaian node yang melakukan fungsi berbeda adalah bagaimana SOAP dapat mendukung hal-hal seperti addressing (pengalamatan), security (keamanan) dan format-independensi.

SOAP memicu peningkatan pesat “*web service* atau layanan web,” yang memiliki potensi untuk mengubah wajah pengembangan aplikasi di internet. Layanan ini menerbitkan antarmuka yang digunakan untuk berinteraksi dengannya di *Web Service Description Language* (WSDL), dan aplikasi lain dapat meminta layanan dengan membuat panggilan SOAP.

Diagram

Description automatically generated

***Gambar 1.1*** *SOAP Arsitektur*

1. Kelebihan SOAP

* Pesan SOAP didefinisikan pada level tinggi dalam XML, tetapi sebagian besar aplikasi SOAP menggunakan bahasa WSDL, yang ditulis dalam markup XML. Struktur XML dari SOAP membuatnya berguna untuk aplikasi yang mengharapkan informasi mereka disediakan dalam bentuk XML, dan fakta bahwa SOAP dapat menggunakan berbagai protokol jaringan, termasuk HTTP, berarti dia dapat dengan mudah melewati firewall, di mana protokol lain mungkin membutuhkan akomodasi khusus.
* Struktur data SOAP didasarkan pada XML, yang mirip dalam banyak hal dengan HTML yang digunakan untuk mendefinisikan halaman web. Seperti HTML, XLM sebagian besar dapat dibaca oleh manusia, yang membuatnya cukup mudah untuk memahami pesan SOAP, tetapi juga membuat pesan relatif besar dibandingkan dengan Common Object Request Broker Architecture (CORBA) dan protokol Remote Procedure Call (RPC) yang akan mengakomodasi data biner.
* Terkait keamanannya, SOAP mendefinisikan keamanannya sendiri yang dikenal sebagai web service security.
* SOAP juga terbilang independen dalam bahasa dan platform, layanan web SOAP dapat ditulis dalam bahasa pemrograman apa pun dan dieksekusi di platform apa pun.

1. Kekurangan SOAP

* Kerugian terbesar dari SOAP (dan SOA secara keseluruhan) adalah bahwa itu adalah protokol kelas berat untuk arsitektur kelas berat. Gagasan pesan yang melewati serangkaian node untuk diproses oleh masing-masing tampaknya mencampur protokol dan model arsitektur bus layanan untuk software (perangkat lunak), dan tidak satu pun dari keduanya dianggap optimal untuk pengembangan berbasis layanan mikro seperti yang populer digunakan saat ini.
* Terkait speed, mereka juga terbilang lambat, SOAP menggunakan format XML yang harus diuraikan untuk dibaca. Ini mendefinisikan banyak standar yang harus diikuti saat mengembangkan aplikasi SOAP. Jadi, SOAP dapat dikatakan lambat dan mengkonsumsi lebih banyak bandwidth dan sumber daya.
* Yang terakhir adalah Simple Object Access Protocol tergantung dengan WSDL. SOAP menggunakan WSDL dan tidak memiliki mekanisme lain untuk menemukan sebuah service (layanan).
  1. **RESTful APIs**

Sebaliknya, *REpresentational State Transfer* (REST) adalah model interaksi komputasi terdistribusi berdasarkan protokol HTTP dan cara kara kerja web server untuk mendukung client (klien). REST dijalankan melalui HTTP, dan hampir selalu menjadi dasar untuk pengembangan dan komunikasi layanan-mikro modern. REASTful APIs menggunakan permintaan HTTP untuk melakukan *get, put, post* dan *delete* data. REST/HTTP ini terbilang sederhana, fleksibel, ringan, dan menawarkan sedikit cara bertukar informasi.

Diagram

Description automatically generated

***Gambar 1.2.*** *SOAP vs Rest*

SOAP juga dapat menggunakan HTTP, namun dia menghubungkan elemen-elemen dari seperangkat alat komputasi terdistribusi yang kompleks (web service dan framework SOA) serta komponen aplikasi dan ini membentuk bagian dari kerangka kerja yang berorientasi layanan total.

1. RESTful

RESTful adalah antarmuka yang digunakan oleh dua sistem komputer untuk bertukar informasi secara aman melalui internet. Sebagian besar aplikasi bisnis harus berkomunikasi dengan aplikasi internal dan pihak ketiga lainnya untuk melakukan berbagai tugas. Misalnya, untuk menghasilkan slip gaji bulanan, sistem akun internal Anda harus berbagi data dengan sistem perbankan pelanggan Anda untuk mengotomatiskan tagihan dan berkomunikasi dengan aplikasi absensi internal. API RESTful mendukung pertukaran informasi ini karena mengikuti standar komunikasi perangkat lunak yang aman, andal, dan efisien.

1. API

*Application Programming Interface* (API) menentukan aturan yang harus Anda ikuti untuk berkomunikasi dengan sistem perangkat lunak lain. Developer mengekspos dan membuat API sehingga aplikasi lain dapat berkomunikasi dengan aplikasinya secara terprogram. Contohnya, aplikasi absensi mengekspos API yang meminta nama lengkap pegawai dan rentang tanggal. Saat menerima informasi ini, API memproses absensi pegawai secara internal dan mengembalikan jumlah jam kerja dalam rentang tanggal tersebut.

*Representational State Transfer* (REST) adalah arsitektur perangkat lunak yang memberlakukan syarat mengenai cara API bekerja. REST pada awalnya dibuat sebagai panduan untuk mengelola komunikasi pada jaringan kompleks seperti internet. Anda dapat menggunakan arsitektur berbasis REST untuk mendukung komunkasi berperforma tinggi dan andal sesuai skala. Anda dapat dengan mudah menerapkan dan memodifikasinya, membawa visibilitas dan portabilitas lintas platform ke semua sistem API. Developer API dapat merancang API menggunakan beberapa arsitektur yang berbeda. API yang mengikuti gaya arsitektur REST disebut sebagai API REST. Layanan web yang menerapkan arsitektur REST disebut sebagai layanan web RESTful. Istilah API RESTful umumnya merujuk pada API web RESTful. Namun, Anda dapat menggunakan istilah API REST dan API RESTful secara bergantian. Berikut adalah beberapa prinsip gaya arsitektur REST:

* Antarmuka yang seragam

Antarmuka seragam adalah desain fundamental dari semua layanan web RESTful. Ini mengindikasikan bahwa server mentransfer informasi dalam format standar. Sumber daya terformat disebut representasi dalam REST. Format ini dapat berbeda dari representasi internal sumber daya pada aplikasi server. Contohnya, server dapat menyimpan data sebagai teks tetapi mengirimkannya dalam format representasi HTML

* Statelessness

Dalam arsitektur REST, statelessness mengacu pada metode komunikasi tempat server menyelesaikan setiap permintaan klien secara independen dari semua permintaan sebelumnya. Klien dapat meminta sumber daya dalam perintah apa pun, dan setiap permintaan berada dalam status stateless atau terisolasi dari permintaan lain. Hambatan rancangan API REST ini menyiratkan bahwa server dapat sepenuhnya memahami dan memenuhi permintaan setiap waktu.

* Sistem berlapis

Dalam arsitektur sistem berlapis, klien dapat terhubung ke perantara resmi lain antara klien dan server, dan masih akan menerima respons dari server tersebut. Server juga dapat meneruskan permintaan ke server lain. Anda dapat merancang layanan web RESTful untuk berjalan pada beberapa server dengan beberapa lapisan seperti keamanan, aplikasi, dan logika bisnis, bekerja sama untuk memenuhi permintaan klien. Lapisan ini tetap terlihat oleh klien.

* Ketersinggahan (*Cacheability*)

Layanan web RESTful mendukung pembuatan *cache*, proses penyimpanan beberapa respons pada klien atau pada perantara untuk meningkatkan waktu respons server. Contohnya, bayangkan Anda mengunjungi situs web yang memiliki citra header dan footer umum pada setiap halaman. Setiap kali Anda mengunjungi halaman situs web baru, server harus mengirim ulang citra yang sama. Untuk menghindari hal ini, klien membuat cache atau menyimpan citra setelah respons pertama lalu menggunakan citra ini langsung dari cache tersebut. Layanan web RESTful mengontrol pembuatan *cache* dengan menggunakan respons API yang menentukannya apakah dapat dibuat cache atau tidak.

* Kode sesuai permintaan

Dalam gaya arsitektur REST, server dapat sementara memperluas atau menyesuaikan fungsionalitas klien dengan mentransfer kode pemrograman perangkat lunak ke klien. Contohnya, saat Anda mengisi formulir pendaftaran pada situs web, peramban Anda dengan segera menyoroti kesalahan yang Anda buat, seperti nomor telepon yang salah. Ini dapat terjadi karena kode yang dikirimkan oleh server.

1. **RESTful APIs Development**

REST telah menjadi standar *de-facto* dalam hal mengekspos data melalui API dan membangun layanan web. Faktanya, sebagian besar aplikasi web saat ini mengakses dan memaparkan data melalui REST API. Dengan popularitas *framework front-end* yang dapat menggunakan REST API dengan mudah, akan selalu menjadi nilai tambah bagi aplikasi web yang menampilkan REST API.

Berikut ini adalah contoh beberapa rest api yang dapat dibangun dengan beberapa Bahasa pemograman seperti:

1. **Rest API Python**

|  |
| --- |
| from flask import Flask, request, jsonify  app = Flask(\_\_name\_\_)  students = [  {"name": "Thesya Marcella", "gender": "Female", "age": 20}  ]  @app.get("/sample-api")  def get\_students():  return jsonify(students) |

Sumber: <https://flask.palletsprojects.com/en/2.2.x/>

1. **Rest API PHP**

|  |
| --- |
| public function sampleAPI(){  $data = array("name"=>"Thesya Marcella",  "gender"=>"Female",  "age"=>"20");  echo json\_encode($data);  } |

Sumber: <https://www.w3schools.com/php/func_json_encode.asp>

1. **Rest API JAVA**

|  |
| --- |
| @GetMapping("/sample-api")  public String getStudents() {  Map<String, String> map = new HashMap<>();  map.put("name", "Thesya Marcella");  map.put("gender", "Female");  map.put("age", "20");  String jobj = new Gson().toJson(map);  System.out.println(jobj);  return jobj;  } |

Sumber: <https://spring.io/learn>

1. **Rest API GoLang**

|  |
| --- |
| type person struct {  name string `json:"name"`  gender string `json:"gender"`  age int `json:"age"`  }  type allPerson []person  var persons = allPerson{  {  name: "Thesya Marcella",  gender: "Female",  age : 20,  },  }  func getStudents(w http.ResponseWriter, r \*http.Request) {  var newPerson person  reqBody, err := ioutil.ReadAll(r.Body)  if err != nil {  fmt.Fprintf(w, "No record found")  }    json.Unmarshal(reqBody, &newPerson)  persons = append(persons, newPerson)  w.WriteHeader(http.StatusCreated)  json.NewEncoder(w).Encode(newPerson)  } |

Sumber: <https://go.dev/doc/>

1. **Rest API Rust**

|  |
| --- |
| #[derive(Serialize)]  struct Person {  name: String,  gender: String,  age: Integer,  }  async fn get\_students() -> impl IntoResponse {  let person = Person {  name: "Thesya Marcella ".to\_string(),  gender: "Female".to\_string(),  age: 20,  };  (StatusCode::OK, Json(person)) } |

Sumber: <https://doc.rust-lang.org/book/>

1. **Output Rest Api**

Dari ke empat Bahasa pemograman diatas semua menghasilkan bentuk yang sama yaitu rest api dalam bentuk Object JSON:

|  |
| --- |
| {"name":"Thesya Marcella","gender":"Female","age":"20"} |

1. **Micro-Frontend**

Mikro-frontend adalah aplikasi kecil yang sebagian besar dibagi berdasarkan subdomain atau fungsionalitas yang bekerja sama untuk menghadirkan aplikasi yang lebih besar. Biasanya, proyek datang dalam berbagai ukuran dan kebutuhan yang berbeda. Jika proyek Anda cukup sederhana untuk memiliki dua atau tiga halaman dan dapat dikelola oleh satu tim, Anda tidak perlu menggunakan Micro-frontend ini. Anda bisa mengimplementasikan dengan framework pilihan Anda seperti Angular, React, atau Vuejs.

Tapi, ini tidak terjadi sepanjang waktu. Terkadang aplikasi frontend Anda mungkin merupakan bagian kecil dari aplikasi besar lainnya atau aplikasi terdiri dari banyak bagian dan fitur yang dikembangkan oleh tim yang berbeda atau aplikasi yang dirilis ke fitur produksi dengan fitur yang dikembangkan oleh tim terpisah. Jika berada dalam salah satu situasi ini maka memerlukan penggunaan Micro Frontends. Mari kita lihat pada diagram di bawah ini.

Diagram

Description automatically generated

***Gambar 3.a.*** *Arsitektur Micro-Frontend*

Seperti yang Anda lihat pada diagram di atas, disana memiliki 6 aplikasi frontend yang bekerja bersamaan untuk menghadirkan aplikasi besar. Komunikasi antar aplikasi ini dapat dilakukan dengan *event bus*, window object, atau metode publish/subscribe. Setiap aplikasi dapat diimplementasikan oleh tim terpisah dan kerangka kerja apa pun. Setiap aplikasi dapat berbicara dengan backend atau endpoint mereka satu per satu. Ada aplikasi bootstrap/launch yang memuat semua aplikasi dan mount dan unmount di DOM tergantung pada interaksi atau perutean pengguna.

Diagram

Description automatically generated

***Gambar 3.b.*** *Arsitektur Micro-Frontend*

Untuk menampilkan fitur-fitur yang dikembangkan oleh setiap tim, akan ada satu host yang bertugas menjadi perantara untuk menampilkan fitur-fitur tersebut. Host di sini dapat berupa halaman utama web atau container pada aplikasi yang bersangkutan. Jadi secara singkat, Micro FrontEnd adalah Arsitektur pada bagian FrontEnd aplikasi yang berpusat pada independensi suatu fitur dengan fitur lainnya.

1. Penerapan Micro-Frontend

Berikut ini adalah contoh memetakan beberapa squad dengan fitur dan tujuan yang berbeda-beda. Kesamaannya, pada setiap squad memakai satu design system yang sama. Dari design tersebut dapat membuat sistem Micro FrontEnd seperti berikut:

Diagram

Description automatically generated

Gambar 3.1. Kerangka Micro-Frontend

1. Reusables

Ini adalah komponen *reusable* yang paling kecil dan dipakai di seluruh proyek. Umumnya komponen ini adalah elemen dan input, seperti Button dan Textbox. Selain itu, bagian ini juga berisi *functions* yang dapat membantu *development* modul yang dibuat. Terakhir, terdapat juga static files seperti gambar dan asset icon.

1. Modules & Repository

Modules yaitu fitur yang dibuat, sedangkan repository adalah tempat penyimpanan proyek di mana modul-modul tersebut dikumpulkan dan ditampilkan menjadi sebuah kesatuan website yang besar. Jika dipermisalkan dengan contoh pada cerita di atas, modules adalah fitur dan halaman yang tampil, sedangkan repository adalah satu web yang menampung fitur-fitur tersebut. Modul ini terletak di dalam repository proyek yang dipakai di setiap tim.

Modul ini diletakkan di repository yang dimiliki oleh Squad yang membuat fitur tersebut. Misalkan, Squad 1 membuat modul yang akan ditampilkan di Dashboard 1, maka modul tersebut diletakkan di repository Squad 1. Bagaimana kalau modul yang dibuat oleh Squad 1 harus ditampilkan di Dashboard 2 yang dibuat oleh Squad 2?

Maka modul tersebut tetap akan dibuat di repository Squad 1, tetapi kami memakai Module Federation untuk membuat modul yang dibuat tersebut tampil di dashboard 2. Dengan begitu, setiap squad tidak akan menyentuh repository di squad lain hanya untuk menambah fitur baru.

1. Link

Jika diperhatikan, tiap repository memiliki satu URL yang sama dengan endpoint yang berbeda. Hal ini karena layout dari setiap proyek itu sama, hanya konten fiturnya saja yang berbeda.

Disini kalian bisa mengakalinya dengan dibuat seolah satu kesatuan, tetapi sebenarnya terpisah-pisah. Dampaknya sama jika dibuat dalam satu host untuk menampung proyek-proyek tersebut.

1. **Micro-Frontend menggunakan Microservices**

Sudah dibahas pada bagain sebelumnya bahwasannya Micro-Frontend salah satu framework yang dapat digunakan ialah React JS. Pada contoh kasus ini disini akan menggunakan Microservices dalam bentuk RESTfull APIs. Untuk memanfaatkan microservices sebagai bentuk dari backend pada sebuah aplikasi website, perlu menggunakan library tambahan yaitu AXIOS atau FETCH. Dalam contoh kasus disini akan menggunakan library AXIOS sebagai bentuk untuk menjembatani antara frontend dengan microservices.

1. **Installasi AXIOS**

Bukalah terminal project react anda dan masukan syntax dibawah ini untuk menginstall library AXIOS dengan npm:

|  |
| --- |
| npm i axios |

Setelah installasi AXIOS berhasil maka pada file package.json akan menyimpan dependencies axois dengan version terbaru. Dan jalankan project react anda dengan benar.

1. **Microservices RESTfull APIs**

Pada bagian ini akan mencontohkan bentuk dari microservices yang dibangun dengan menggunakan JAVA Spring Frameworks. File microservices dapat di download di <https://github.com/FebryFairuz/IBIK-20222023-GENAP-PWL/tree/master/PROJECT/my-app>/microservices. File microservices tersebut merupakan bentuk extracted file JAR dimana file tersebut dapat dijalankan tanpa perlu menginstallasi tool chain dari Spring Maven Framework. Namun perlu diperhatikan minimum JAVA yang digunakan ialah versi ~17.0.0.

Setelah berhasil mendownload bukalah terminal dan pastikan MYSQL anda sudah nyala dan buatlah sebuah database bernama *ibik\_pwl\_services*. Pada terminal anda masukan syntax dibawah ini untuk menjalan file microservices:

|  |
| --- |
| java -jar name-of-jar-file.jar |

Contoh:

Text

Description automatically generated

***Gambar 4.1.a.*** *Menjalankan microservices*

Pada gambar diatas adalah cara menjalankan atau menyalakan services anda, jika tidak terdapat pesan kesalahan dan menampilkan sebuah info Tomcat started on port(s): 8080 (http) maka microservices anda sudah berjalan dengan baik. Untuk mengetesnya silakan gunakan postman atau tools lainnya untuk menguji sebuah API. Dan url yang dapat diakses ialah localhost:8080.

Setelah berhasil menjalankan microservices, silakan cek kembali database ibik\_pwl\_services, maka akan menciptakan Entity Relationshio Diagram seperti dibawah ini:

Graphical user interface, text, application, chat or text message

Description automatically generated

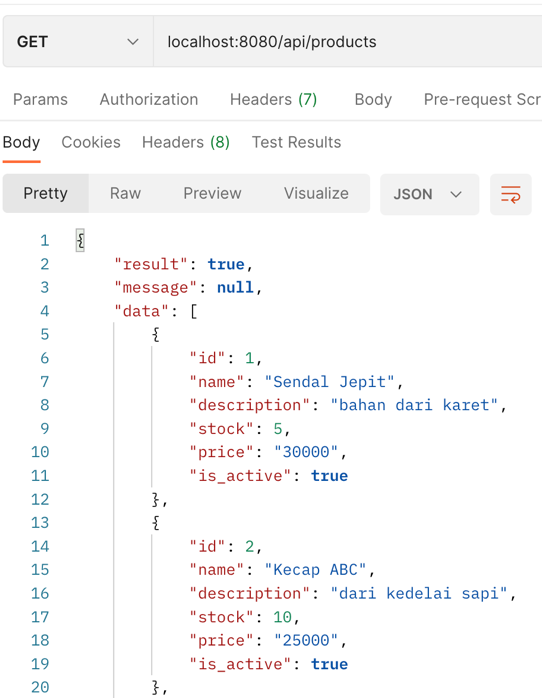
***Gambar 4.1.b.*** *ERD microservices*

Berikut ini adalah beberapa end-point services yang dapat anda gunakan untuk pengujian pada microservices yang telah anda download sebelumnya:

***Table 4.2.*** *Table End Point Microservices*

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Type Request** | **End-Point** | **Parameters** |
| Login | POST | api/auth/login | {  "email":"string", "password":"string"  } |
| Menampilkan seluruh data students | GET | api/students |  |
| Menampilkan data student berdasarkan id | GET | api/students/{id} | id |
| Menampilkan data student berdasarkan pencarian | POST | api/students/search | {  "searchKey":"string"  } |
| Menambahkan data student baru | POST | api/students | {  "npm":"string",  "firstname":"string",  "middlename":"string",  "lastname":"string",  "programs":{"id":int},  "programStudy":{"id":int}, "email":"string",  "birthdate":"string"  } |
| Memperbaharui data student | PUT | api/students | {  "id":int,  "npm":"string",  "firstname":"string",  "middlename":"string",  "lastname":"string",  "programs":{"id":int},  "programStudy":{"id":int}, "email":"string",  "birthdate":"string"  } |
| Menampilkan data product | GET | api/products |  |
| Menampilkan data product berdasarkan id | GET | api/products/{id} | Id |
| Menambahkan data product baru | POST | api/products | {  "name": "string",  "description": "string",  "stock": int,  "price": "string",  "is\_active": boolean  } |
| Memperbaharui data product | PUT | api/products | {  "id": int,  "name": "string",  "description": "string",  "stock": int,  "price": "string",  "is\_active": boolean  } |
| Menghapus data product | DELETE | api/products/{id} | id |
| Menampilkan keranjang belanja | GET | api/chart/mychart | AuthToken |
| Menambahkan product kedalam keranjang belanjaan | POST | api/chart/mychart | AuthToken,  {  "invoice\_code":"string",  "student\_id":int,  "product\_id":int,  "total":int  } |

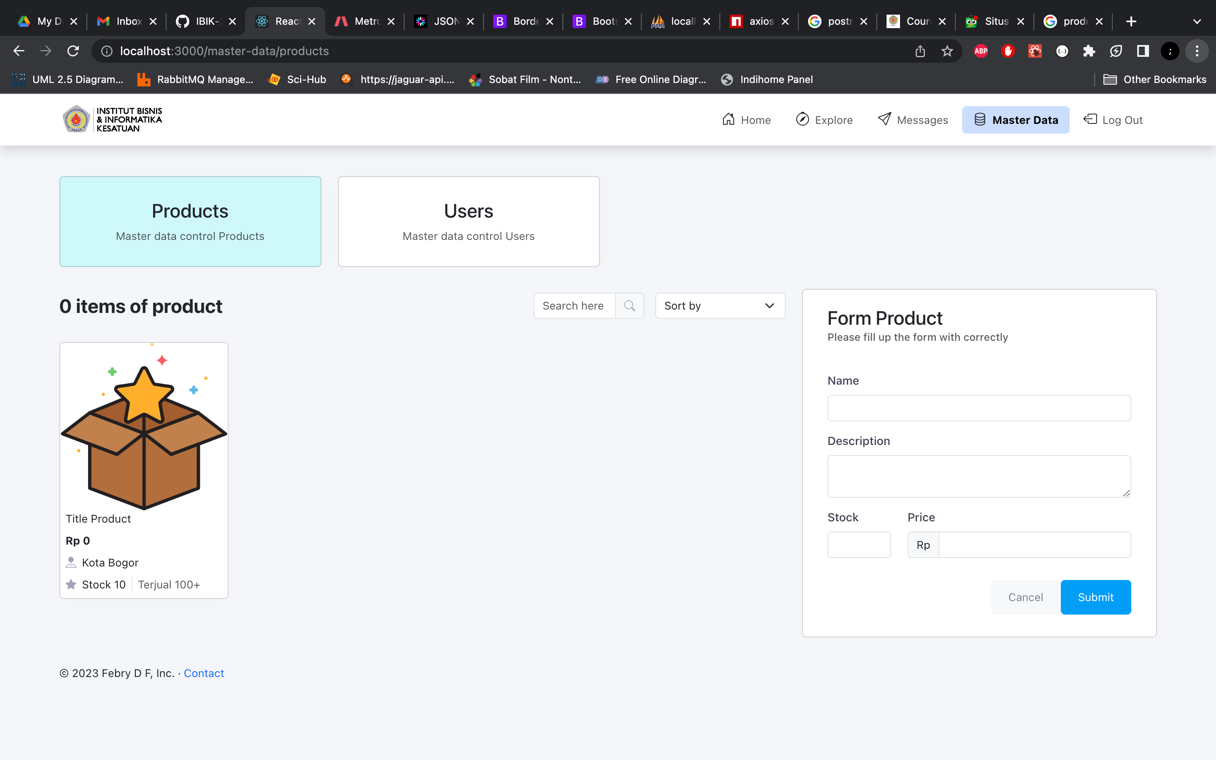
Contoh:



***Gambar 4.2.*** *Microservices testing end-point product*

1. **Cara penggunaan microservices dengan micro-frontend**

Pada tugas minggulalu mengenai layoutin dan route, disini akan membuat sebuah menu atau route baru bernama Master Data. Didalam menu ini memiliki dua buah turunan Outlet yaitu Products dan Users. Pada item pertama yaitu Products, halaman ini diperuntukan untuk mengolah data-data master product seperti melakukan penambahan, penghapusan atau perubahan terhadap seluruh product yang dimiliki, lihat gambar dibawah ini:



***Gambar 4.3.*** *Halaman Micro-Frontend Masrter Data Products*

Berikut ini adalah break down script dari UI gambar 4.3:

**MasterData.js**

|  |
| --- |
| Timeline  Description automatically generated  Diagram  Description automatically generated with low confidence |

Script MasterData.js merupakan bagain dari Class Komponen React dimana file ini diperuntukan untuk membuat titik awal halaman Master Data. Jika dijalankan maka akan memiliki tampilan layout dari komponen MasterData sebagai berikut:

Graphical user interface, application

Description automatically generated

***Gambar 4.3.a.*** *Halaman komponen class MasterData*

Pada master data memiliki dua buah turunan yaitu Products dan Users. Berikut ini adalah bagian turunan Outlet yaitu untuk komponen Products:

**Products.js**

|  |
| --- |
| Timeline  Description automatically generated  Text  Description automatically generated |

**Graphical user interface, text, application, chat or text message

Description automatically generated**

***Gambar 4.3.b.*** *Tampilan komponen function Product*

Script Products.js adalah bagian dari widget yang dimiliki oleh halaman Master Data, oleh karenanya komponen ini dibangun dengan bentuk React Fucntion Components. Pada komponen Products memiliki dua buah function komponen yaitu Form dan TableData.

**TableData.js**

|  |  |
| --- | --- |
| Timeline  Description automatically generated  Text  Description automatically generated |  |

**Form.js**

|  |  |
| --- | --- |
| Timeline  Description automatically generated  Text  Description automatically generated | Graphical user interface, text, application  Description automatically generated |

1. **Menyambungkan Microservices Products kedalam Micro-Frontend Products**

Sebelum menyambungkan micro-frontend dengan Rest API, disini akan membuat sebuah function komponen yang bersifat global, hal ini diperuntukan bagi penggunaan AXIOS itu sendiri. Berikut adalah sebuah function bernama CallAxios.js :

Graphical user interface, text, application, email

Description automatically generated

1. Request GET

Pada function komponen Products tambahkan code dibawah ini:

1. Tambahakan Life Cycle Hook pada function component products

Text

Description automatically generated

**useEffect()** Hook ini aktif secara otomatis saat render pertama dan setiap kali state berubah. Sesuai namanya, ia dipakai untuk melakukan suatu effect setiap kali state berubah. Hook ini cocok dipakai untuk menyiapkan listener, mengambil data dari API serta menghapus listener sebelum component dihapus dari DOM.

**useState()** ada fungsi yang menerima satu parameter, state awal (bisa berupa objek atau data lain yang dipakai suatu component), dan memberikan dua nilai: sebuah state dan fungsi untuk mengubah state tersebut.

1. Mengirimkan variable paramProduct ke JSX TableData sehingga data yang diambil oleh Axios dapat dirender oleh JSX:

Text

Description automatically generated

1. Menuju file komponen TableData tambahkan logical transaction untuk mengolah data yang dikirimkan di property TableData.

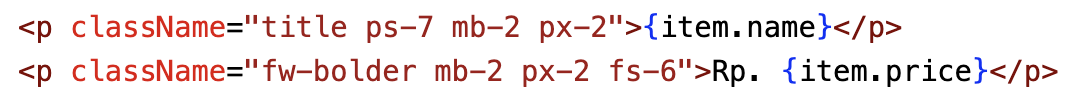
Text

Description automatically generated

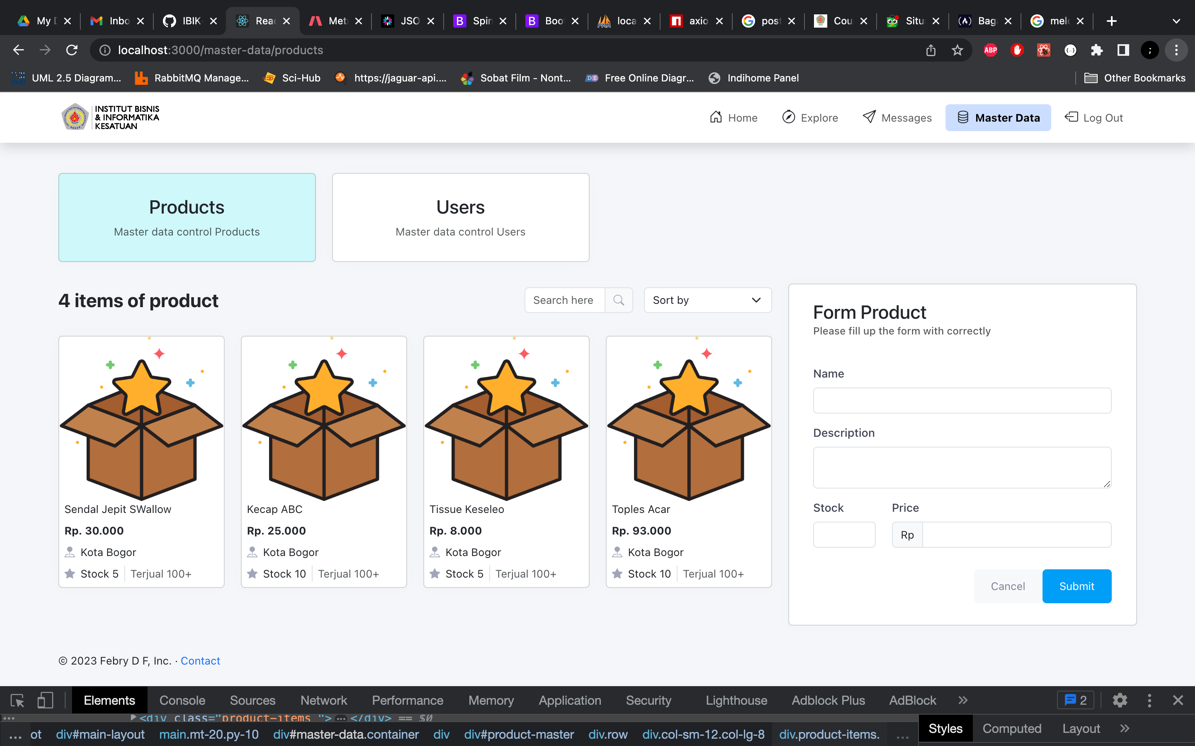
**Menambahkan logical transaction seperti mengecek jika datanya ada maka melooping data dan mengitimkan perObject kedalam <ItemProduct /> agar dapat dirender.**

**Sedangkan tidak ada data maka mencetak No Record Found**

1. Pada function ItemProduct ubahlah value yang bersifat dynamic menjadi static dengan menggantinya seperti berikut:



1. Maka hasil akhir dari tampilan halaman Master Data Product ialah :



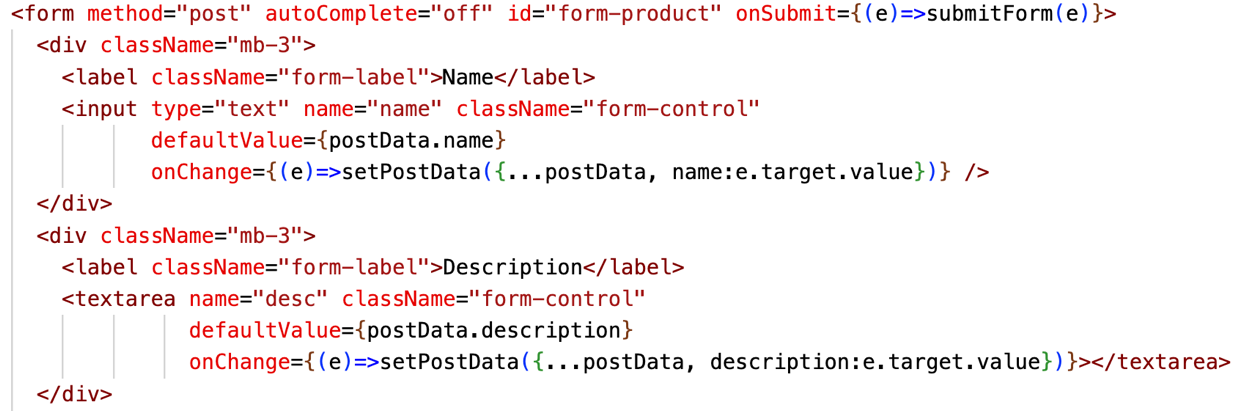
1. Request POST

Contoh request POST kedalam HTTP dengan Rest API ialah pada bagain komponen Form pada Master Data Product.

1. Tambahkanlah Life Cycle Hook kedalam komponen Form untuk mengirimkan nilai data kedalam bentuk Rest API dan akan diterima oleh microservices.



1. Mengubah field pada isian form input dengan HOOK agar datanya dapat berubah secara asynchronous.



Dan seterusnya silakan anda lanjutkan mengupdate isian form dengan HOOK

1. Jika sudah menambahkan semuanya silakan anda coba untuk POST data atau menambahkan data baru melalui layout Form Product.
2. **Latihan Praktikum**
3. Pada project sebelumnya buatlah sebuah package baru didalam folder components dengan nama *latihan-4*.
4. Silakan anda lanjutkan untuk membuat form data master User, dimana end-point Rest Api yang dapat diikuti berdasarkan Table 4.2. Buatlah cara merender seluruh data User, melihat detail dari data user yang dipilih dan dapat menambahkan data User baru.

Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PWL-TI-20-PA-NPM. File source code disimpan sesuai nama project-praktikum dan masukan kedalam repositori tersebut. Buatkanlah file dokumen dalam bentuk file pdf yang berisi Screen Capture dari hasil program yang telah dikerjakan. Simpan dalam file PDF tersebut kedalam project tersebut.

Tambahkan Collaborator management access pada repository anda ke *@FebryFairuz*