BAB 3
PELAKSANAAN KERJA MAGANG

Pada bab ini, akan dijelaskan mengenai pelaksanaan kerja magang yang akan dimulai dari kedudukan dan juga organisasi di kantor, tugas yang dilakukan selama kegiatan magang, serta permasalahan dan juga solusi yang dihadapi selama melakukan kegiatan kerja magang.

3.1 Kedudukan dan Organisasi

Dalam kegiatan magang ini, penulis berkesempatan untuk bergabung sebagai *Fullstack Developer* dengan tim EX *(Employee Experience)*. Tim *Employee Experience* memiliki tugas untuk melakukan pengelolaan terhadap website dan juga aplikasi android yang berhubungan dengan manajemen karyawan yang ada di DEOS Group yang nantinya akan dipakai untuk mengelola data karyawan dan dipakai oleh seluruh anak perusahaan dari DEOS Group seperti AAB (Andalan Anak Bangsa), DAA (Digital Animasi Asia), AMK (Advika Media Kreasi) dan lain sebagainya, artinya cakupan yang aplikasi website maupun mobile yang dibuat oleh tim EX bukan hanya untuk satu PT saja, melainkan membuat aplikasi yang bersifat *generic* yang bisa dipakai oleh berbagai macam perusahaan untuk melakukan manajemen pengelolaan karyawan.

Didalam pelaksanaan kegiatan magang ini, penulis dibimbing oleh rekan-rekan kerja dalam tim EX yang mengerjakan aplikasi untuk manajemen karyawan yaitu *HiTo* yang terdiri dari Karib Chiang selaku *Project Manager*, Guritno Andri Zulverdi selaku *Senior Developer* yang juga merangkap sebagai *Team Leader*, Fa'iq Syahbani Sulisno yang bertugas sebagai *UI/UX Designer*, Anisa Mardhatillah yang bertugas sebagai *Technical Writer* Mohammad Irwansyah Somantri yang bertugas sebagai *Fullstack Mobile Developer*, Indra Aditya, Moch Rian Suhada, Santo Hotmaringantua Lumbantobing, dan Clement Lemuel Setiabudi yang bertugas sebagai *Backend Developer*, Frans Alfiando yang bertugas sebagai *Frontend Web Developer*, dan Atras Shalhan yang bertugas sebagai *Fullstack Web Developer*.

Dalam pengerjaan tugas magang, komunikasi dengan sesama anggota tim di EX HiTo dilakukan secara tatap muka jika bekerja offline, dan juga melalui discord jika pengerjaan dilakukan secara online. Rapat secara hybrid dilakukan setiap harinya melalui aplikasi *Microsoft Teams* yang membahas tentang sprint planning

maupun perkembangan progress aplikasi sekarang. Rentang waktu setiap sprint yaitu sekitar 4-6 hari.

Dalam tahap proses *sprint*, akan ada beberapa *backlog* didalamnya dimana satu *backlog* bisa mempunyai banyak *task*. *Task-task* yang sudah dilakukan *listing* didalam setiap *backlog* nantinya akan dikerjakan pada saat *sprint* sedang berlangsung dan dapat di *assign* orang yang bertugas untuk mengerjakan satuan *task* dalam sprint tersebut serta ada orang yang ditugaskan untuk bertanggung jawab dan mengawasi satu *backlog* dari *sprint* tersebut.

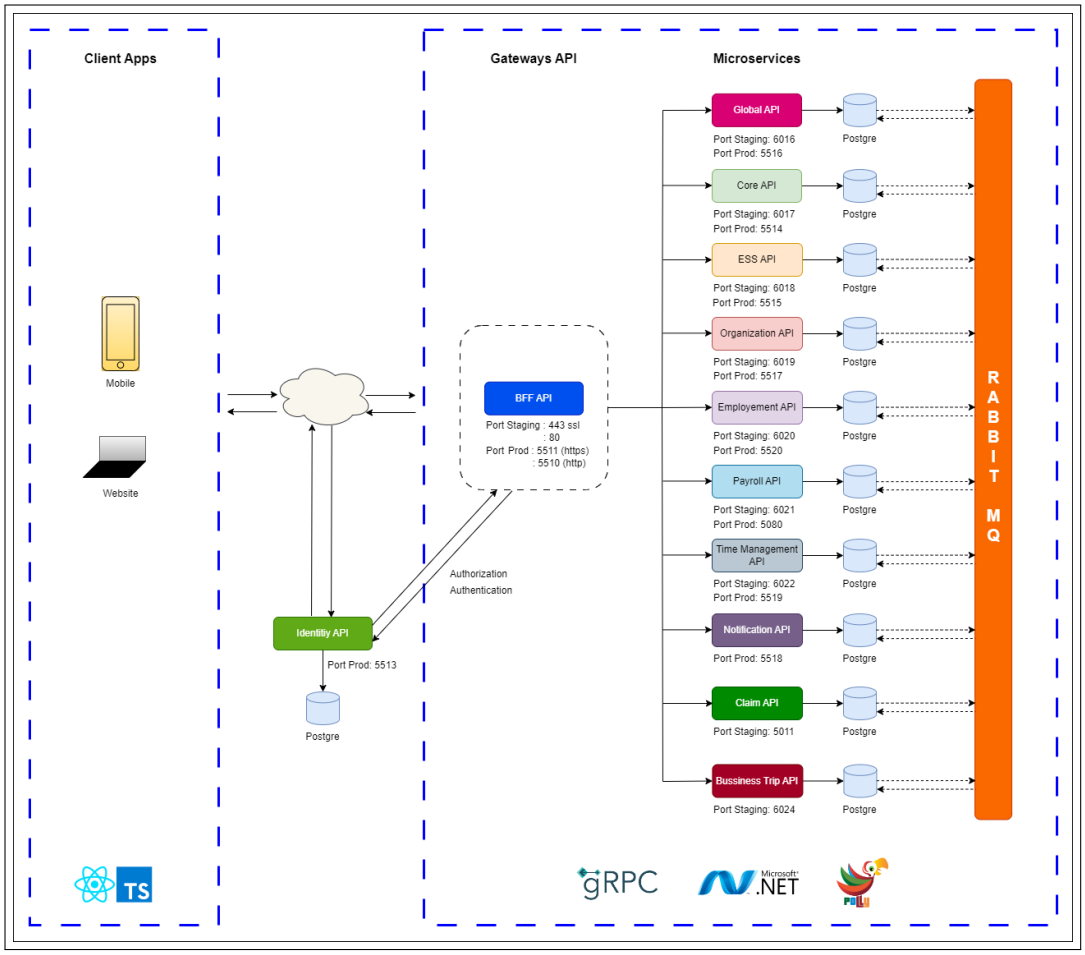
Secara umum, *task* terbagi menjadi dua. Yaitu *task* yang bersifat *bug* yang nantinya *developer* akan mengerjakan *bugfix* untuk *task* tersebut, dan juga task yang bersifat fitur yang dimana nantinya developer akan membuat fitur baru tersebut dalam proses pengerjaannya.

Proses pengerjaan suatu task oleh developer meliputi beberapa langkah yang dimulai dari pembuatan *Branch* baru dalam *Git Repository* yang memiliki format [nama feature/bug]-dev-[nama developer]. Lalu setelah selesai melakukan pengerjaan dalam *branch* tersebut, nantinya developer bisa melakukan *commit* dan *push* terhadap *remote repository* lalu menggabungkan *branch* yang sudah dikerjakan tersebut dengan *branch* utama, yaitu *branch Dev* lalu melakukan *Pull Request* dan setelah di *approve* oleh *approver* maka selanjutnya task tersebut akan berada dalam tahap testing dimana orang *QA (Quality Assurance)* akan menentukan apakah fitur yang sudah dikembangkan tersebut sudah layak untuk dibawa ke website utama *(Production)* atau masih ada yang kurang. Tahap terakhir setelah mendapat approve dari pihak *QA* maka bisa untuk melakukan *deployment* ke environtment *production*.

3.2 Tugas yang Dilakukan

Dalam pelaksanaan magang ini, penulis ditugaskan untuk menjadi Fullstack Developer dalam pengembangan aplikasi website (HiTo) dimana aplikasi tersebut menggunakan arsitektur microservice dari sisi Backend.

Dalam pembuatan aplikasi dengan arsitektur microservice di HiTo, berikut skema yang digunakan.



Gambar 3.1. Skema Aristektur Microservice HiTo

Dalam skema arsitektur microservice hito tersebut, bisa dilihat bahwa sistem yang dibangun untuk membuat Website HiTo terdiri dari dua sisi yang berbeda. Yaitu dari sisi *Client* dan sisi *Server*.

Sisi Client memakai teknologi *React* dengan bahasa *Typescript* untuk websitenya dan *React Native* untuk menunjang kebutuhan aplikasi *Mobile*-nya. *Client-Side* akan melakukan komunikasi dan pertukaran data dengan sisi *Backend* yang dikembangkan secara microservice yang artinya sistem tersebut memiliki berbagai macam service sehingga akhirnya menjadi sebuah aplikasi backend yang utuh dan berjalan dalam protokol Http sehingga Client-Side bisa menggunakan data yang telah disediakan oleh backend dengan melakukan request Http terhadap server HiTo.

Sisi *Server* dari *HiTo* dikembangan dengan beberapa service, meliputi service Global, Core, Identity, ESS, Organization, Employment, Payroll, Time Management, Notification, Claim, dan Business Trip yang dimana semua service

tersebut akan terhubung antara satu sama lain dengan memakai layanan teknologi *Message Broker* yaitu *Rabbit MQ* dan menggunakan *Database PostgreSQL* .

Ketika ada HTTP Request yang datang dari sisi Client ke Server, maka request tersebut akan masuk ke API Gateaway backend terlebih dahulu untuk melakukan otorisasi dan juga autentikasi. Setelah berhasil melakukan otorisasi, maka selanjutnya request akan ditangani oleh Backend For Frontend yang bertugas untuk menentukan service mana yang akan melakukan handling terhadap request tersebut. Setelah itu akan diteruskan ke service yang bersangkutan lalu diproses di service tersebut, hingga hasil akhirnya akan diterima lagi oleh Backend For Frontend untuk dikembalikan lagi ke Client.

Pada kesempatan magang kali ini, penulis berkesempatan untuk mengembangkan optimalisasi di aplikasi website HiTo dengan membuat komponen template pagination pada saat setiap kali melakukan fetching data ke database maupun pada saat melakukan loading asset ke server agar performance yang dihasilkan dalam aplikasi tersebut lebih efisien yang nantinya akan berdampak signifikan pada user experience yang tidak perlu menunggu lama untuk mendapatkan data dari server.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	Memahami Struktur dan alur dari project InBranch
2	Melakukan research terhadap tools yang dipakai dan melakukan implementasi pembuatan Menu Province, dan Branch
3	Melakukan implementasi pengerjaan berbagai fitur yang berhubungan dengan Agent
4	Melakukan implementasi fitur Insentif dan upload data dengan excel ke server
5	Melakukan implementasi fitur report dengan teknologi Telerik Reporting meliputi report Payslip
6	Melakukan implementasi fitur report to Excel meliputi report perpajakan
7	Melakukan implementasi fitur direct Mailing dan scheduled mailing serta masuk ke project HiTo dan menyelesaikan implementasi fitur Grade dan Salary Range
8	Melakukan performance improvement yaitu dengan mengimplementasikan fitur Pagination dari sisi Frontend
9	Melakukan implementasi background service job dengan Hangfire di Inbranch dan implementasi menu Settings Maintenance di HiTo
10	Melakukan performance improvement lanjutan di HiTo dengan melakukan lazy loading terhadap component React TS, menyelesaikan beberapa fitur HiTo dan juga melakukan revisi terhadap UAT Inbranch
10	Melakukan implementasi menu baru yaitu Exceptional Settings Maintenance dan juga penambahan fitur untuk penempatan Agent di Inbranch
11	Melakukan implementasi Excel Data Upload tunjangan mobile di Inbranch
12	Melakukan implementasi pagination untuk seluruh page di HiTo baik dari segi Backend maupun Frontend
12	Melakukan implementasi fitur Excel reporting Feebase Bank Partner untuk Inbranch dan finalisasi Pagination untuk frontend HiTo

3.3 Uraian Pelaksanaan Magang

Pada bagian ini, akan dijelaskan secara teknis bagaimana tugas dan alur program manajemen karyawan HiTo yang dikerjakan oleh penulis baik dari logika proses backend, cara kerja microservice HiTo secara teknis, sampai hasil tampilan *FrontEnd* untuk ditampilkan kepada client.

3.3.1 Proses Pelaksanaan

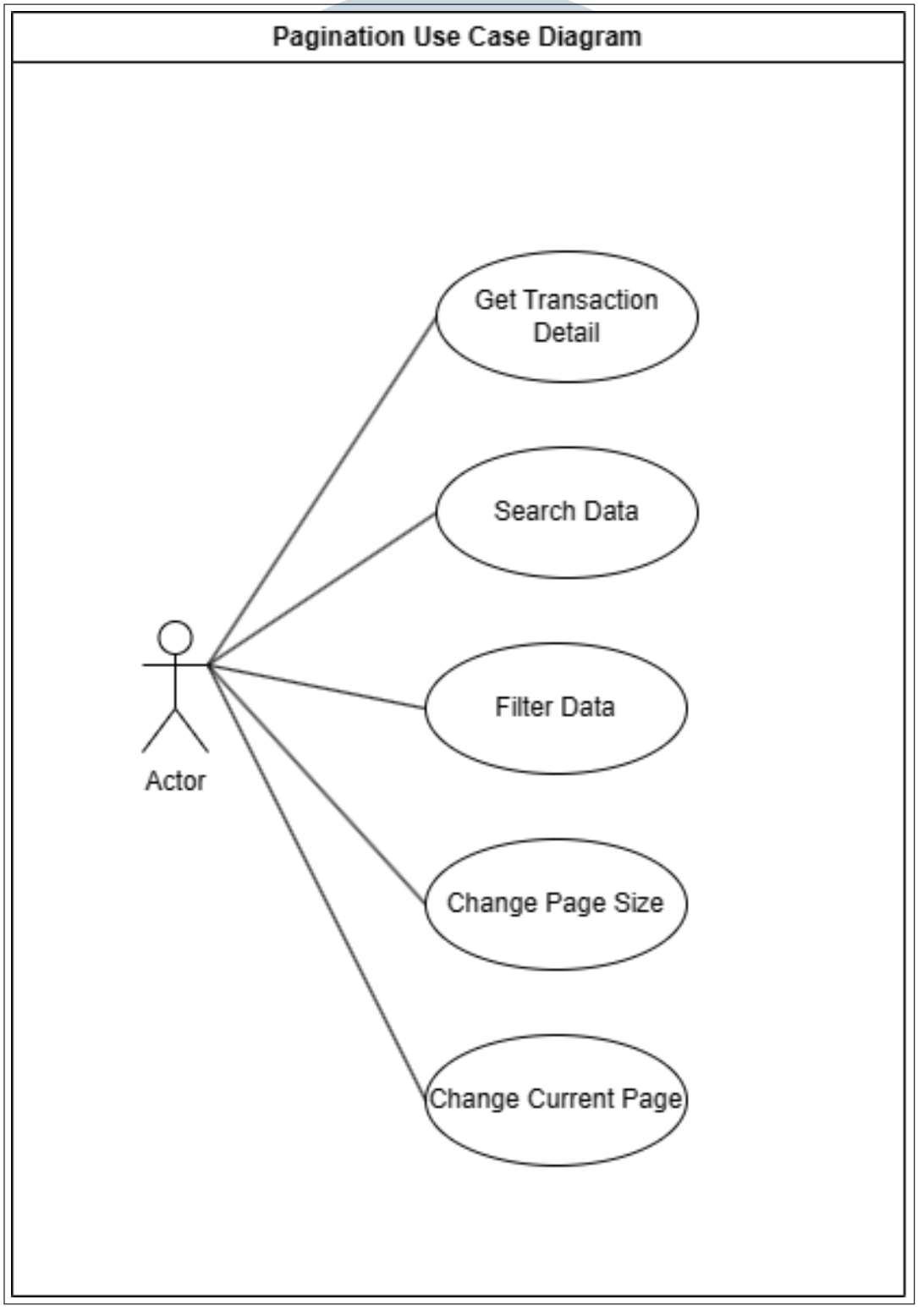
Dalam proses pelaksanaan kerja magang, digunakan perangkat keras dan perangkat lunak untuk menunjang kegiatan kerja magang. Berikut adalah perangkat lunak yang digunakan dalam pengerjaan tugas magang

1. Visual Studio Code (IDE)
2. Typescript (Programming Language)
3. React (Typescript Library/Framework)
4. Ant Design (Styling Component Library)
5. Node Package Manager (Frontend Package Manager)
6. Axios (Frontend HTTP Utility Library)
7. Visual Studio 2022 (IDE)
8. PostgreSQL (Database)
9. Dotnet Framework 6.0 LTS (Software Development Kit)
10. DBeaver (Database Management System)
11. Git (Version Control)
12. Github Desktop (Version Control Support)
13. Windows 11 Home (64 bit) (Operating System)
14. Microsoft Edge (Browser)

Adapun perangkat keras yang digunakan dalam kegiatan magang ini yaitu menggunakan Laptop Lenovo S-340 14IML dengan spesifikasi sebagai berikut :

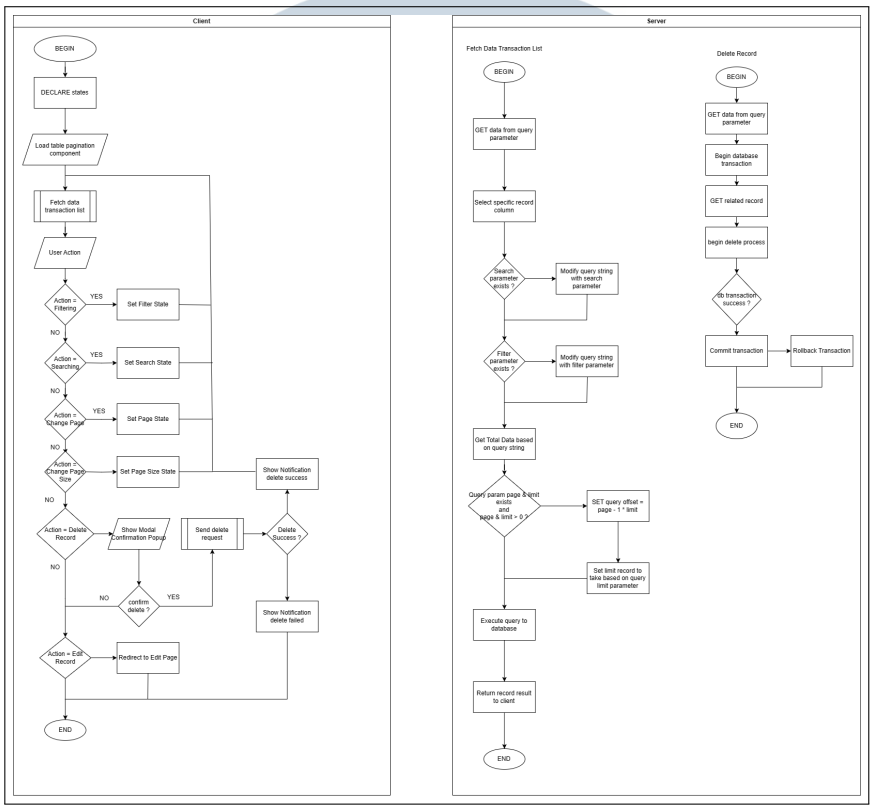
1. I7 Gen 10 (Processor)
2. Nvidia MX230 (Graphic Card)
3. Random Access Memory (RAM) 20 GigaBytes
4. Solid State Drive (SSD) NVME 512 GigaBytes

3.3.2 Use Case Diagram Data Pagination



Gambar 3.2. Use Case Diagram for pagination client and server

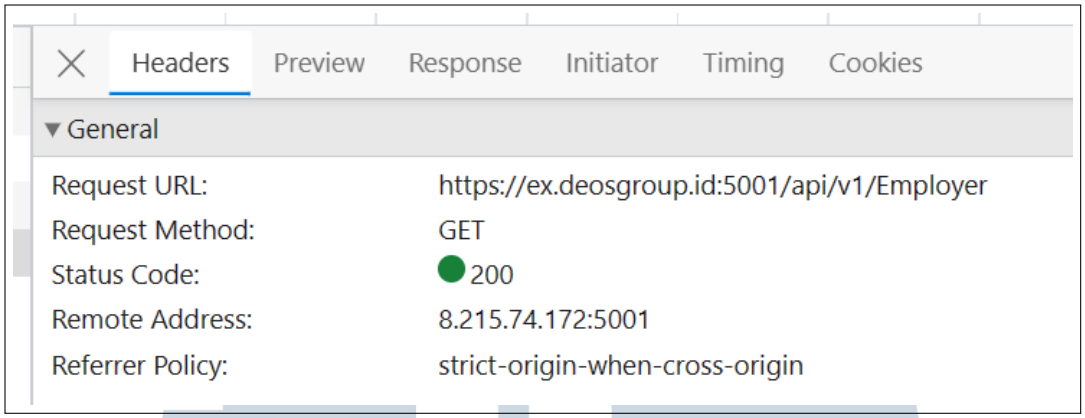
3.3.3 Flowchart Data Pagination



Gambar 3.3. Flowchart for pagination client and server

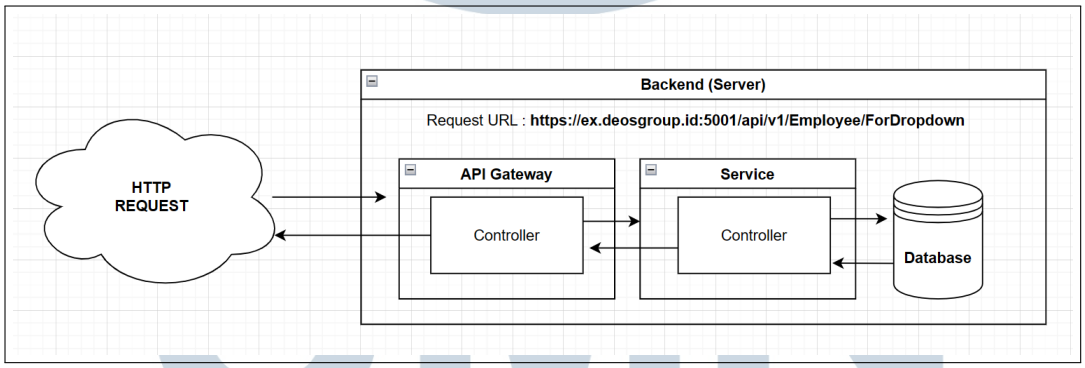
3.3.4 HTTP Request Cycle

Dalam arsitektur miroservice yang dibuat dalam aplikasi HiTo, perputaran data yang dikirim dan diterima oleh website ke server adalah dengan melakukan komunikasi dengan HTTP Request yang akan menghasilkan response dalam bentuk JSON.



Gambar 3.4. Inisiasi HTTP Request Frontend

Cycle dimulai dari frontend yang melakukan panggilan request ke server, kemudian server akan menerima sinyal panggilan request tersebut lalu diterima oleh controller dari API Gateaway. Tugas dari API Gateaway adalah untuk memastikan bahwa request yang masuk adalah request yang sudah terotentikasi oleh server dan juga untuk melakukan *mapping* terhadap routingan url untuk menentukan service mana yang akan menjalankan proses tersebut.



Gambar 3.5. Api Gateaway Controller

Pada request http tersebut dengan url **https://ex.deosgroup.id:5001/api/v1/Employee/ForDropdown**, terlihat bahwa path dari request tersebut adalah **/api/v1/Employee/ForDropdown** dengan ketentuan sebagai berikut :

1. **/api** menandakan bahwa request tersebut ingin melakukan akses terhadap API dari server yang berada pada domain ”ex.deosgroup.id”
2. **/v1** merupakan version API yang ingin dipakai

3. **/Employee** merupakan nama controller yang melakukan *handling* untuk request tersebut di API Gateaway, yaitu **EmployeeController**

4. **/ForDropdown** merupakan nama method yang ada pada controller EmployeeController

Setelah controller menerima informasi request yang masuk tersebut berdasarkan URL yang diterima, selanjutnya API Gateaway akan memanggil *service* yang telah dilakukan *dependencies injection* untuk melakukan mapping routing ke service yang bertanggung jawab yang dapat ditentukan melalui variable **uri** dan juga **httpclient**. P ada kasus di gambar 3.4 tersebut, uri akan diisi **api/v1/Employee/ForDropdown** dengan seluruh *query parameter* yang dikirimkan oleh client, dan **httpclient** yang dibuat dengan nama servicenya adalah **Employment** sehingga didapatkan service yang bertanggung jawab untuk menangani request tersebut yaitu service **Employment** lalu akan ditangkap oleh controller **EmployeeController** fungsi **ForDropdown** di service tersebut.

Selanjutnya pada controller EmployeeController di service **Employment** fungsi **ForDropdown** akan memanggil GetEmployeeDropdownQuery yang akan melakukan handling logic untuk request tersebut lalu mengembalikan response yang merupakan sebuah object ke service

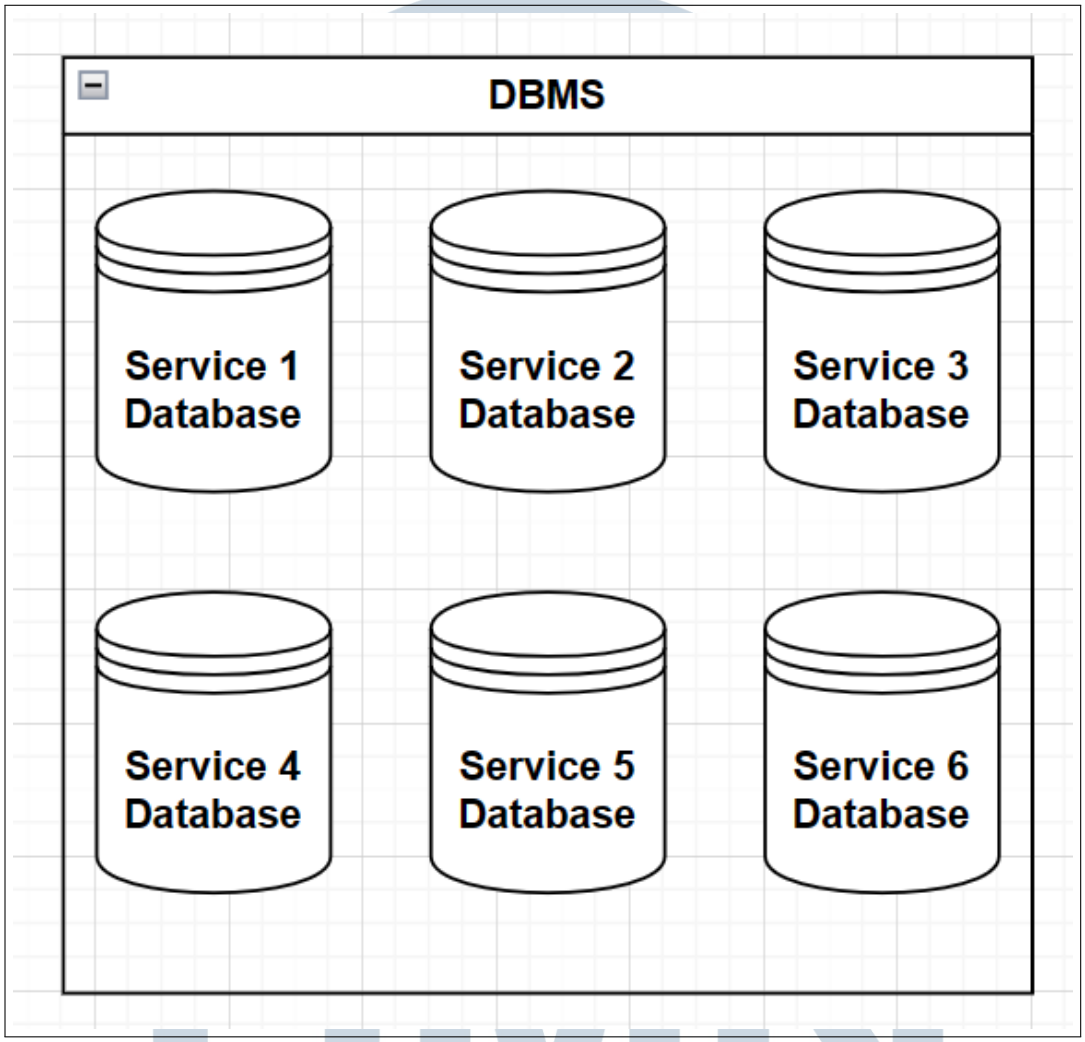
Logic daripada HTTP request cycle tersebut ada didalam class *GetEmployeeDropdownQuery* dimana class tersebut melakukan inheritance terhadap class IRequest dari design pattern mediator yang dikembangkan melalui library open source MediatR. Design pattern mediator tersebut digunakan untuk melakukan pemisahan terhadap dependency antar modul sehingga modul satu dengan modul lainnya tidak memiliki banyak dependency antara satu sama lain karena modul yang baik itu adalah modul yang bisa mengusung konsep *Low Coupling dan High Cohesion* dimana artinya modul yang bagus adalah modul yang tidak banyak terikat dengan modul lainnya karena ketika suatu modul sumber mempunyai *dependency* ke modul lainnya, akan sangat sulit untuk melakukan maintenance kedepannya. Hal tersebut dikarenakan jika ada perubahan di suatu modul, maka hal tersebut juga akan mempengaruhi cara kerja dari modul lain sehingga modul yang memiliki dependency terhadap modul sumber perlu dilakukan adjustment juga.

Class yang melakukan inheritance terhadap design pattern MediatR harus memiliki property berupa query parameter yang akan dipassing dari dlient melalui HTTP Request query params. Selain itu class yang melakukan implementasi

terhadap library MediatR harus memiliki implementasi terhadap handler dari class IRequestHandler dengan nama method Handle. Didalam method handler akan dilakukan fetching dan pengolahan data melalui database PostgreSQL. Setiap kali sistem melakukan pertukaran data ke database, HiTo memiliki standarisasi untuk melakukan implementasi database transaction seperti begin transaction, rollback transaction, commit transaction, serta save transaction changes.

Setelah logic diolah di service, kemudian service akan mengembalikannya lagi ke API Gateaway kemudian controller di API Gateaway akan melakukan pengecekan status response yang akan diterima oleh client dan memiliki kemungkinan return responsenya adalah *OK*, *Client Error*, maupun *Server Error*. Jika response yang dikembalikan berupa OK dengan status code 200, maka data akan dilakukan converting dari object menjadi notasi JSON hingga akhirnya request tersebut dikembalikan ke client.

3.3.5 Penyimpanan Data



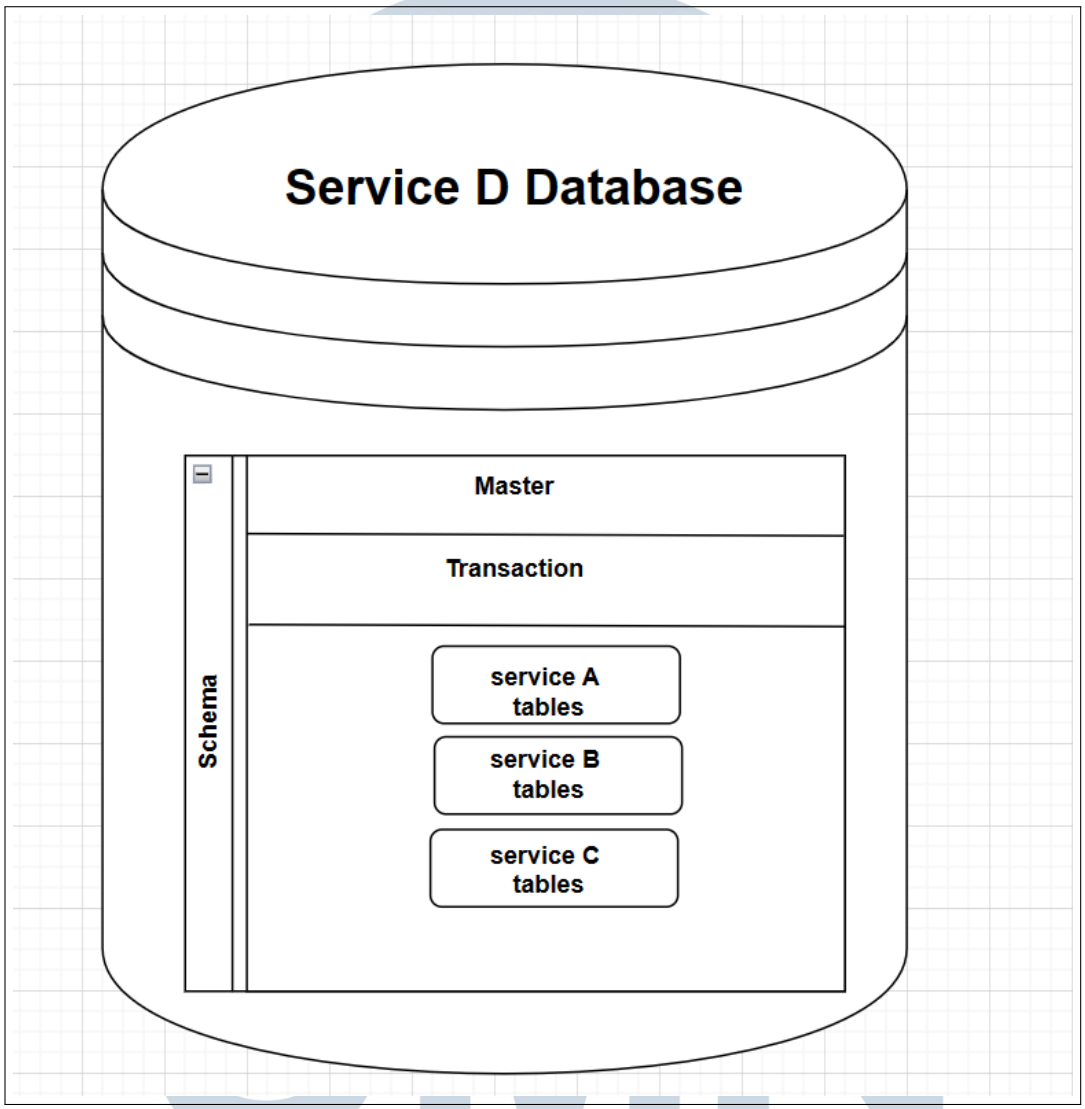
Gambar 3.6. UnitOfWork Interface

Desain arsitektur HiTo mengusung konsep arsitektur Microservice yang artinya aplikasi ini tidak berdiri secara satu kesatuan melainkan berdiri secara terpisah yang dipisahkan berdasarkan modul atau service yang memiliki tanggung jawab untuk pengelolaan data tergantung dari konteks service tersebut. Dalam konsep arsitektur Microservice, setiap service yang ada dalam aplikasi ini, memiliki databasenya terpisah secara sendiri-sendiri. Sebagai contoh service Employment memiliki database dengan nama DEOS_EmploymentDB yang akan menyimpan data untuk melakukan pengelolaan data Employee, begitu juga dengan service lainnya.

Namun pada saat implementasinya, antara serive satu dengan service

lainnya, kadang kala memiliki kemungkinan untuk mengambil data dari kedua service tersebut. Sebagai contoh, ketika service ESS memerlukan data employee, namun data employee hanya tersedia di database Employment, maka service ESS tidak bisa mengambil data dari database Employment. Cara untuk mengatasi masalah tersebut adalah dengan menggunakan message broker. Message broker adalah sebuah layanan yang banyak digunakan dalam arsitektur microservice yang berjalan berdasarkan event yang ada dan memiliki berbagai macam fungsi salah satunya adalah untuk melakukan replikasi data pada satu database dengan database lainnya sehingga ketika service ESS membutuhkan data yang data tersebut hanya tersedia di service Employment, hal tersebut sekarang memungkinkan utuk dilakukan.

A. Database Per Service



Gambar 3.7. UnitOfWork Interface

Setiap service yang memiliki databasenya sendiri dapat melakukan pengambilan data terhadap database di service lain memanfaatkan message broker yang pemisahan datanya dipisahkan berdasarkan schema dari database tersebut. Setiap database akan memiliki 3 pembagian schema utama yang membentuk database tersebut yaitu: schema master yang berfungsi sebagai tempat untuk menyimpan table-tabel inti dari service tersebut; schema transaction yang menyimpan seluruh transaksi perubahan dari service tersebut; dan juga schema dengan nama service lain yang isinya adalah replika data dari service lain yang

dibutuhkan dalam database tersebut sehingga memungkinkan suatu database untuk mengakses data dari service lain.

3.3.6 Performance Optimization

Aplikasi HiTo terdapat beberapa hal yang berpotensi terjadi masalah yang major dalam jangka panjang. Yaitu performance dari sisi loading data ke table, maupun performance untuk melakukan downloading asset ke server. Namun kedua hal tersebut tentu bisa ditanggulangi dengan menambahkan salah satu teknik untuk melakukan fetching data ke server dengan cara memecah data tersebut menjadi beberapa bagian yang kecil sehingga beban kerja server tidak terlalu berat untuk melakukan pengambilan data dari database. Teknik pagination bisa diterapkan dalam hal yang berpotensi menjadi masalah di HiTo tersebut, yaitu menerapkan pagination ketika ingin melakukan loading data dari server, maupun pagination untuk melakukan downloading asset secara sedikit-sedikit.

Pagination adalah teknik yang digunakan untuk menampilkan data besar dalam potongan-potongan kecil berdasarkan interaksi pengguna dengan beberapa parameter yang biasanya terdiri dari nomor halaman yang ingin diambil, jumlah data yang ingin diambil dalam sekali pengambilan data, nilai searching dan juga nilai filtering. Pagination dapat meningkatkan kinerja aplikasi web dengan mengurangi waktu pemuatan dan penggunaan memori secara signifikan. Pada kasus ini, penulis menerapkan pagination dengan react js sebagai frontend dan c# sebagai backend dengan bantuan beberapa library seperti Axios, Ant Design, Entity Framework, dan juga linQ. Axios adalah klien HTTP yang memungkinkan kita untuk mengirim permintaan ke web API dan menerima data dari server. Web API adalah layanan web yang menggunakan protokol HTTP untuk berkomunikasi dengan aplikasi klien dan menyediakan data dalam format JSON atau XML. Dengan menggunakan kombinasi dari beberapa library yang disebutkan tadi, kita dapat membuat aplikasi web yang lebih efisien dengan pagination.

A. Backend Performance Optimization

Dalam hal melakukan performance optimization dari sisi server yang dimana nantinya server akan melakukan pengambilan data ke database. Ada beberapa hal yang dapat membuat performa pengambilan data tersebut bisa menjadi lebih cepat dengan teknik, teknik tertentu salah satunya adalah Pagination. Teknik

pagination dapat di implementasikan dari sisi server ketika pengambilan data ke database dapat didapatkan dengan melakukan optimalisasi query dengan bantuan Entity Framework serta teknik pemotongan pengambilan data secara sedikit-demi sedikit.

A.1 Query Optimization

```
0 references
public async Task<BasePagingWithDataResponse> Handle(GetAllPositionQuery request, CancellationToken cancellationToken)
{
    var page = request.CurrentPage < 0 ? 0 : request.CurrentPage;
    var limit = request.Limit;
    var search = request.Search;

    BasePagingWithDataResponse response = new BasePagingWithDataResponse()
    {
        Count = 0,
        CurrentPage = 0,
        IsSuccess = false,
        Limit = 0,
        Message = string.Empty,
        Data = null
    };
```

Gambar 3.8. Query Parameter and async function

Dalam implementasinya, ada beberapa cara untuk melakukan optimisasi dari segi query serta sifat function untuk meningkatkan performa pengambilan data ke database. Pada saat request masuk ke server untuk mengambil data pagination tertentu, function yang digunakan untuk melakukan handling logic pagination harus bersifat asynchronous dimana nantinya request tersebut akan berjalan secara paralel di belakang layar supaya tidak menghambat dan mengganggu proses lainnya.

Selain itu, dalam logic implementasi pagination ini juga perlu untuk memberikan return data yang konsisten antar requestnya. Beberapa reutrn data yang diperlukan dalam teknik pagination ini antara lain adalah : jumlah data, halaman data sekarang, limit ambil data berapa banyak, serta data yang telah dipotong-potong.

```
positions = _unitOfWork.BaseRepositoryNonIdAsync<Organization.Domain.Entities.Position.Position, long>().Entities.AsNoTracking()
.Where(w => w.ExpiredDate == null || w.ExpiredDate > DateTime.Now)
.Select(c => new GetAllPositionResponse
{
    Id = c.Id,
    ErCode = c.EmployerCode,
    EmployerName = employerRepo.Entities.AsNoTracking().FirstOrDefault(m => m.ErCode == c.EmployerCode)!.ErName,
    PosCode = c.PositionCode,
```

Gambar 3.9. Query Parameter and async function

Selain itu, karena penulis menggunakan Entity Framework untuk ORM (Object Relational Mapping) nya, maka ada lagi hal lainnya yang bisa dilakukan

improve terhadap kecepatan query untuk melakukan get request ke database yaitu dengan method yang sudah disediakan dalam ORM Entity Framework yang bernama asNoTracking(). Method asNoTracking adalah method bawaan entity framework yang dapat digunakan untuk disable tracking ketika melakukan query ke database. Tracking dilakukan ketika ingin melakukan manipulasi record di database seperti aksi untuk menambah, mengubah, ataupun menghapus record kedalam database, maka entity framework perlu mengubah status trackingnya sesuai dengan aksi yang dilakukan oleh program. Namun dalam hal pagination, kita tidak ingin mengubah data apapun di database, melainkan kita hanya ingin mengambil potongan kecil data di database. Maka dari itu dalam kasus pagination, kita bisa memanfaatkan method asNoTracking() milik Entity Framework untuk meningkatkan response time ketika sedang melakukan query kedalam database.

A.2 Pagination Query Formula

Logika untuk melakukan implementasi pagination, dimulai dari menentukan query string yang dibutuhkan, pada kasus ini penulis menggunakan entity framwork untuk membuat query stringnya, juga menambahkan kondisi sesuai dengan parameter yang dikirimkan didalam request yaitu bisa berupa potongan kata dari suatu kolom, ataupun melaukan filtering dengan kondisi tertentu.

```
response.TotalData = positions.Count();
if (page > 0 && limit > 0)
    positions = positions.Skip((page - 1) * limit).Take(limit);

response.IsSuccess = true;
response.Message = "Success Get Data";
response.Limit = limit;
response.CurrentPage = page;
response.Count = positions.Count();
response.Data = await positions.ToListAsync(cancellationToken: cancellationToken);

return response;
```

Gambar 3.10. Query Parameter and async function

Setelah mendapatkan query stringnya, query string tersebut belum di eksekusi kedalam database, melainkan akan ditambahkan limit dan juga offset dari query string tersebut. Setelah itu query string akan di eksekusi ke database dan diambil jumlah record yang valid berdasarkan kondisi yang dikirimkan oleh http request melalui query parameter. Juga disarankan untuk melakukan eksekusi query kedalam databasenya secara asynchronous juga untuk lebih mempercepat query.

B. Frontend Performance Optimization

Untuk melakukan optimalisasi performa tidak cukup jika hanya dilakukan di backend saja, bagian frontend juga perlu melakukan penyesuaian terhadap penarikan data yang diambil dari backend sesuai dengan data yang diminta dan dikirimkan dari backend.

Dalam hal ini, penulis memanfaatkan *library axios* untuk melakukan *fetching* data ke *server* yang akan dimanipulasi penggunaannya dengan bantuan *state* dari *react JS*.

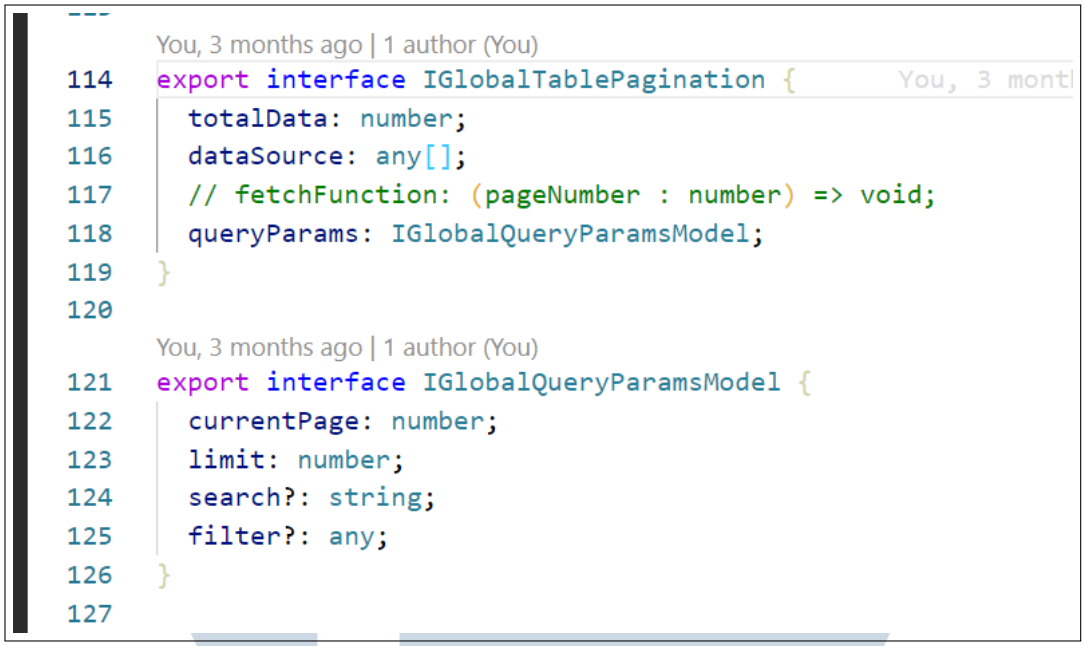
B.1 Lazy Loading Table Template Component

Aplikasi HiTo merupakan aplikasi yang besar dan memiliki banyak halaman yang hampir dari semua halamannya menampilkan data dalam bentuk tabel. Melihat kebutuhan dari pagination tabel ini dibutuhkan di banyak halaman, maka penulis membuat suatu komponen khusus supaya dapat menerapkan salah satu teknik *clean code* yaitu *Code Reusability* pada setiap page nya sehingga penulis hanya perlu membuat satu template yang nantinya dapat diimplementasikan di berbagai halaman HiTo yang membutuhkan penampilan data dalam bentuk tabel dengan mudah.

Manajemen state merupakan hal yang penting dalam melakukan tracking terhadap data yang berputar dalam aplikasi React JS dalam mengelola data Pagination dalam tabel. Pada pembuatan template component pagination ini ada beberapa state yang berperan besar untuk template pagination ini dapat berjalan, diantaranya adalah :

```
const [tablePagination, setTablePagination] = useState<IGlobalTablePagination>(INIT_TABLE_PAGINATION);
const [openFilter, setOpenFilter] = useState<boolean>(false);
const [openAddDialog, setOpenAddDialog] = useState<boolean>(false);
const [selectedCheckbox, setSelectedCheckbox] = useState<{ keys: any[]; records: any[] }>();
const [openDeleteModal, setOpenDeleteModal] = useState(false);
const [deleteRecord, setDeleteRecord] = useState<any>();
const itemsOption = [10, 20, 50, 100];
```

Gambar 3.11. Table Pagination State



Gambar 3.12. Table Pagination Model

State utama yang membangun template pagination ini adalah state tablePagination yang memiliki tipe data IGlobalTablePagination yang memiliki beberapa property yaitu totalData untuk memberi info total data dalam suatu tabel, dataSource yang menyimpan data dalam tabel, serta queryParams dimana property tersebut akan menyimpan meta-data yang nantinya akan mengatur perputaran data jika ada meta-data yang berubah yang di trigger oleh pengguna website melalui beberapa *event*.

B.2 Fetching Data

system akan melakukan rendering awal pada saat pertama kali webpage dibuka dengan menjalankan salah satu *react hooks* yaitu *useEffect* dengan *dependency* yaitu *array* kosong yang berfungsi untuk melakukan *rendering* awal yang akan mengambil data *pagination* pertama yang memiliki beberapa step.

B.2.1 Define axios service

```
29  export const getEmployerListPagination = async (qp: IGlobalQueryParamsModel) => {
30    return await axios
31      .get(`${END_POINT}?${QueryString.stringify(qp)}`)
32      .then((response) => {
33        return { error: false, data: response.data, metaData: response, message: "success" };
34      })
35      .catch((error) => {
36        return { error: true, data: null, metaData: null, message: error.message };
37      });
38  };
```

Gambar 3.13. Axios service

Service layer adalah sebuah layer *asyncronous* yang dibuat untuk memisahkan antara layer kode untuk logic program inti dengan kode yang berfokus untuk melakukan fetching data.

Layer ini bertujuan untuk melakukan pemanggilan data ke server melalui library Axios berdasarkan dengan url tertentu untuk melakukan request HTTP dengan jenis GET, POST, ataupun PUT dimana GET dilakukan untuk mengambil data, POST dilakukan untuk membuat data baru, dan put dilakukan untuk melakukan edit data sert juga ada parameter tambahan jika dibutuhkan yang memiliki tipe data *IGlobalQueryParamsModel* yang nilai nya adalah *meta data* tentang data *pagination* yang ingin diambil.

Axios melakukan pemanggilan data ke server dengan function *axios.get(queryParams: IGlobalQUeryParamsModel)* yang berarti client akan melakukan HTTP Request dengan method *GET* dan akan mengirimkan ke *endpoint* tertentu dengan *meta-data* sesuai dengan apapun yang dipassing kedalam service tersebut oleh logic program di layer sebelumnya.

Jika http request tersebut berhasil, maka backend akan mengembalikan response dengan status 200 (OK) namun jika request HTTP tersebut gagal, server akan mengembalikan response dengan status 400-500 tergantung dari jenis errornya. Service layer kemudian akan mengembalikan request tersebut berupa response JSON yang memiliki property berupa error, data, metaData, dan juga message. MetaData diperlukan supaya nantinya bisa dikelola oleh component table lazy loading yang mengharuskan mengisi beberapa metaData ke dalam component tersebut.

B.2.2 Create Fetch Data Logic

```
async function fetchData(queryParams : IGlobalQueryParamsModel) {
  setLoading(true);
  const data = await getEmployeeList(queryParams)
    .then(({ error, data, metaData, message }) => {
      if (error && message) {
        Notification.error(message);
        return;
      }
      const temp = [...data].map((res, idx) => {
        return { key: idx, id: res.employeeId, ...res };
      });

      temp.sort((a, b) => b.employeeId - a.employeeId)

      return {dataSource: temp, metaData: metaData};
    })
    .catch((error) => {
      console.log("Error fetching data");
    })
    .finally(() => {
      setLoading(false);
    });

    return data;
}
```

Gambar 3.14. Fetch data frontend logic

FetchData merupakan sebuah *function* yang akan mendefinisikan *logic* yang akan dijalankan setiap kali ada *request HTTP GET pagination* ke *server*.

di function ini, kita akan melakukan pemanggilan function dari service layer yang bersifat *asyncronous* untuk melakukan HTTP Request dengan meta data tertentu sesuai dengan nilai yang ada di parameter queryParams. karena function ini merupakan function yang asinkron, yang berarti function tersebut akan berjalan secara paralel di thread yang berbeda di belakang layar, maka system harus menunggu sampai proses di service tersebut selesai dengan keyword 'await' supaya mendapatkan hasil yang sesuai.

Setelah function di service layer tersebut berhasil dijalankan maka selanjutnya akan dilakukan pengecekan terhadap error jika ada dan juga melakukan

mapping data untuk menambahkan key di setiap recordnya lalu di sorting berdasarkan kolom tertentu. Setelah itu function tersebut harus mengembalikan sebuah object dengan 2 property yaitu dataSource dan juga metaData untuk kemudian dikelola oleh *component template pagination*

B.2.3 Template Pagination Component

```
useEffect(() => {
    getData();
    // eslint-disable-next-line react-hooks/exhaustive-deps
}, [tablePagination.queryParams, deleteSignal, fetchSignal]);
```

Gambar 3.15. UseEffect hooks in template pagination

```
async function getData() {
    // System ngirimin filterString tapi belum ada isinya
    if (typeof filterString !== "undefined" && !filterString) return;

    if (setLoading !== undefined) setLoading(true);

    const { dataSource, metaData } = await fetchData(tablePagination.queryParams);

    setTablePagination((prevTablePagination) => {
        return {
            ...prevTablePagination,
            dataSource: dataSource,
            totalData: metaData.totalData,
        };
    });

    if (setLoading !== undefined) setLoading(false);
}
```

Gambar 3.16. Get Data Definition in pagination template component

Pada saat pertama kali webpage melakukan rendering, maka state tablePagination akan terbuat lalu menjalankan function di useEffect dengan dependency tablePagination.queryParams yang menjalankan function getData.

Function getData akan melakukan pemanggilan request HTTP yang sudah didefinisikan functionnya pada gambar 3.9 lalu program akan melakukan inisialisasi data kepada state tablePagination dengan nilai apapun yang dikembalikan oleh

server untuk kemudian data tersebut ditampilkan ke webpage melalui component table Ant Design.

B.3 Filtering Data

Untuk melakukan filtering data pada *component template pagination table* dibutuhkan 2 (Dua) hal yang harus disiapkan, Yaitu adalah konten *User Interface (UI)* daripada *filtering data* itu sendiri, dan juga sebuah *state* yang akan berubah nilai nya setiap kali *user* melakukan perubahan pada komponen *UI*.

```
const [filterLeaveType, setFilterLeaveType] = useState<any>([]);
const [filterStatus, setFilterStatus] = useState<string>("Active");
const [filterString, setBaseFilterString] = useState<string>("");
```

Gambar 3.17. Filter States

```
179     function _renderFilterList() {
180         // Logic
181
182         return (
183             <Fragment>
184                 <div className="leave-filter-container">
185                     {/* UI Implementation */}
186                 </div>
187                 <div className="leave-filter-button-container">
188                     {/* UI Implementation */}
189                 </div>
190             </Fragment>
191         );
192     }
```

Gambar 3.18. Render Filter List Component Definition

```
159     function applyFilter() {
160         const filterObject: ISettingsMaintenanceFilter = {
161             leaveType: filterLeaveType.length > 0 ? filterLeaveType : getAllLeaveTypeId(leaveTypeList),
162             status: filterStatus,
163         };
164
165         setFilterString(filterObject);
166     }
167
168     function resetFilter() {
169         const filterObject: ISettingsMaintenanceFilter = {
170             leaveType: [...leaveTypeList].map((o) => o.leaveTypeId),
171             status: "Active",
172         };
173
174         setFilterStatus("Active");
175         setFilterLeaveType([]);
176         setFilterString(filterObject);
177     }
```

Gambar 3.19. Filter Apply and Reset Buttons

B.3.1 Rendering Filter UI Component

UI Component akan disiapkan didalam sebuah function yang bisa dilihat pada contoh gambar 3.18 dan akan mengembalikan sebuah *JSX Element* yang terbagi menjadi 2 (Dua) komponen utama didalamnya, yaitu Container dari semua list pilihan filter yang ada, lalu container untuk menampung button untuk melakukan perubahan pada state filterstring yang nantinya ketika state filterString mengalami perubahan, maka komponen table pagination akan otomatis melakukan fetching data berdasarkan nilai dari state filterString yang berisikan sebuah string dalam format JSON yang nantinya akan memberitahu server untuk menambahkan fitur filtering ketika sedang melakukan query data.

B.3.2 Filter String Trigger

```
72    useEffect(() => {
73        setTablePagination((prevTablePagination) => {
74            return {
75                ...prevTablePagination,
76                queryParams: {
77                    ...prevTablePagination.queryParams,
78                    filter: filterString ?? "",
79                },
80            };
81        });
82    }, [filterString]);
```

Gambar 3.20. Filter String UseEffect Dependency

Function tersebut akan melakukan checking secara terus menerus memperhatikan apakah state filterString mengalami perubahan atau tidak.

Ketika salah satu button dari filter ditekan oleh user, maka hal tersebut akan melakukan perubahan pada nilai dari state filterString yang akan secara otomatis mengirimkan sinyal ke template pagination yang sudah dibuat untuk melakukan http request ulang dengan konfigurasi yang ada.

B.4 Deleting Data

```
13    const [deleteSignal, setDeleteSignal] = useState<boolean>(true);
```

Gambar 3.21. Delete Signal State

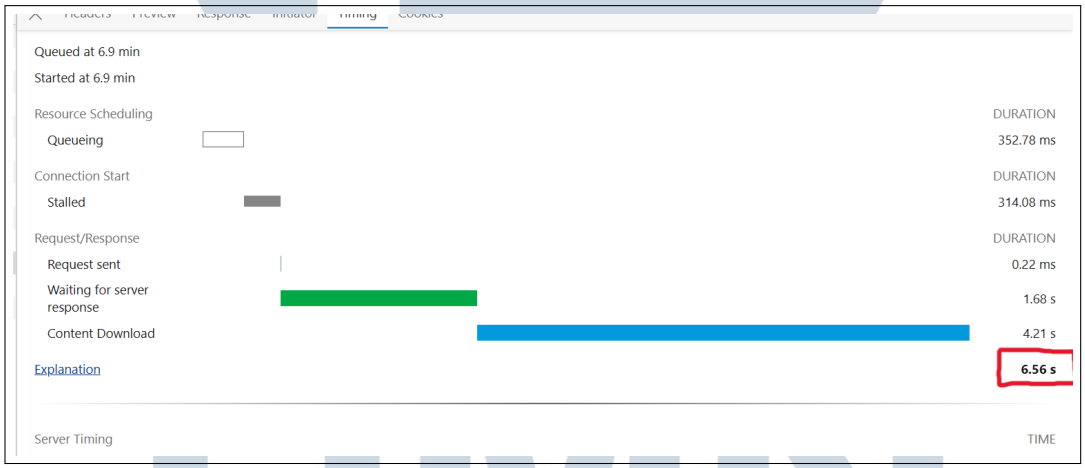
```
41  const onDeleteFormulaVocab = (record: IFormulaVocabularyListItem) => {
42    deleteFormulaVocab(record.vocabId).then(({ error, message }) => {
43      if (error && message) return Notification.error(message);
44
45      Notification.success("Formula Vocab has been removed successfully.")
46      setDeleteSignal(prevState => !prevState);
47      // _fetchFormulaVocabList(INIT_QUERY_PARAMS);
48      // getData();
49    })
50  }
```

Gambar 3.22. Delete Signal Function

Ketika suatu record dalam table mengalami *action delete*, maka sistem harus memberitahu kepada component table pagination bahwa ada suatu record yang di delete, hal tersebut dilakukan supaya component table pagination melakukan http request ulang guna selalu mendapatkan list record yang paling baru setelah melakukan penghapusan data di server.

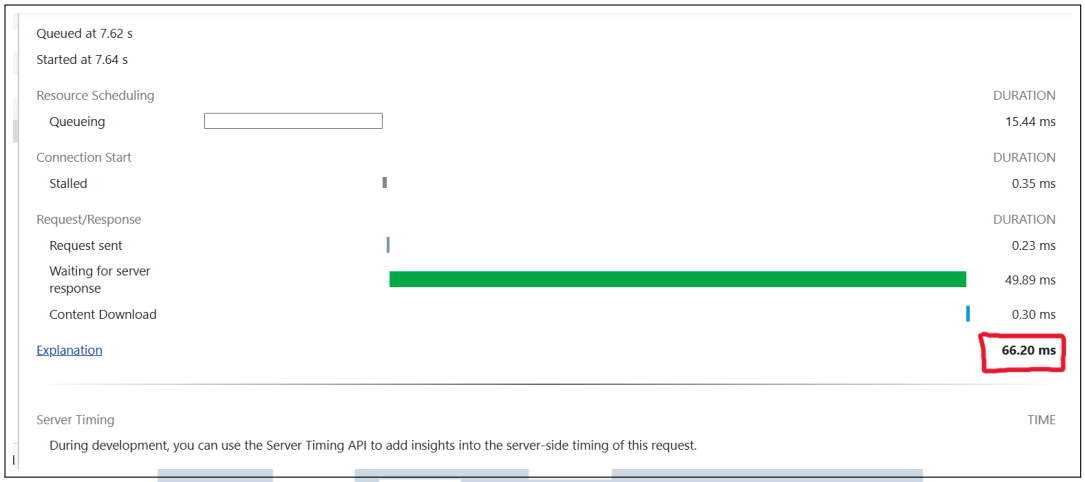
sistem dapat meberitahu component table pagination bahwa ada record yang terhapus melalui sebuah state yang bernama deleteSignal yang akan dipassing kedalam component table pagination dimana ketika state tersebut berubah, maka secara otomatis component table pagination akan melakukan http request ulang sehingga record yang ada pada list table pagination akan selalu *ter-update*.

3.4 Hasil Implementasi Teknik Pagination



Gambar 3.23. Filter String UseEffect Dependency

Pada sebelum dilakukan implementasi teknik pagination dalam hito, response time yang dilakukan untuk 1 (Satu) cycle http request membutuhkan waktu 6.56 detik, Waktu tersebut merupakan waktu yang cukup lama bagi user untuk menunggu hasil response ketika ingin mengambil data dan cukup tidak bagus jika dilihat dari sisi user experience nya.



Gambar 3.24. Filter String UseEffect Dependency

Setelah dilakukan implementasi teknik pagination pada server hito, ketika dilakukan hit http request, response time nya hanya perlu membutuhkan waktu total 68 Milisekon. Tentu hal ini merupakan *improvement* yang sangat bagus dari sebelum dilakukan teknik pagination ini untuk pengambilan data dari sisi server maupun client.

3.5 Kendala dan Solusi yang Ditemukan

Pada Subbab ini, akan dijelaskan tentang kendala dan juga solusi yang ditemukan selama kegiatan magang berlangsung.

3.5.1 Kendala yang ditemukan

.1 Tidak adanya dokumentasi yang baku

Permasalahan pertama yang penulis hadapi adalah permasalahan ketika pertama kali penulis memasuki magang di PT KPSG, penulis diberikan project yang hanya dikerjakan oleh penulis sendiri. Hal tersebut tentu membuat siapapun yang baru masuk kedalam project tersebut kesusahan untuk mengerti alur, flow program, dan juga source code dari project yang ada. Terlebih lagi programmer terdahulu dalam project ini tidak mempunyai dokumentasi bagaimana cara menjalankan programnya, konfigurasi apa yang dibutuhkan, dan lain sebagainya.

.2 Dituntut untuk memahami project meskipun bukan bidang penulis

Dalam kegiatan magang kali ini, penulis diberikan tugas di berbagai macam project, yaitu: Inbranch, HiTo Web frontend, HiTo Mobile, HiTo Backend, dan juga FAS dimana background dari penulis adalah programmer dalam bahasa pemrograman PHP dan Javascript tetapi banyak project tersebut memakai bahasa pemrograman yang tidak pernah penulis sentuh sebelumnya seperti C# dan juga React Native.

3.5.2 Solusi dari permasalahan

.1 Tidak adanya dokumentasi yang baku

Langkah yang penulis ambil dalam kegiatan magang untuk permasalahan ini adalah mempelajari source code sendiri, mencari banyak referensi, juga bertanya kepada senior mengenai project apa yang sedang saya kerjakan dan bertanya seputar bahasa pemrograman dan juga library yang dipakai dalam project ini. Seiring dengan berjalannya waktu, penulis akan selalu lebih familliar dengan project yang sedang dikerjakan walaupun project tersebut tidak memiliki dokumentasinya sekalipun.

.2 Dituntut untuk memahami project meskipun bukan bidang penulis

Dalam dunia kerja, tentu dituntut untuk beradaptasi terhadap situasi di lingkungan pekerjaan merupakan hal yang sangat lumrah terjadi. Hal ini juga penulis hadapi dan untuk mengatasi hal tersebut, penulis mencoba untuk selalu memiliki rasa penasaran terhadap teknologi yang digunakan dalam setiap project yang dikerjakan, memiliki rasa pantang menyerah, dan keingintahuan yang tinggi sehingga pemahaman penulis tentang project tersebut maupun teknologi yang digunakan selalu bertambah setiap harinya. Selain itu juga penulis memiliki keyakinan untuk mempunyai rasa kepemilikan akan project yang sedang dikerjakan sehingga project tersebut akan *ter-deliver* dengan baik di kemudian hari dan tidak meninggalkan kesan yang buruk ketika penulis sudah tidak *meng-handle* project tersebut lagi.