

ANALISIS SISTEM OTENTIKASI PADA BADAN PENGELOLAAN PENDAPATAN, KEUANGAN DAN ASET DAERAH KABUPATEN BREBES

15 Maret 2019

Priyanto Tamami, S.Kom.

1 SASARAN DAN BATASAN SISTEM

Sasaran dari sistem ini adalah mampu untuk melakukan otentikasi terhadap pengguna (*resource owner*) yang melakukan akses ke *resource server* melalui aplikasi klien (*user agent*).

Batasan dari sistem ini yaitu hanya melingkupi proses otentikasi terhadap seluruh pengguna (*resource owner*) yang telah terdaftar, kemudian memberikan hak kepada pengguna (*resource owner*) yang karena kewenangannya ditunjuk sebagai administrator sistem, sehingga mampu untuk melakukan akses ke *resource server* yang melayani pengaturan atau pengelolaan daftar pengguna (*resource owner*) dan aplikasi klien (*user agent*) di Badan Pengelolaan Pendapatan, Keuangan dan Aset Daerah Kabupaten Brebes.

2 ARSITEKTUR SISTEM

Seperti membangun sebuah gedung atau bangunan, agar bangunan dapat berdiri dengan kokoh dan memudahkan dalam pemeliharaan, maka diperlukan sebuah arsitektur, sedangkan pada pembuatan sistem aplikasi maka diperlukan sebuah arsitektur sistem aplikasi yang tujuannya pun agar kokoh dalam artian stabil dengan sedikit permasalahan yang muncul, dan memberikan kemudiahian pada saat

pemeliharaan atau penambahan fitur pada sistem aplikasi yang sudah jadi.

Arsitektur sistem akan terbagi menjadi 3 (tiga) bagian, yaitu :

- a. Bagian Basis Data
- b. Bagian Logika Aplikasi
- c. Bagian Tampilan Aplikasi

3 DESKRIPSI SUB SISTEM

3.1 Bagian Basis Data

Pada bagian ini akan terdapat beberapa tabel yang digunakan untuk melakukan otentikasi pengguna (*resource owner*) beserta hak aksesnya (*role*), dan sebuah tabel yang digunakan untuk melakukan otorisasi bagi aplikasi klien (*user agent*).

3.2 Bagian Logika Aplikasi

Pada bagian ini nantinya akan terdiri dari diagram struktur dan diagram perilaku dari aplikasi, bagaimana masing-masing komponen saling bertukar informasi, bagaimana alur data dari basis data ke bagian tampilan dan sebaliknya dimodelkan pada bagian ini.

Bagian ini bisa disebut inti dari aplikasi, karena bagian ini yang nantinya mengontrol transfer data antar komponen dan lapisan.

3.3 Bagian Tampilan Aplikasi

Istilah lainnya biasa dikenal dengan antar muka (*user interface*) pengguna (*resource owner*), bagian ini yang nantinya akan menjadi desain atau model dari tampilan yang berhadapan langsung dengan pengguna (*resource owner*). Bagian ini yang

nantinya mengumpulkan informasi untuk disampaikan kepada bagian logika aplikasi untuk diproses, bagian ini pula yang nantinya menampilkan informasi yang diproses oleh sistem untuk dapat dibaca dan dipahami oleh pengguna (*resource owner*).

4 PERTIMBANGAN KHUSUS KINERJA SISTEM

Karena sistem datanya terpusat, yaitu tersimpan pada peladen (*server*) basis data Postgresql, maka perawatan atau pemeliharaan mutlak harus dilakukan pada sisi peladen (*server*) basis data yang diakibatkan dari peningkatan jumlah data transaksi perekaman yang selalu bertambah dan berubah dari sisi ukuran untuk setiap harinya.

Hal yang perlu dilakukan adalah pembersihan terhadap berkas *log* (pencatatan), yaitu berkas yang digunakan oleh sistem basis data PostgreSQL untuk mencatatkan kegiatan atau aktivitas yang telah dilakukannya selama beroperasi melayani permintaan data. Pembersihan ini dilakukan secara periodik karena transaksi perubahan yang terjadi sifatnya insidental, sehingga kegiatan ini dapat dilakukan dalam rentang bulanan.

Selain peladen (*server*) sistem basis data, peladen (*server*) aplikasi *web* pun perlu mendapatkan perhatian pada sisi perawatan atau pemeliharaan, dimana peladen (*server*) aplikasi *web* pun memiliki berkas *log* tersendiri yang perlu dibersihkan secara rutin karena menyimpan transaksi antara komputer klien dan peladen (*server*).

Hal lain yang perlu dilakukan sebagai langkah pengamanan data yaitu dilakukannya duplikasi data (*backup data*) sehingga apabila terjadi hal-hal yang tidak diinginkan, data dapat langsung dikembalikan atau dipulihkan (*recovery*). Ini pun

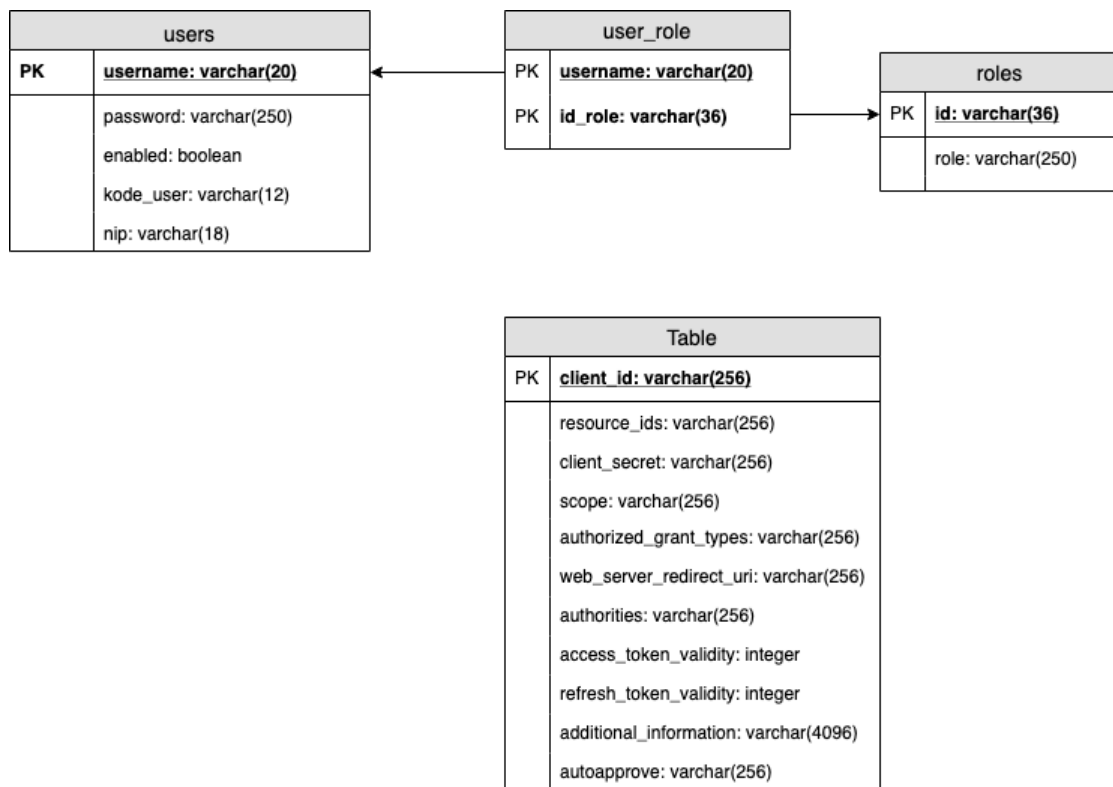
sebaiknya dilakukan secara rutin, sebaiknya dalam periode harian.

5 HASIL PEMODELAN

Hasil pemodelan untuk sistem otentikasi ini nantinya akan dibagi menjadi beberapa diagram dasar pada beberapa bagian dari arsitektur sistem, berikut rincian mengenai hasil pemodelan tersebut :

5.1 Bagian Basis Data

Struktur basis data yang dibangun untuk kebutuhan sistem otentikasi pada Badan Pengelolaan Pendapatan, Keuangan dan Aset Daerah Kabupaten Brebes adalah seperti pada gambar 1 berikut ini :



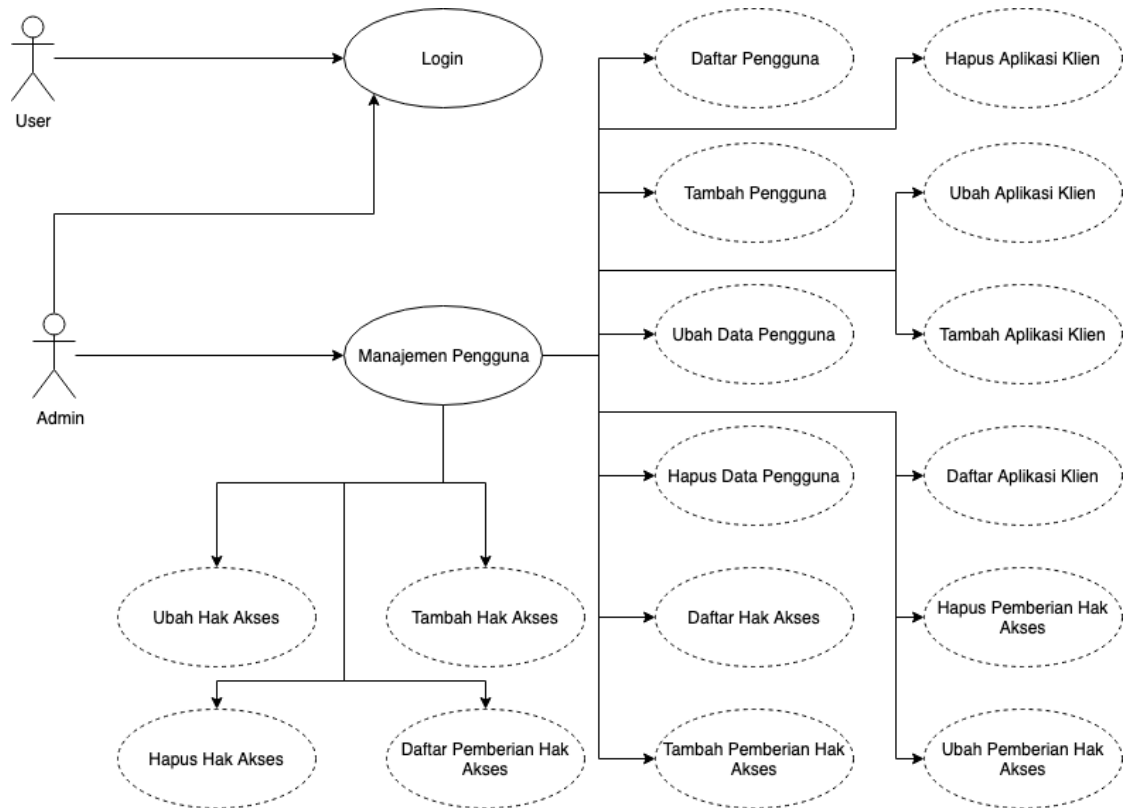
Gambar 1: Diagram Tabel Yang Digunakan

5.2 Bagian Logika Aplikasi

Ada beberapa pemodelan logika aplikasi di dunia teknologi informasi, jika pengembangan dilakukan dengan bahasa pemrograman terstruktur, maka akan menggunakan *flowchart* atau diagram alir sebagai alat untuk pemodelan, karena yang akan digunakan sekarang adalah bahasa pemrograman Kotlin yang menggunakan metodologi orientasi objek, maka digunakan *Unified Modeling Language* (UML) yang akan menggambarkan struktur sistem dari awal sampai akhir. Berikut adalah pemodelan atau diagram yang menggambarkan bagian sistem otentikasi pada Badan Pengelolaan Pendapatan, Keuangan dan Aset Daerah Kabupaten Brebes bekerja.

5.2.1 Diagram *Use Case*

Diagram *use-case* dari keseluruhan sistem yang akan dibangun adalah seperti gambar 2 ini :



Gambar 2: Diagram *Use-Case*

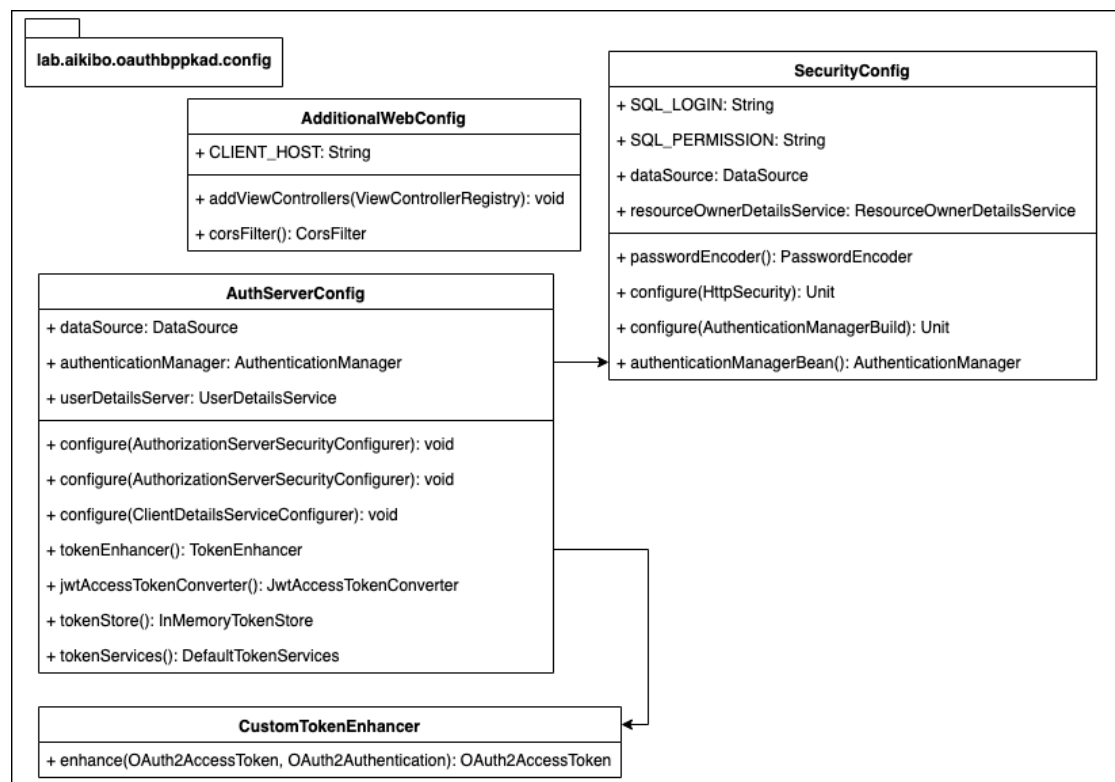
Sistem otentikasi hanya akan menampilkan halaman *login* saja bagi pengguna selain administrator, namun yang memiliki kewenangan sebagai administrator dapat melakukan akses ke manajemen atau pengelolaan daftar pengguna setelah melakukan *login* pada aplikasi klien (*user agent*)

5.2.2 Diagram *Class*

Diagram *class* pada sistem otentikasi ini akan terbagi menjadi 3 (tiga) bagian, yaitu pada *OAuth Server*, *Resource Server*, dan aplikasi klien (*user agent*).

Diagram *class* untuk *OAuth Server* sendiri akan menjadi beberapa bagian yang dipisahkan dengan penamaan paketnya, berikut adalah daftar paket yang membentuk sebuah *OAuth Server* :

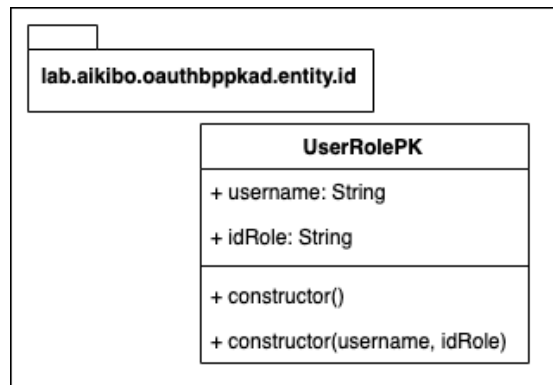
1. Paket yang pertama yaitu `lab.aikibo.oauthbppkad.config` yang terlihat seperti pada gambar 3 berikut ini :



Gambar 3: Diagram *Class* OAuth Bagian 1

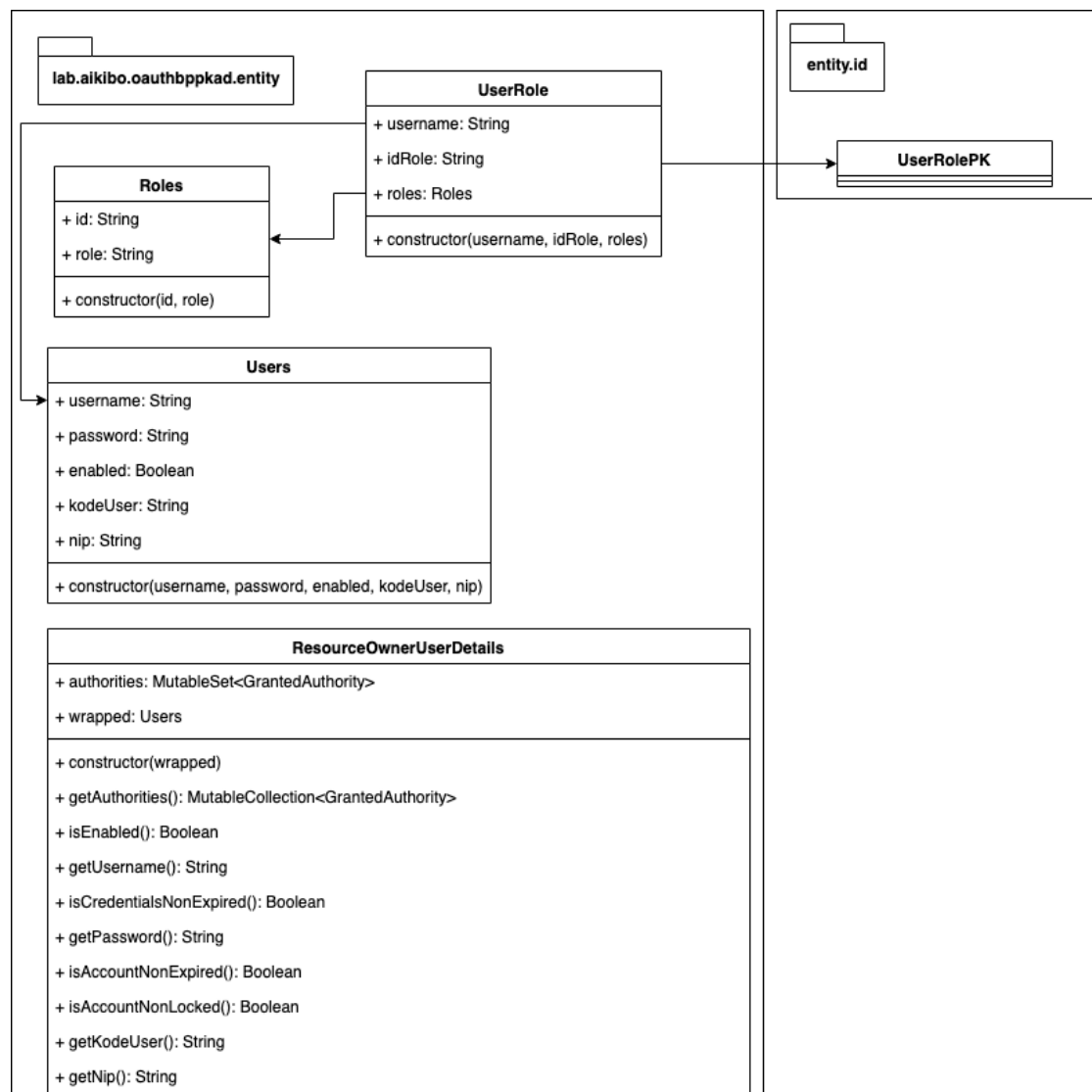
Paket ini berhubungan dengan konfigurasi yang terjadi untuk memberikan layanan *service OAuth* yang ditawarkan.

2. Paket berikutnya adalah `lab.aikibo.oauthbppkad.entity.id` yang berisi kelas-kelas pembentuk *primary key* dari tabel yang memiliki *composite key* atau bentuk *primary key* yang terdiri dari beberapa *field* atau kolom. Berikut adalah diagram *class* pembentuknya :



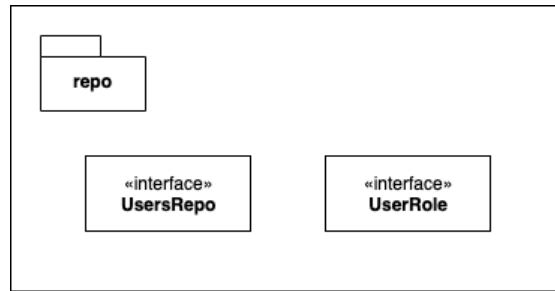
Gambar 4: Diagram *Class* OAuth Bagian 2

3. Paket berikutnya adalah `lab.aikibo.oauthbppkad.entity` yang terdiri dari beberapa kelas entitas yang digunakan untuk memetakan struktur tabel pada sistem database ke dalam objek Kotlin. Berikut adalah diagram *class* dari paket ini :



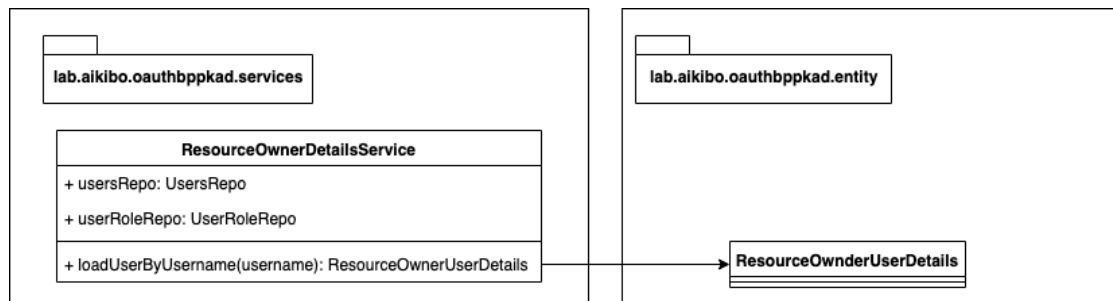
Gambar 5: Diagram *Class* OAuth Bagian 3

- Paket berikutnya adalah `lab.aikibo.oauthbppkad.repo` yang berisi dari deklarasi *interface* dimana fungsinya adalah untuk melakukan operasi manipulasi data pada sistem basis data. Daftar *interface* dari paket ini adalah seperti berikut :



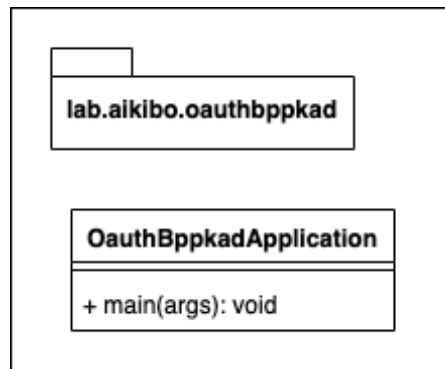
Gambar 6: Diagram *Class* OAuth Bagian 4

5. Paket berikutnya adalah `lab.aikibo.oauthbppkad.services` yang berisi kelas-kelas pengolah data baik yang akan disimpan melalui *interface-interface* di paket `repo`, atau yang datang dari paket `repo` untuk kemudian dikirimkan ke luar. Berikut adalah kelas pembentuk dari paket ini :



Gambar 7: Diagram *Class* OAuth Bagian 5

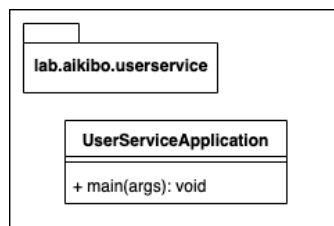
6. Paket berikutnya adalah `lab.aikibo.oauthbppkad` yang berisi sebuah kelas yang nantinya akan dieksekusi pertama kali saat layanan dari *OAuth Server* berjalan. Berikut adalah diagram *class* dari paket ini :



Gambar 8: Diagram *Class* OAuth Bagian 6

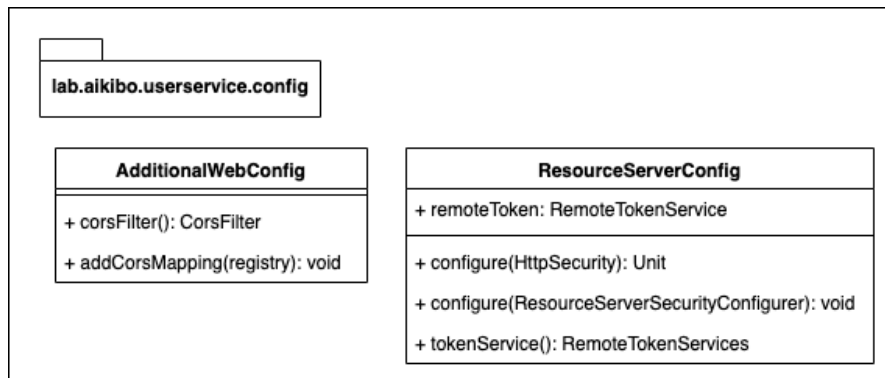
Diagram *class* untuk *Resource Server* pun akan terdiri dari beberapa paket untuk memperjelas bagan dari diagram *class* yang terbentuk, berikut adalah daftar nya :

1. Paket **lab.aikibo.userservice** akan terdiri dari sebuah kelas saja, dimana kelas ini akan dijalankan pertama kali saat layanan dari *Resource Server* memulai aktivitasnya. Diagram *class* dari paket ini adalah seperti berikut :



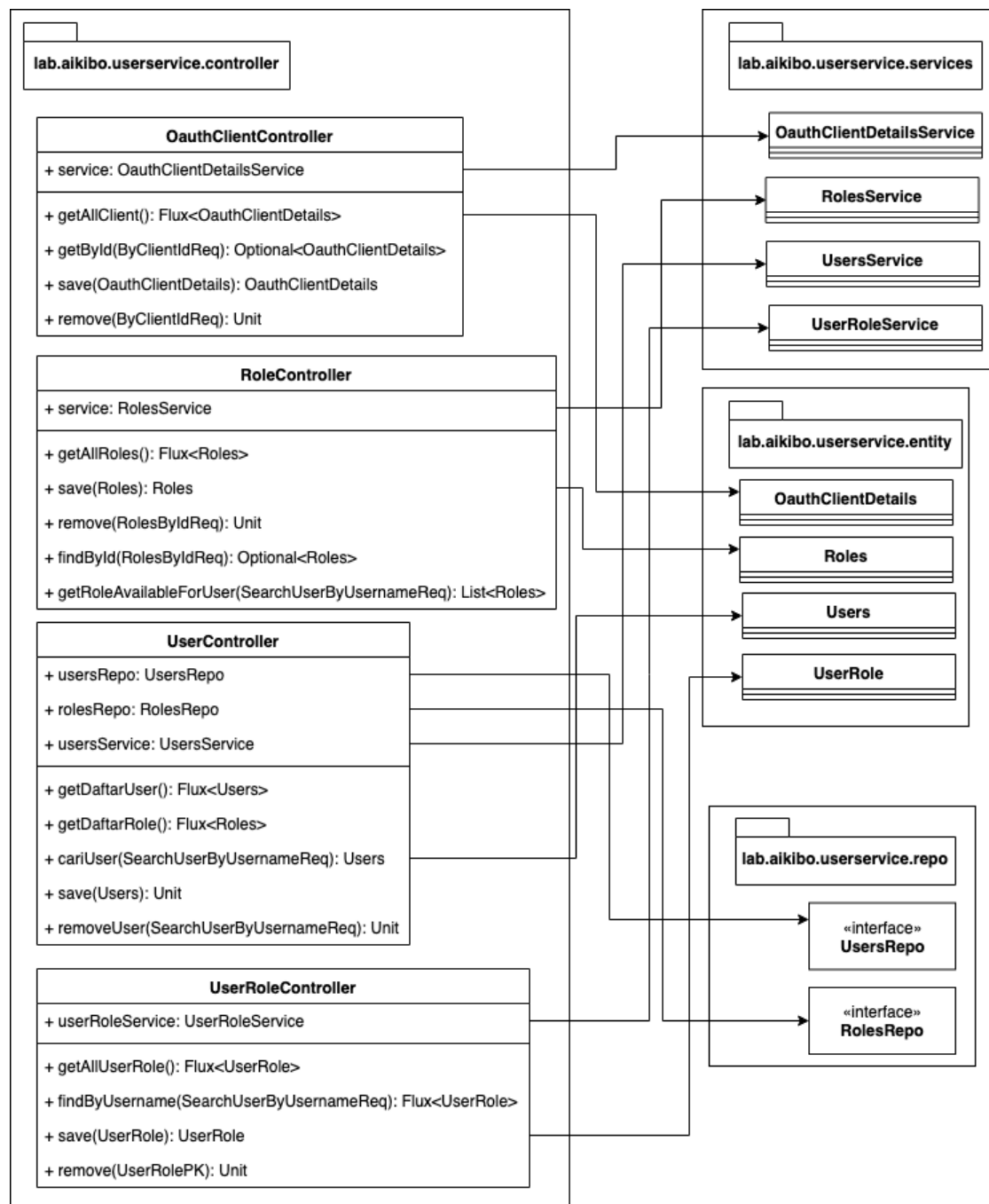
Gambar 9: Diagram *Class* *Resource Server* Bagian 1

2. Paket **lab.aikibo.userservice.config** akan terdiri dari beberapa kelas untuk melakukan konfigurasi *Resource Server*. Diagram *class* dari paket ini adalah seperti berikut :



Gambar 10: Diagram *Class Resource Server* Bagian 2

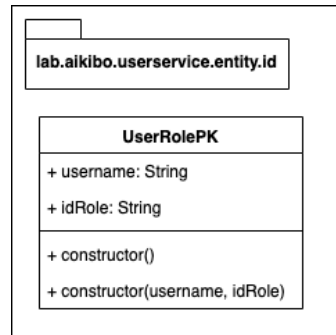
3. Paket `lab.aikibo.userservice.controller` akan terdiri dari beberapa kelas yang mendefinisikan URI (*Uniform Resource Identifier*) dari layanan yang diberikan oleh *Resource Server*. Berikut adalah diagram *class* untuk paket ini :



Gambar 11: Diagram *Class Resource Server* Bagian 3

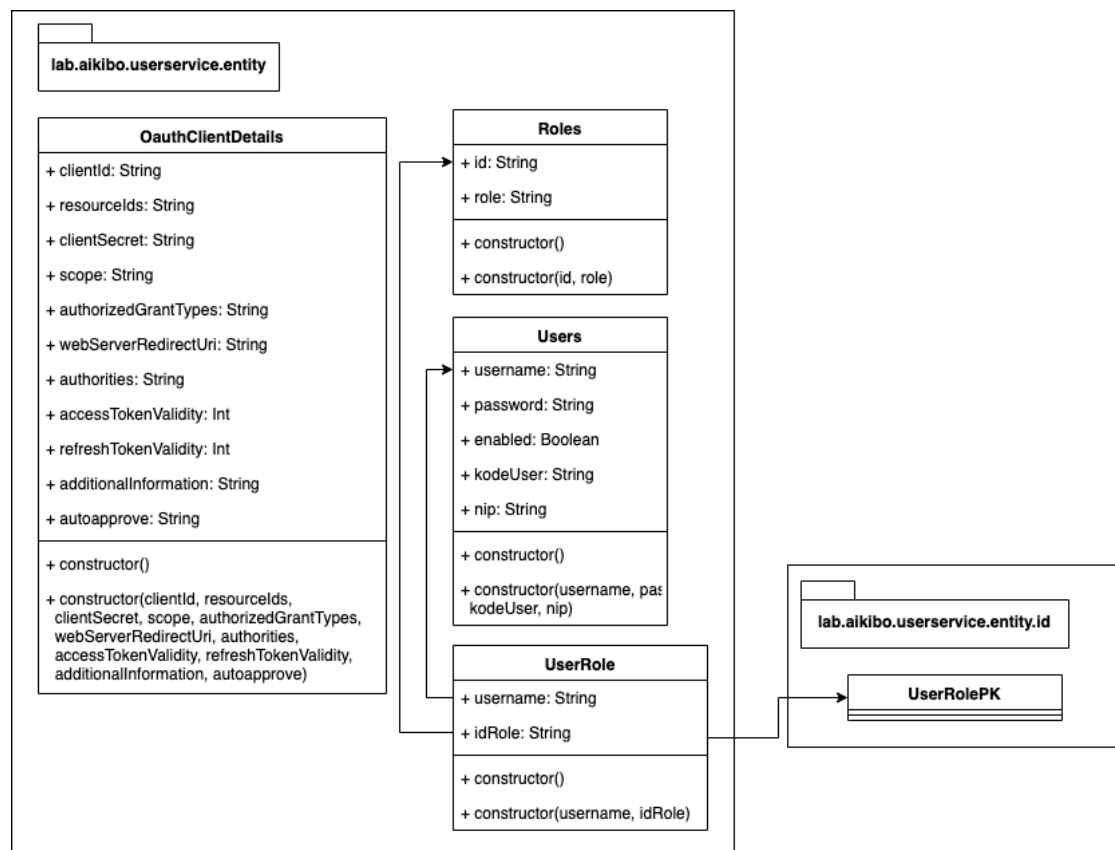
4. Paket `lab.aikibo.userservice.entity.id` akan terdiri dari kelas-kelas

pembentuk *primary key* bagi tabel yang memiliki *primary key* berupa *composite key* atau lebih dari 1 (satu) *field* atau kolom. Diagram kelas dari paket ini adalah seperti berikut :



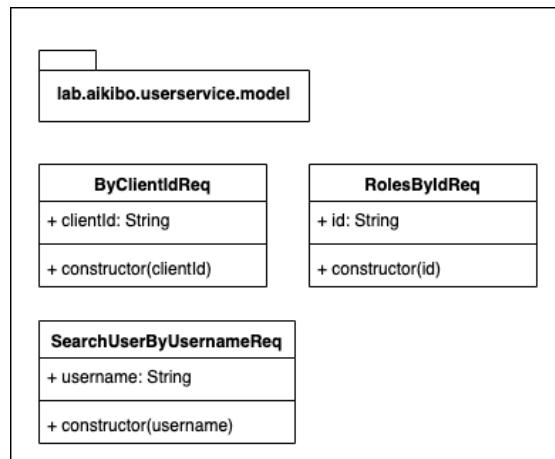
Gambar 12: Diagram *Class Resource Server* Bagian 4

5. Paket `lab.aikibo.userservice.entity` akan terdiri dari kelas-kelas yang digunakan untuk memetakan tabel ke dalam objek Kotlin, jadi tiap tabel yang terdapat pada sistem basis data, akan memiliki pasangan masing-masing di objek Kotlin. Gambar diagram *class* dari paket ini adalah seperti berikut :



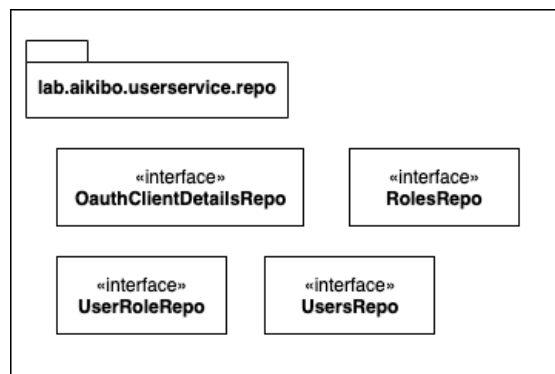
Gambar 13: Diagram *Class Resource Server* Bagian 5

6. Paket `lab.aikibo.userservice.model` akan terdiri dari kelas-kelas yang digunakan untuk pemetaan objek JSON yang datang dari *request* aplikasi klien (*user agent*). Terkadang kelas-kelas yang di dalam paket ini pun difungsikan untuk membentuk objek JSON sebagai *response* atau *request* yang terjadi dari aplikasi klien (*user agent*). Diagram *class* untuk paket ini adalah seperti berikut :



Gambar 14: Diagram *Class Resource Server* Bagian 6

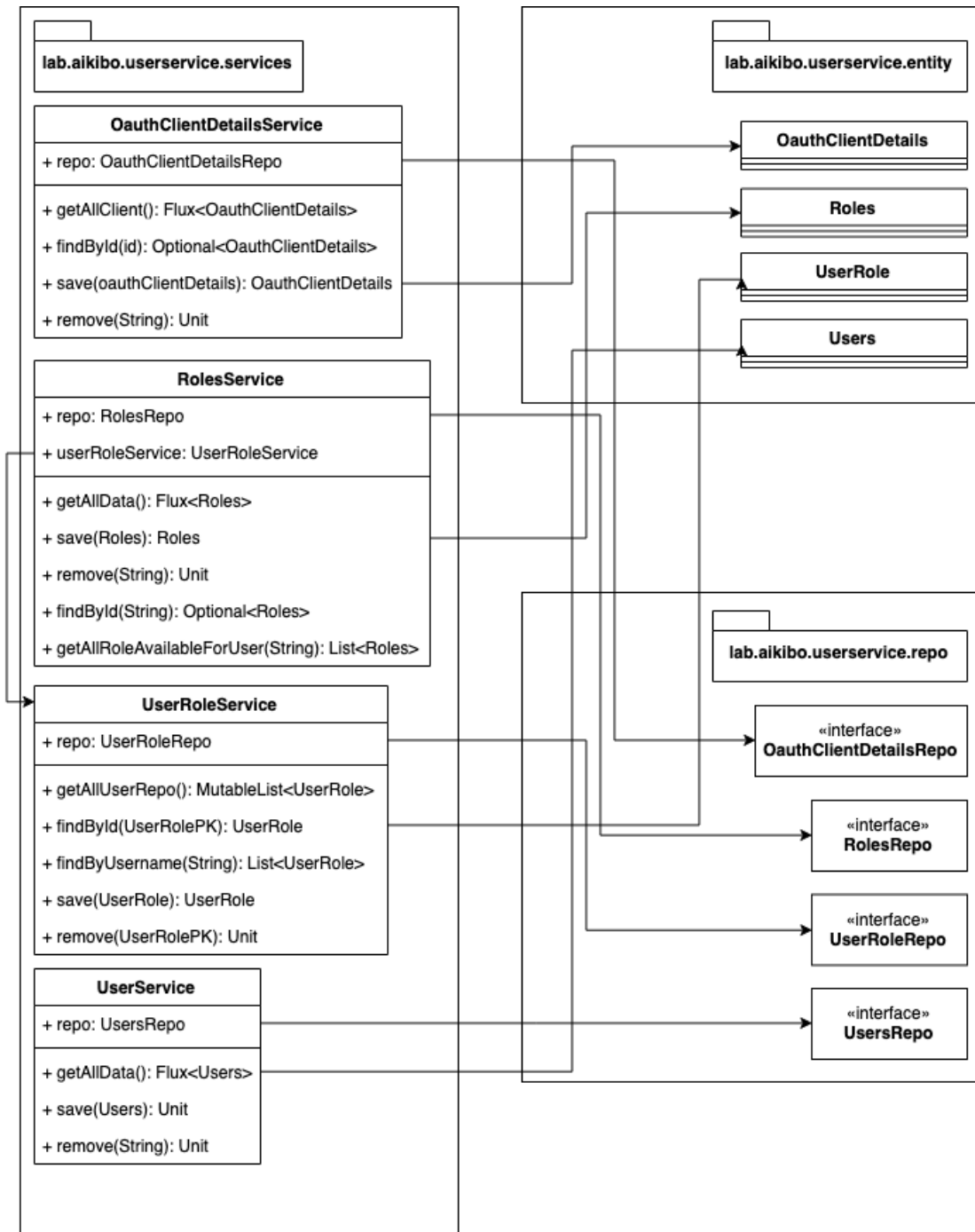
7. Paket `lab.aikibo.userservice.repo` akan terdiri dari beberapa *interface* yang fungsinya adalah untuk melakukan manipulasi data pada sistem basis data. Diagram *class* dari paket ini adalah seperti gambar berikut :



Gambar 15: Diagram *Class Resource Server* Bagian 7

8. Paket `lab.aikibo.userservice.services` akan terdiri dari beberapa kelas yang fungsinya untuk melakukan penyesuaian parameter pada saat akan melakukan manipulasi di sistem basis data melalui *interface* pada paket `repo`, atau pada saat melakukan pengambilan nilai dari sistem basis data. Diagram

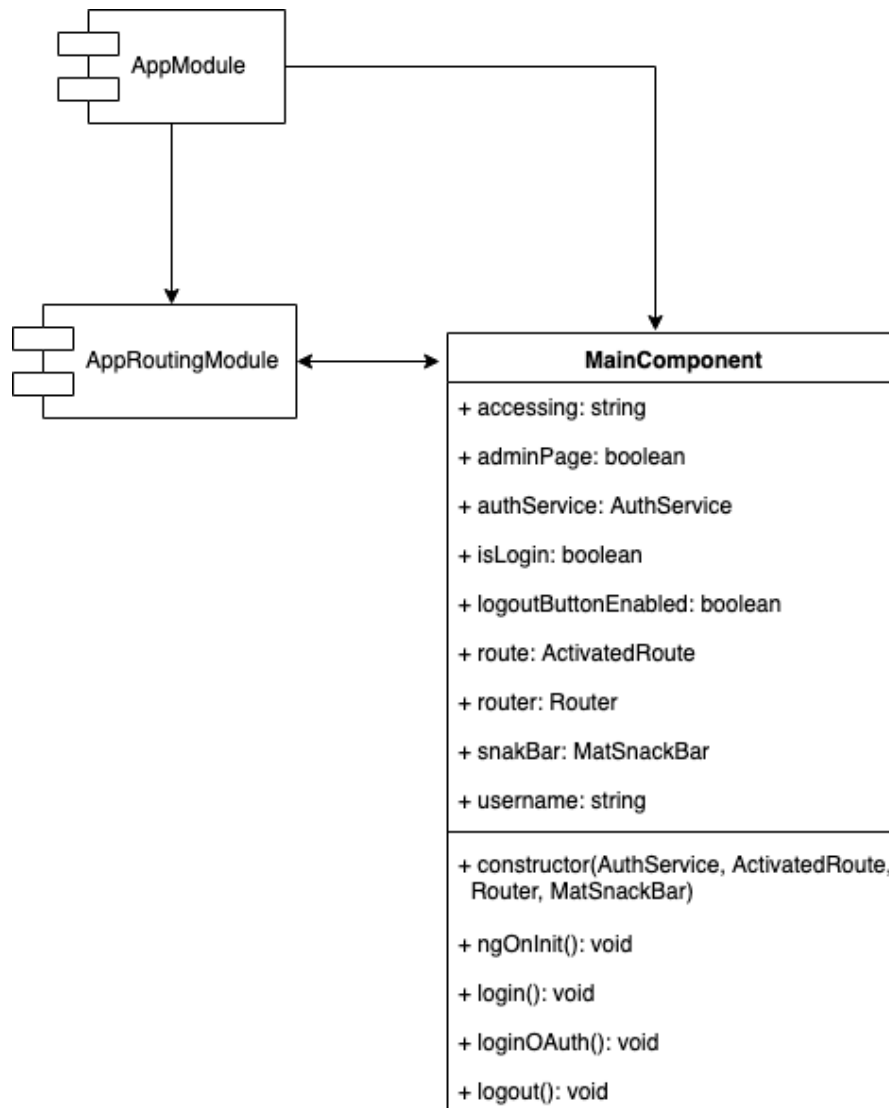
class untuk paket ini adalah seperti berikut :



Gambar 16: Diagram *Class Resource Server* Bagian 8

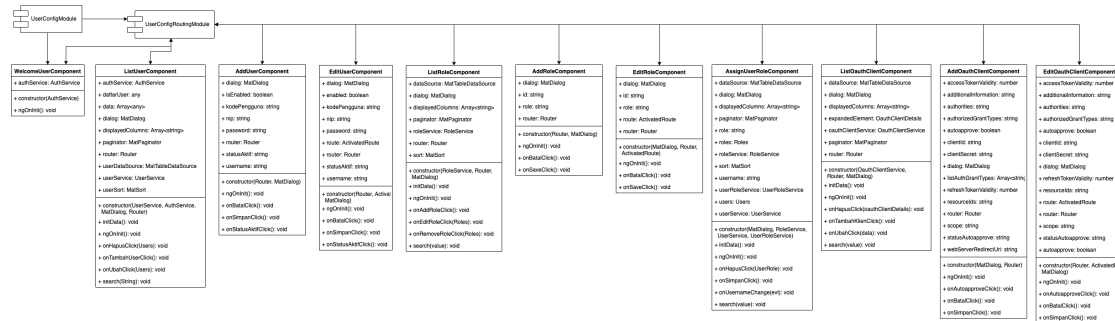
Sekarang ke bagian *front-end* yang akan dibangun dengan Angular . Karena menganut konsep *Object Oriented* juga, sehingga kita dapat melakukan pemodelan dengan UML, termasuk diagram *class* yang akan dibangun.

Untuk membentuk rangka dari aplikasi klien (*user agent*), beberapa kelas yang dibutuhkan akan terdapat pada diagram *class* berikut :



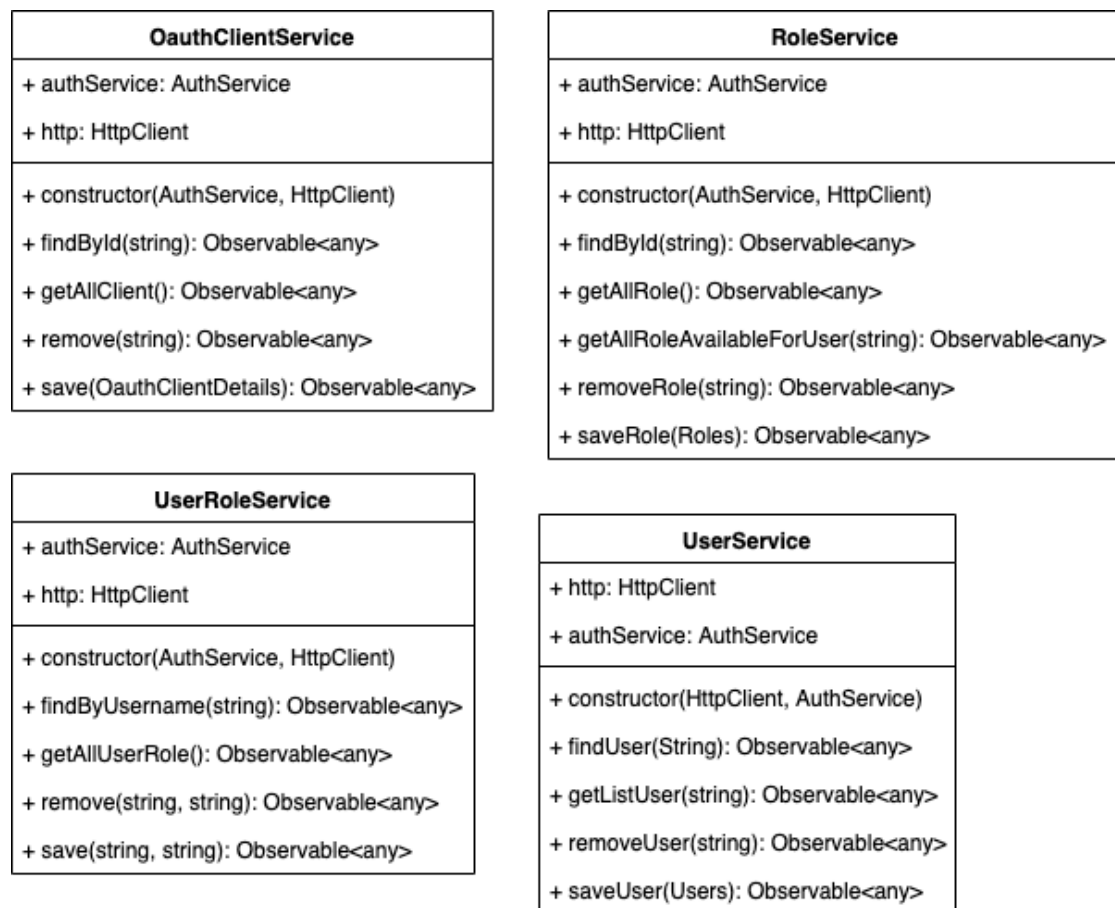
Gambar 17: Diagram *Class* Aplikasi Klien Bagian 1

Untuk aplikasi klien (*user agent*) pengelolaan daftar pengguna akan terdiri dari beberapa kelas pembentuk seperti diagram *class* berikut :



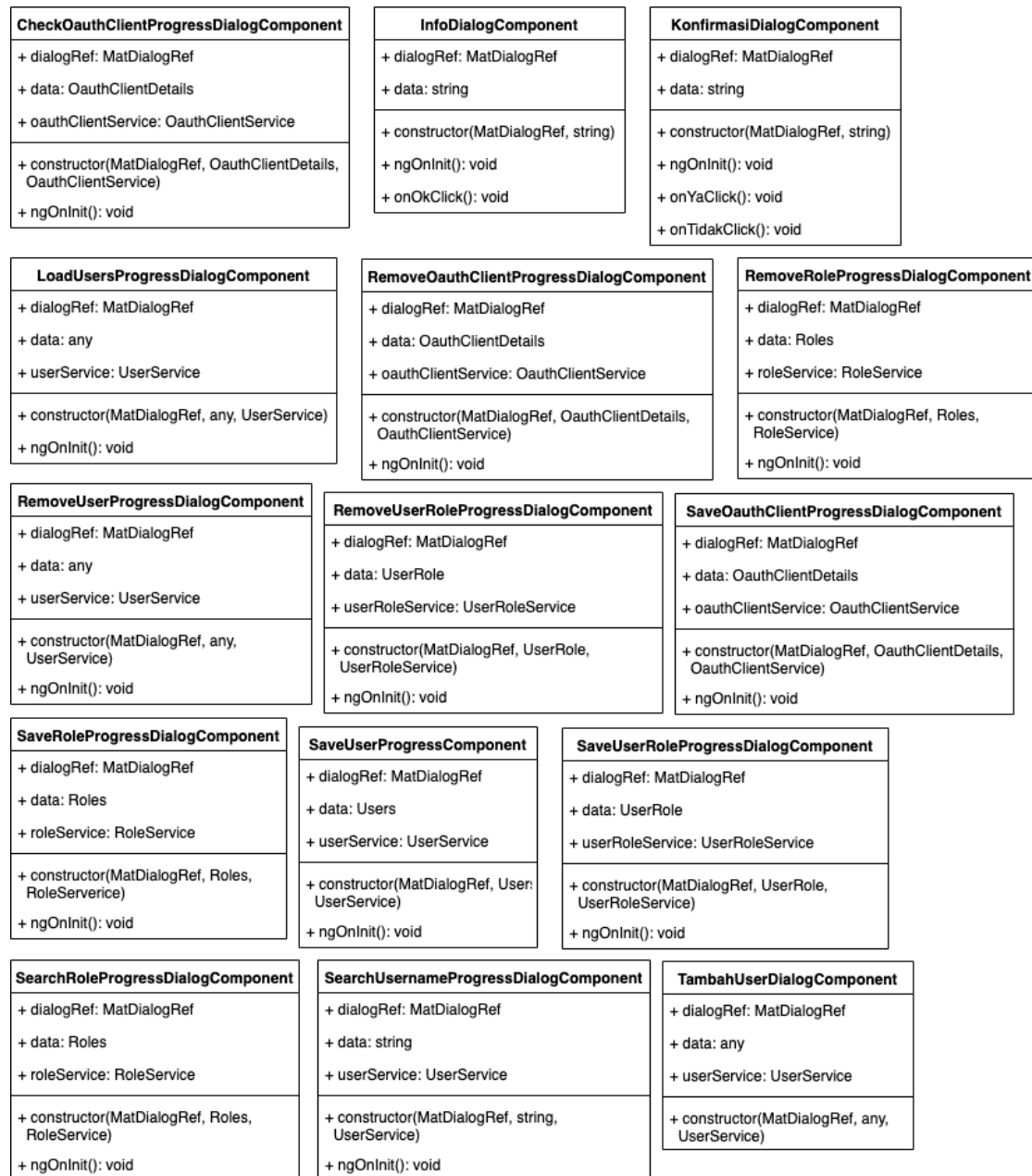
Gambar 18: Diagram *Class* Aplikasi Klien Bagian 2

Adapun kelas-kelas pendukung yang digunakan untuk melakukan akses ke *resource server* dilakukan oleh beberapa *services*, berikut adalah diagram *class* dari kelas-kelas pendukung tersebut :



Gambar 19: Diagram *Class Services*

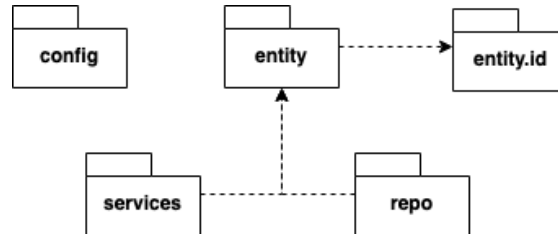
Berikutnya adalah kelas-kelas yang membentuk jendela dialog yang digunakan untuk menampilkan informasi terbatas bagi pengguna aplikasi, diagram *class* dari jendela dialog ini adalah seperti berikut :



Gambar 20: Diagram *Class* untuk *Dialog*

5.2.3 Diagram *Package*

Diagram ini akan menunjukkan struktur desain sistem pada level *package*. Diagram *package* untuk *oauth server* adalah seperti pada gambar 21 berikut ini :

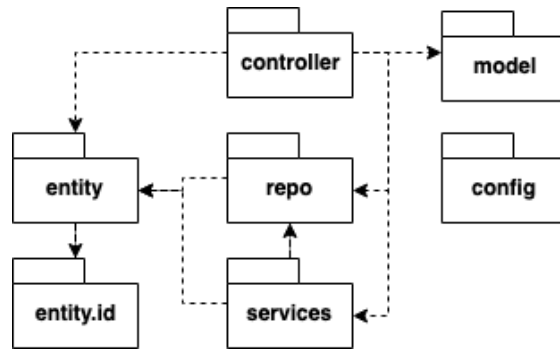


Gambar 21: Diagram *Package* pada *OAuth Server*

Berikut penjelasan dari tiap *package* :

1. *Package* **config** berfungsi untuk melakukan konfigurasi terhadap *OAuth Server* yang akan berjalan, yang tidak berkait dengan beberapa *package* lainnya.
2. *Package* **entity.id** ini sebagai wadah dari kelas-kelas pembentuk *primary key* yang digunakan oleh kelas-kelas yang terdapat dalam *package* **entity**.
3. *Package* **entity** berisi kelas-kelas yang digunakan untuk memetakan tabel dari sistem basis data.
4. *Package* **repo** berisi *interface* untuk melakukan operasi data ke sistem basis data.
5. *Package* **services** berisi kelas-kelas yang melakukan pengolahan data sebelum dilakukan proses simpan data ke sistem basis data atau setelah melakukan pengambilan data dari sistem basis data.

Diagram *package* untuk *resource server* adalah seperti pada gambar berikut :



Gambar 22: Diagram *Package* pada *Resource Server*

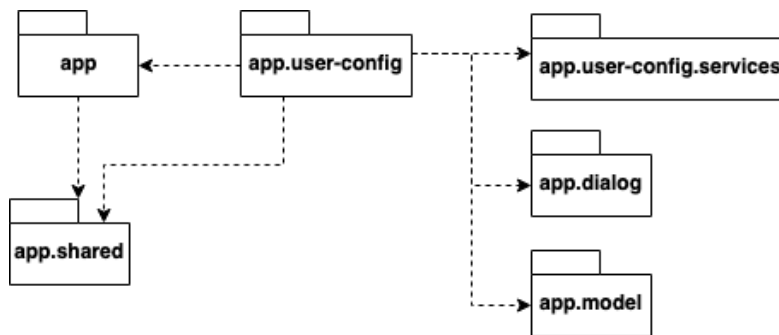
Penjelasan dari paket-paket tersebut adalah seperti berikut ini :

1. *Package* **entity** adalah tempat bagi kelas-kelas yang digunakan untuk me-
metakan tabel-tabel yang terdapat dalam basis data.
2. *Package* **entity.id** adalah tempat untuk kelas-kelas yang menjadi definisi
primary key bagi kelas-kelas entitas yang terdapat pada *package* **entity**.
3. *Package* **controller** adalah tempat bagi kelas-kelas yang mendefinisikan
URI (*Uniform Resource Identifier*) dari layanan-layanan yang diberikan oleh
Resource Server.
4. *Package* **repo** adalah tempat bagi *interface-interface* yang digunakan untuk
melakukan akses atau operasi data pada sistem basis data.
5. *Package* **services** adalah tempat bagi kelas-kelas yang melakukan adaptasi
data yang akan disimpan pada sistem basis data atau akan ditampilkan dari
sistem basis data.
6. *Package* **model** adalah tempat bagi kelas-kelas yang digunakan sebagai *mo-
del* untuk melakukan *respon* atau menerima *request* dari aplikasi klien (*user*

agent). Nantinya kelas-kelas di dalam *package* ini akan secara otomatis di-konversi ke dalam JSON atau sebaliknya, dari format JSON akan bentuk ke dalam objek-objek dari kelas-kelas dalam *package* ini.

7. *Package config* adalah tempat bagi kelas-kelas yang melakukan konfigurasi *framework* dari *Resource Server*.

Untuk bagian *User Interface* atau dalam hal ini adalah aplikasi klien (*user agent*), karena tidak berbentuk paket seperti halnya pada Kotlin, untuk memudahkan akan dianalogikan sebuah paket adalah sebuah kandar / direktori dari tiap kelas yang terbentuk. Diagram *package* dari bagian *front-end* adalah seperti berikut :



Gambar 23: *Package* Diagram dari *Front-End*

Penjelasan dari tiap *package* di atas adalah seperti berikut :

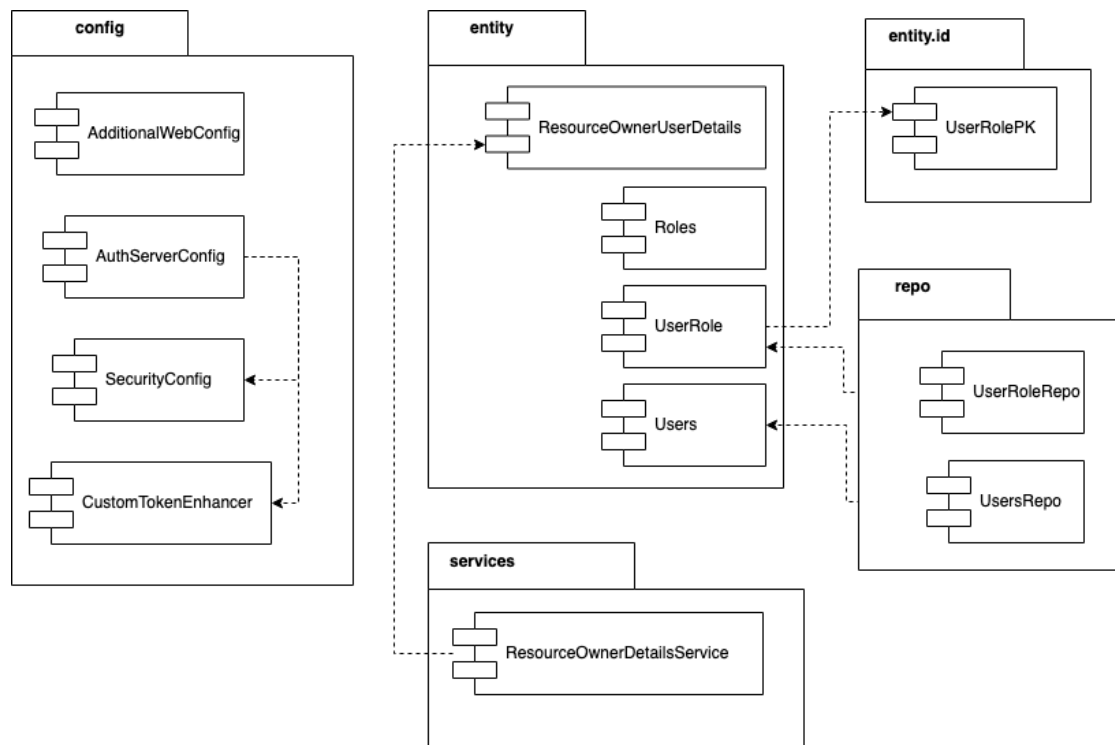
1. *Package app* adalah paket utama dari *project* yang dibangun, nantinya *routing* akan diatur dalam *package* ini yang akan bergantung pada otorisasi dari *package app.shared*.
2. *Package app.shared* adalah paket yang berisi kelas-kelas pendukung yang nantinya akan digunakan dari seluruh *package* untuk menentukan otentikasi yang didapat, dan alamat dari *service-service* yang dapat diakses.

3. *Package* `app.user-config` akan berisi kelas-kelas pembentuk tampilan tatap muka (*user interface*) dari aplikasi manajemen pengguna.
4. *Package* `app.user-config.services` akan berisi kelas-kelas yang berfungsi untuk menyediakan akses atau operasi data dari dan ke *resource server*.
5. *Package* `app.dialog` akan berisi berbagai macam kelas yang berbentuk tampilan *dialog* untuk menampilkan berbagai macam informasi seperti proses simpan data, proses unduh data, konfirmasi aksi, informasi kesalahan, dan sebagainya.
6. *Package* `app.model` akan berisi dari objek-objek yang digunakan untuk menerima *response* dari atau mengirimkan *request* ke *resource server* dalam format JSON.

5.2.4 Diagram *Component*

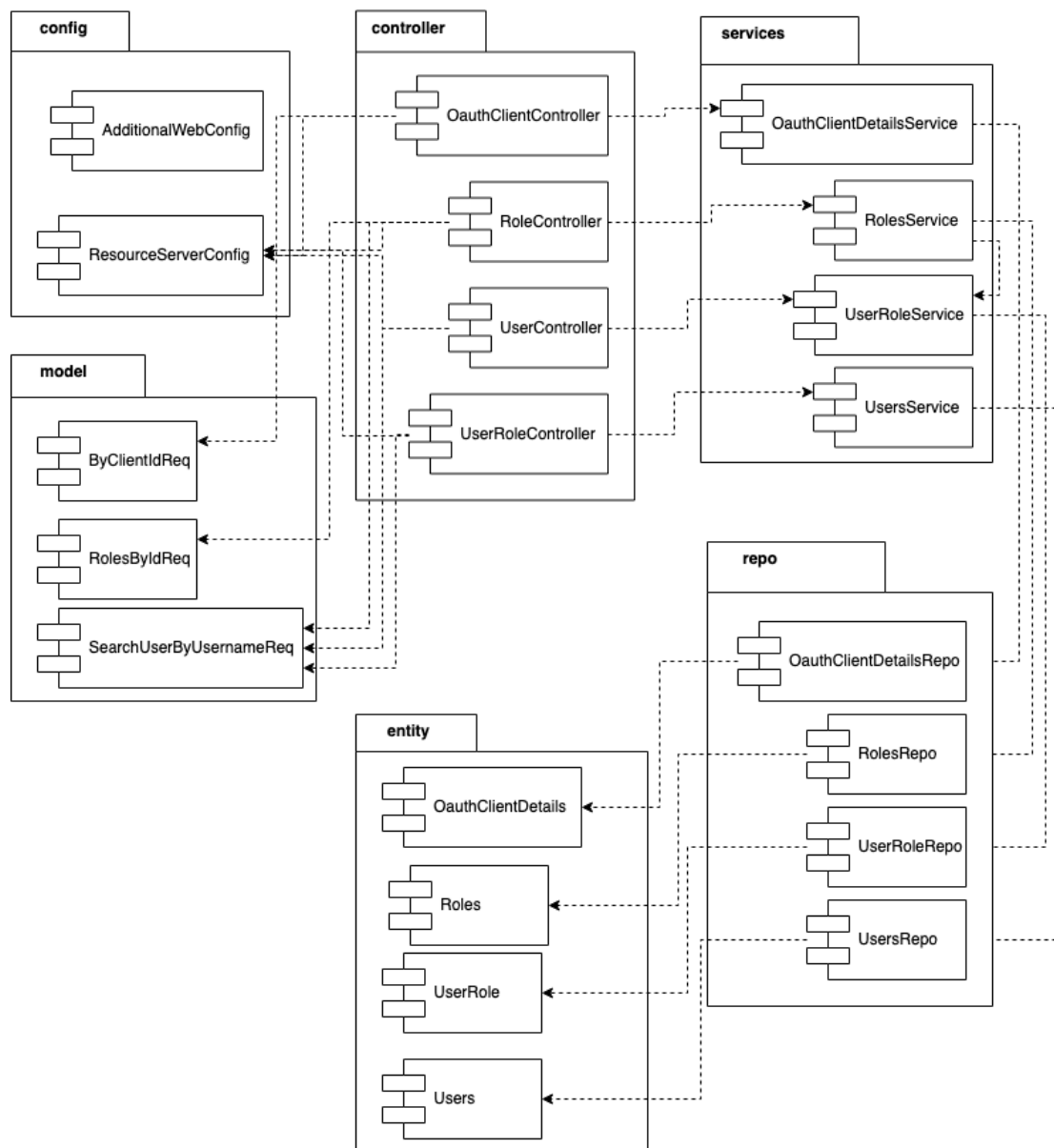
Diagram ini memberikan gambaran hubungan antar komponen, komponen mana yang membutuhkan data dan komponen mana yang memberikan data akan terlihat jelas pada diagram komponen ini. Berikut adalah diagram *component* yang terbagi menjadi beberapa bagian berdasarkan fungsinya :

1. Diagram *Component* pada *Oauth Server* akan terlihat seperti pada gambar berikut :



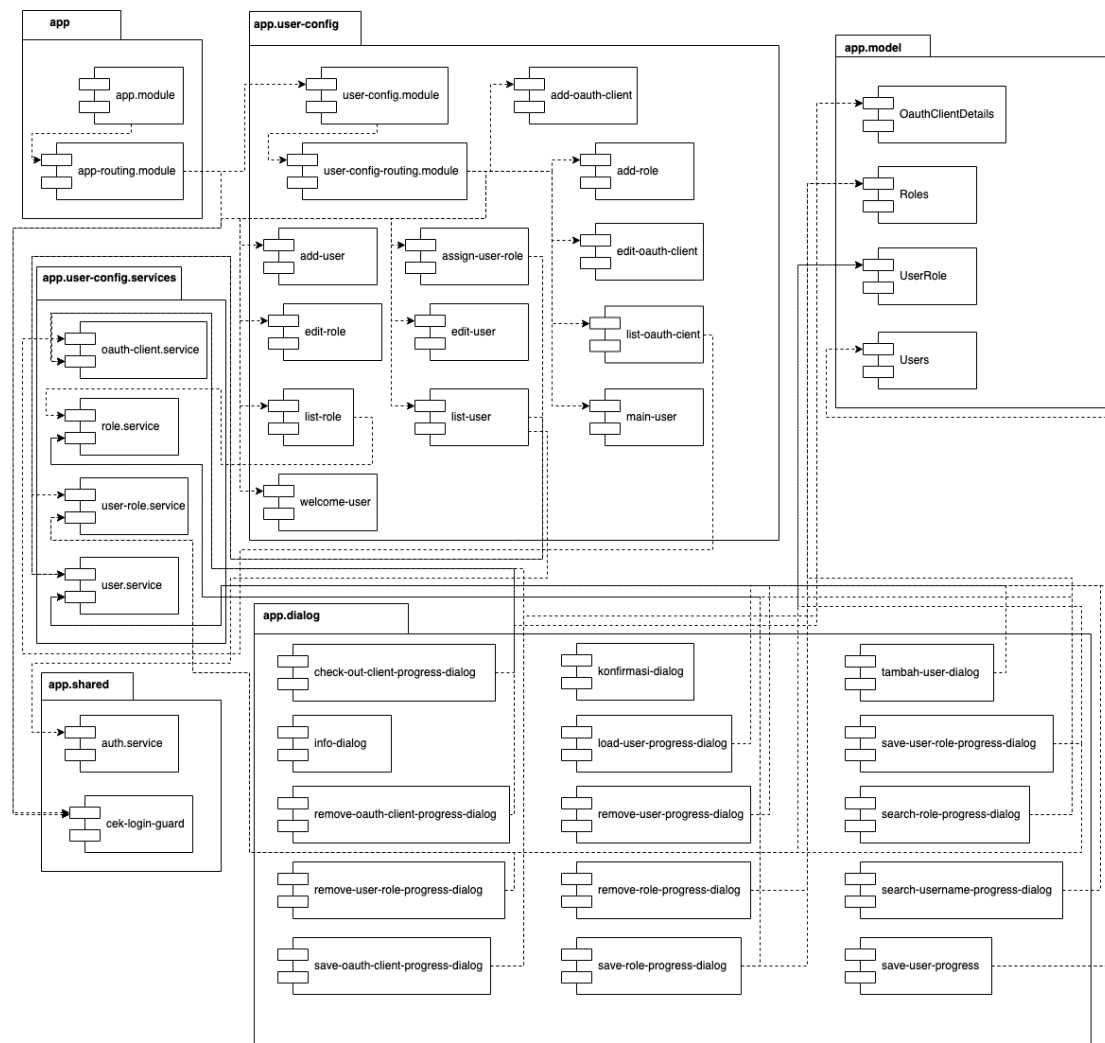
Gambar 24: Diagram *Component* Untuk *OAuth Server*

2. Diagram *Component* pada *Resource Server* akan terlihat seperti pada gambar berikut :



Gambar 25: Diagram *Component* untuk *Resource Server*

- Diagram *Component* pada Aplikasi Klien (*User Agent*) adalah seperti terlihat pada gambar berikut :



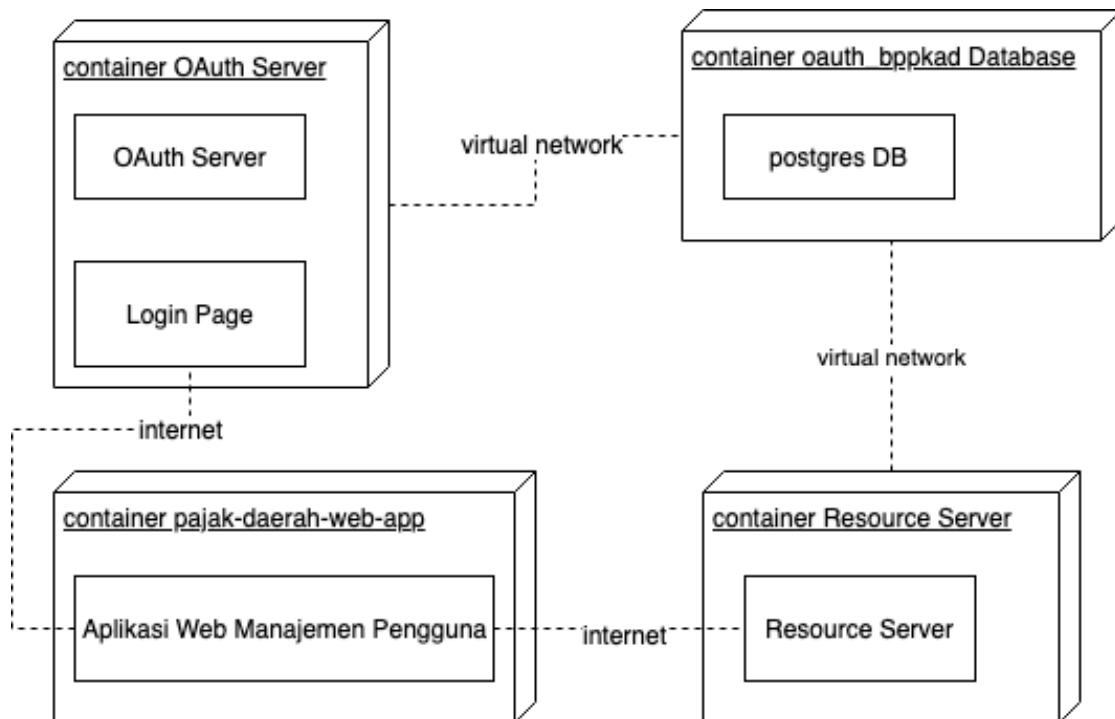
Gambar 26: Diagram *Component* Untuk Aplikasi Klien (*User Agent*)

Diagram ini berisi komponen-komponen yang membangun sistem ini menjadi utuh, diagram ini digunakan untuk menjabarkan artifak sistem secara fisik, dimana berkas harus ditempatkan, apa saja nama paket / kandarnya, dan hubungan antar *class* atau berkas, kemana data atau proses berikutnya akan berjalan.

5.2.5 Diagram *Deployment*

Diagram ini menunjukkan arsitektur dari sistem pada saat didistribusikan dari mesin tempat untuk mengembangkan dan uji coba, ke mesin produksi tempat aplikasi siap untuk melayani pengguna aslinya.

Diagram *deployment* terlihat seperti pada gambar berikut :



Gambar 27: Diagram *Deployment*

Pada diagram tersebut ditunjukkan bahwa akan ada 4 (empat) docker *container* yang masing-masing *container* memiliki layanannya sendiri, pengguna nantinya hanya akan terhubung melalui *container pajak-daerah-web-app* karena ini adalah tatap muka (*user interface*) yang dapat diakses oleh pengguna, kemudian saat pengguna akan melakukan akses terhadap data sensitif, pengguna akan diarahkan (*redirect*) ke halaman *login* untuk melakukan otentikasi, setelah itu akan diarahkan kembali ke Aplikasi Web Manajemen Pengguna, semua aktifitas peng-

guna akan terjadi pada layanan ini.

Simpanan data / basis data akan terhubung dengan *OAuth Server* dan *Resource Server* melalui jaringan virtual yang digunakan sebagai tempat simpanan data dari daftar pengguna (*resource owner*) dan daftar aplikasi klien (*user agent*).

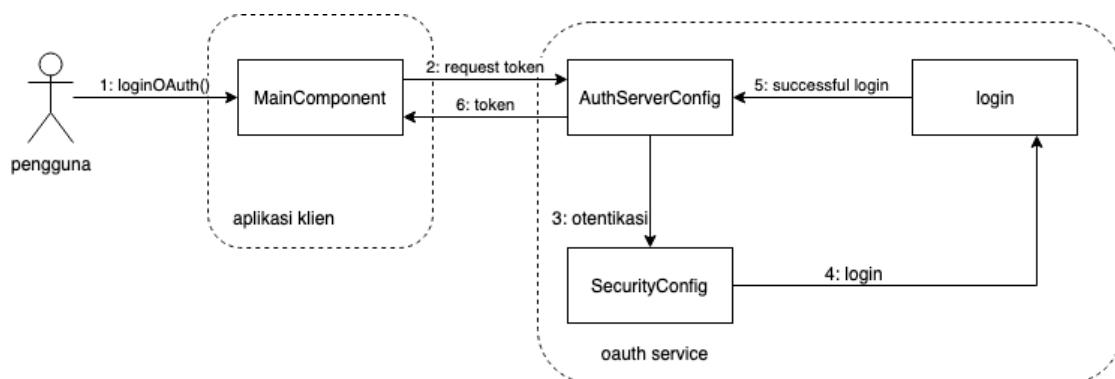
Komunikasi antara Aplikasi Web Manajemen Pengguna dengan *Resource Server* akan sering terjadi setiap pengguna melakukan operasi data. Komunikasi akan terjalin menggunakan arsitektur REST dengan bentuk JSON sebagai format pertukaran datanya.

5.2.6 Diagram *Communication*

Diagram ini menggambarkan interaksi antar objek yang disertai urutan komunikasi dalam bentuk bagan yang bebas. Diagram ini akan terbagi menjadi beberapa bagian berdasarkan fitur yang diberikan oleh sistem otentikasi, berikut adalah daftar diagram yang terbentuk :

- Fitur *Login*

Diagram *communication* untuk fitur ini akan menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) melakukan otentikasi aplikasi, prosesnya akan terlihat seperti gambar diagram *communication* berikut :

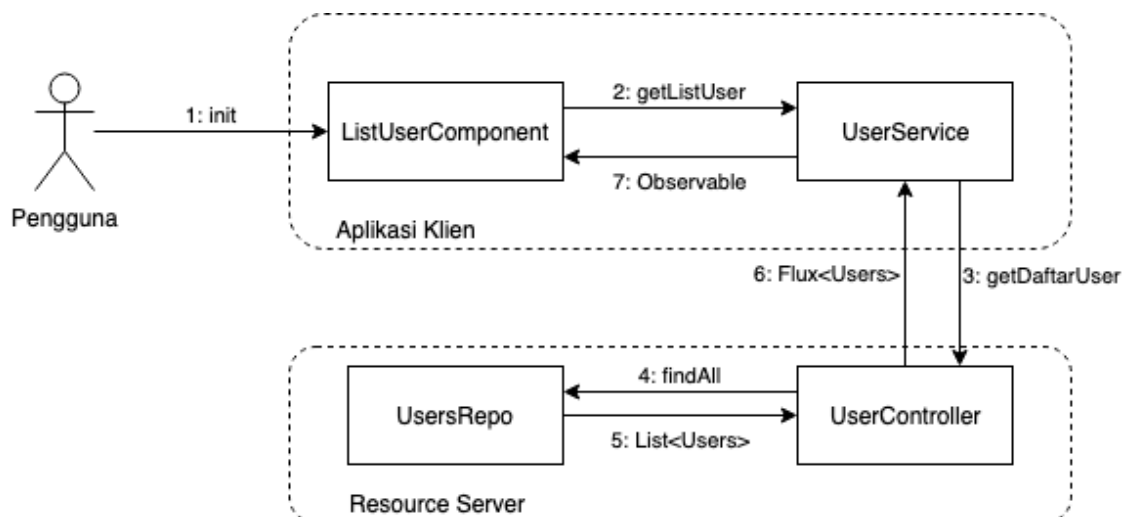


Gambar 28: Diagram *Communication* Untuk Fitur *Login*

Skenario proses yang terjadi adalah seperti berikut :

1. Pengguna mulai melakukan akses ke aplikasi klien (*user agent*)
 2. Aplikasi klien (*user agent*) melakukan *request* token ke *OAuth Server*.
 3. *OAuth Server* melakukan otentikasi dan verifikasi *request* terhadap Pengguna (*resource owner*) dan aplikasi klien (*user agent*)
 4. *OAuth Server* akan mengarahkan pengguna ke halaman *login*.
 5. Setelah proses di halaman *login* berhasil diverifikasi, proses selanjutnya kembali ke kelas `AuthServerConfig` untuk membuat sebuah token berdasarkan pengguna yang telah melakukan *login*.
 6. *OAuth Server* mengirimkan token JWT ke aplikasi klien (*user agent*)
- Fitur Tampilkan Daftar Pengguna

Diagram *communication* untuk fitur ini akan menjelaskan proses yang terjadi pada saat pengguna menggunakan fitur untuk menampilkan daftar pengguna dari aplikasi klien (*user agent*). Berikut adalah gambar diagramnya :

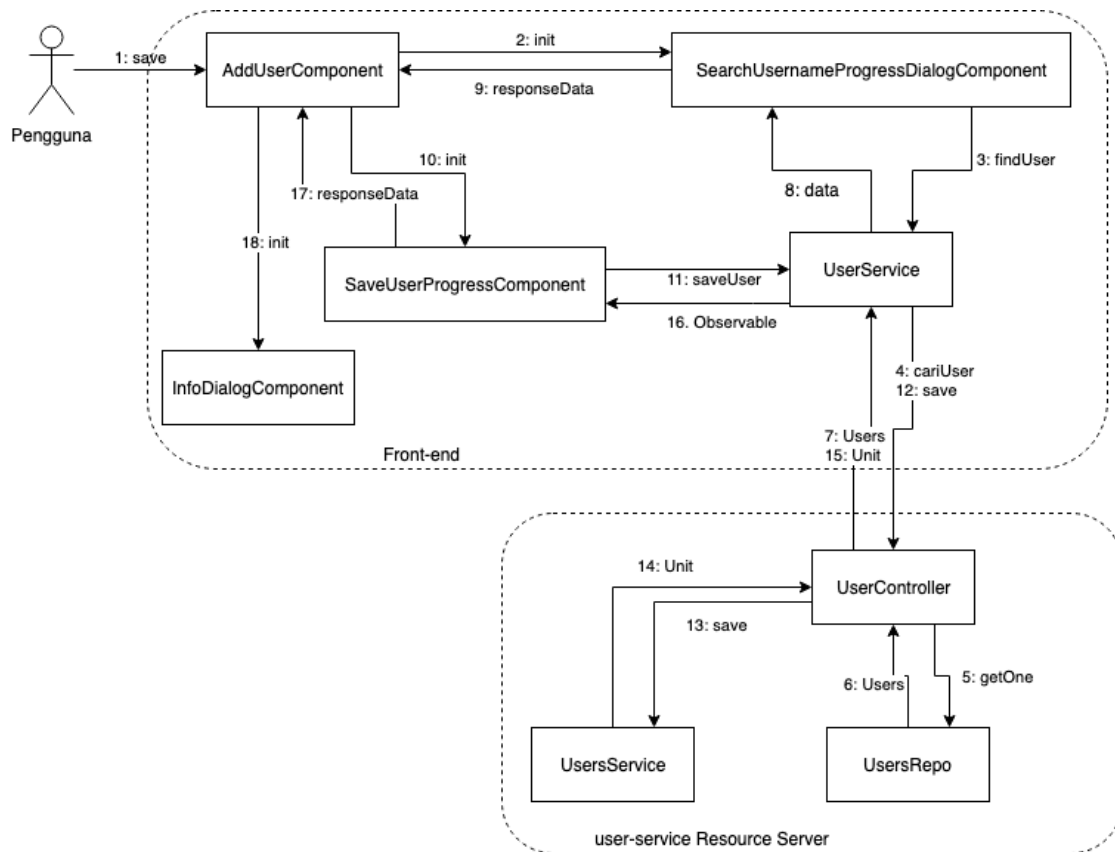


Gambar 29: Diagram *Communication* Untuk Proses Menampilkan Data Pengguna

Prosesnya akan berlangsung seperti skenario di bawah ini :

1. Pengguna (*resource owner*) akan melakukan akses ke menu daftar pengguna.
 2. Dari komponen `ListUserComponent` melakukan akses ke `UserService` untuk mendapatkan daftar pengguna.
 3. Komponen `UserService` melakukan akses ke `UserController` di *Resource Server*.
 4. Kelas *UserController* akan melakukan akses ke sistem basis data melalui *UsersRepo* untuk mendapatkan daftar pengguna.
 5. Komponen `UsersRepo` akan mengembalikan daftar pengguna (*user agent*) ke `UserController`.
 6. `UserController` akan memberikan *response* ke aplikasi klien (*user agent*) dalam bentuk JSON dimana transfer datanya akan bersifat *asynchronous*.
 7. Komponen `UserService` akan mengirimkan data ke ke `ListUserComponent` untuk ditampilkan dalam sebuah tabel.
- Fitur Tambah Pengguna Baru

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna menggunakan fitur tambah pengguna dari aplikasi klien (*user agent*). Berikut adalah diagramnya :



Gambar 30: Diagram *Communication* Untuk Proses Tambah Data Pengguna

Proses komunikasi antar komponen untuk fitur ini akan terjadi seperti skenario berikut :

1. Pengguna melakukan aksi simpan data pengguna
2. Komponen `AddUserComponent` akan melakukan pemanggilan komponen lain, yaitu `SearchUsernameProgressDialogComponent`
3. Komponen `SearchUsernameProgressDialogComponent` akan melakukan permintaan data ke komponen `UserService`.
4. `UserService` akan melakukan *request* ke *Resource Server* melalui `UserController`.

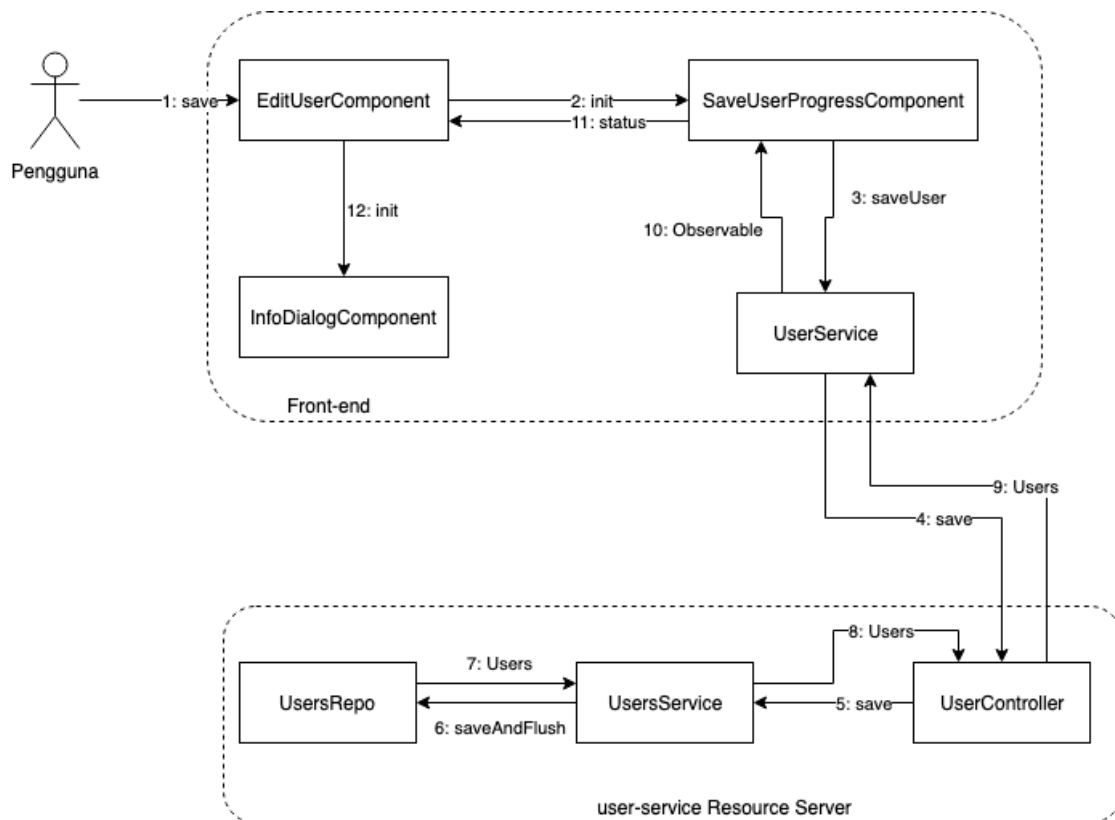
5. `UserController` melakukan akses ke basis data melalui `UserRepo`
6. `UserRepo` mengembalikan informasi detail dari pengguna yang diminta
7. `UserController` memberikan *response* ke *front-end*, dalam hal ini adalah `UserService` berupa informasi detail pengguna
8. `UserService` akan memberikan data ke `SearchUsernameProgressDialogComponent`.
9. `SearchUsernameProgressDialogComponent` akan memberikan *response* ke `AddUserComponent` bahwa data tersebut ada atau tidak ada, terdaftar atau belum terdaftar.
10. Bila data pengguna belum pernah terdaftar sebelumnya, `AddUserComponent` akan melakukan pemanggilan ke komponen `SaveUserProgressComponent`
11. `SaveUserProgressComponent` akan mengirimkan data ke `UserService` untuk dikirimkan ke *resource server*.
12. `UserService` melakukan *request* ke *resource server* melalui `UserController` dan mengirimkan data pengguna untuk disimpan dalam basis data.
13. `UserController` kemudian mengirimkan data pengguna yang akan disimpan ke basis data melalui `UserService`.
14. `UsersService` akan mengirimkan informasi ke `UserController` bahwa data telah tersimpan.
15. `UserController` akan mengirimkan *response* ke *front-end* bahwa data telah sukses tersimpan.
16. `UserService` akan mengirimkan informasi suksesnya ke komponen `SaveUserProgressComponent`

17. Dari `SaveUserProgressComponent` mengirimkan statusnya ke `AddUserComponent`.

18. Kemudian `AddUserComponent` melakukan pemanggilan ke `InfoDialogComponent` untuk memberitahu pengguna (*resource owner*) bahwa data telah tersimpan.

- Fitur Ubah Detail Pengguna

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna menggunakan fitur ubah data pengguna dari aplikasi klien (*user agent*). Berikut adalah diagramnya :



Gambar 31: Diagram *Communication* Untuk Proses Ubah Data Pengguna

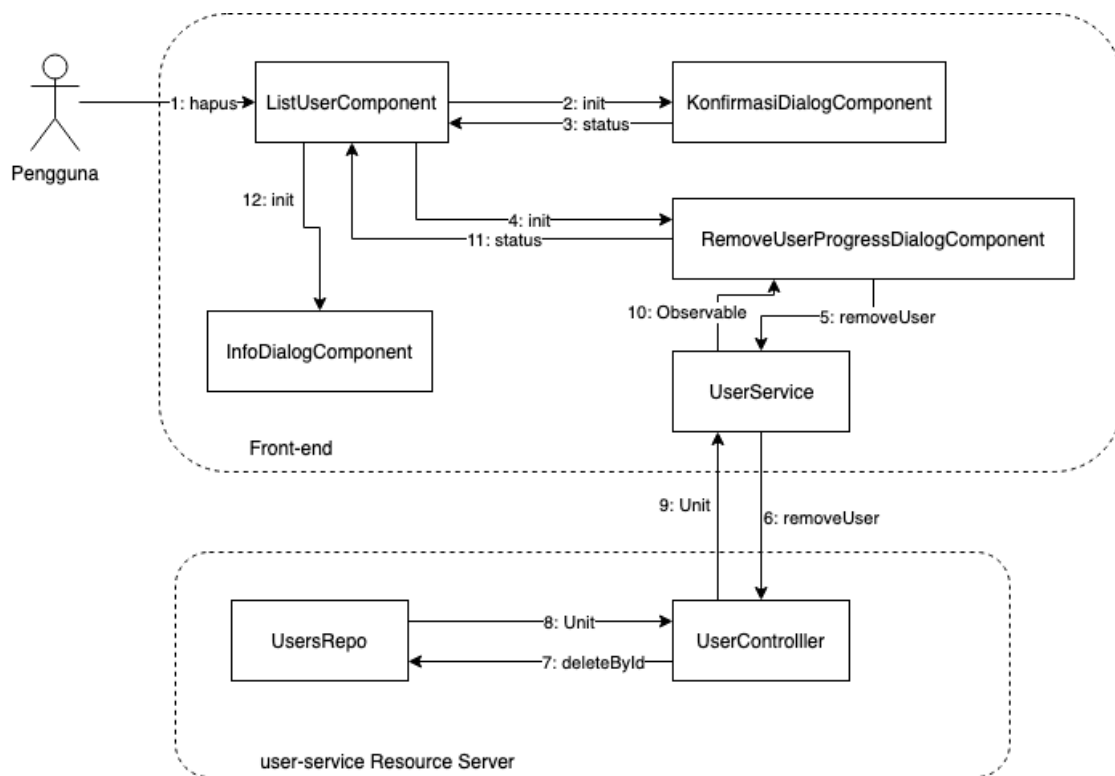
Penjelasan dari diagram tersebut adalah seperti berikut ini :

1. Pengguna melakukan simpan data melalui `EditUserComponent`
2. Komponen `EditUserComponent` membuka komponen `SaveUserProgressComponent`
3. Komponen `SaveUserProgressComponent` melakukan pemanggilan fungsi `saveUser` milik kelas `UserService`.
4. `UserService` melakukan *request* ke *resource server*, dalam hal ini melalui `UserController` dengan mengirimkan data hasil perubahan yang telah dilakukan pengguna (*resource owner*).
5. `UserController` melakukan pemanggilan fungsi `save` milik kelas `UsersService` untuk melakukan pengolahan data.
6. `UsersService` melakukan pemanggilan *method* `saveAndFlush` milik `UsersRepo` sehingga data terbaru akan tersimpan dalam sistem basis data.
7. `UsersRepo` setelah melakukan simpan data akan mengembalikan sebuah nilai dalam objek `Users` ke `UsersService`.
8. Objek `Users` akan dikembalikan oleh `UsersService` ke `UserController` untuk nantinya dikirimkan ke aplikasi klien (*user agent*)
9. `UserController` melakukan *response* ke aplikasi klien (*user agent*) dengan mengirimkan objek `Users` dalam bentuk JSON, kembali ke `UserService`.
10. `UserService` akan mengirimkan data ke `SaveUserProgressComponent` dalam bentuk `Observable`
11. `SaveUserProgressComponent` akan mengirimkan status keberhasilan proses simpan perubahan data pengguna ke `EditUserComponent`.

12. `EditUserComponent` akan memanggil `InfoDialogComponent` untuk menampilkan informasi ke pengguna (`resource owner`) bahwa data telah tersimpan atau gagal disimpan.

- Fitur Hapus Pengguna

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi saat pengguna menggunakan fitur hapus data pengguna dari aplikasi klien (*user agent*). Berikut adalah diagramnya :



Gambar 32: Diagram *Communication* Untuk Proses Hapus Data Pengguna

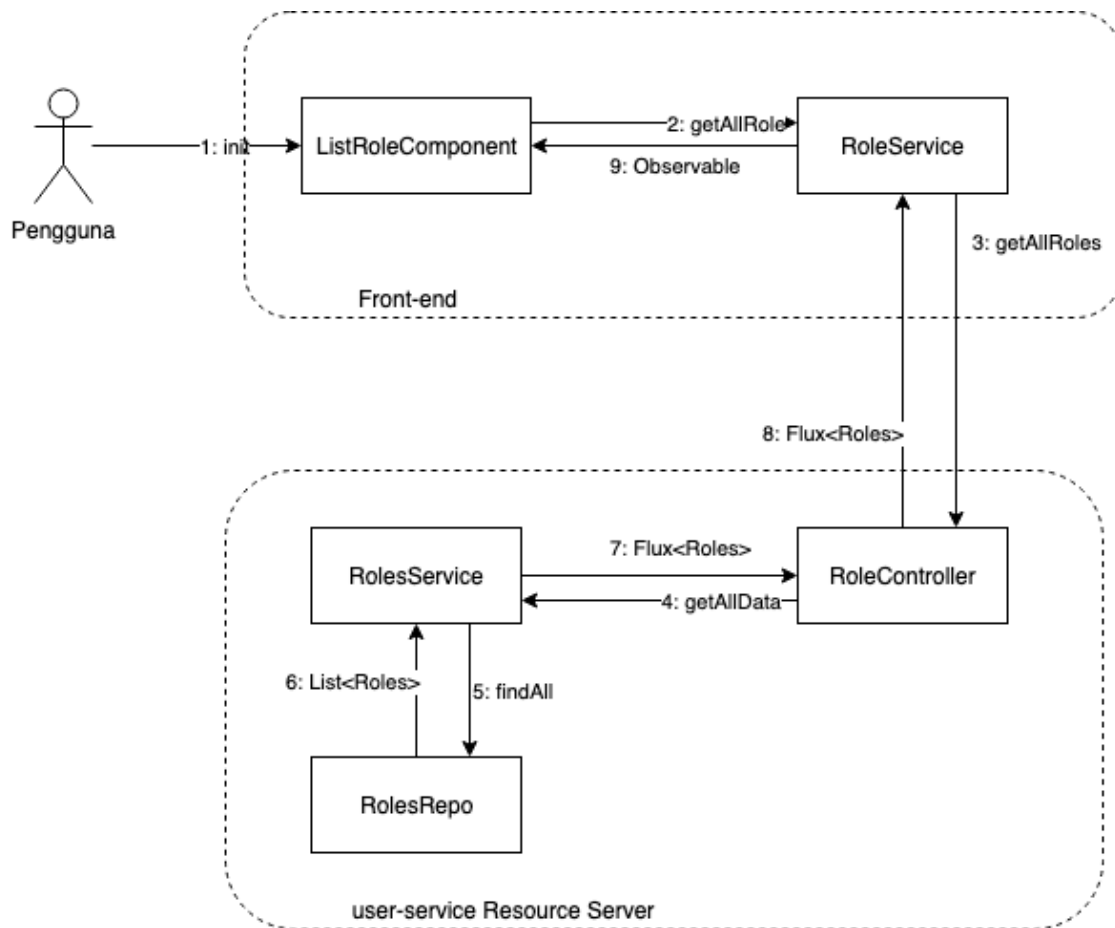
Penjelasan dari diagram tersebut adalah seperti berikut ini :

1. Pengguna (*resource owner*) melakukan hapus data dari `ListUserComponent`.

2. `ListUserComponent` memanggil komponen `KonfirmasiDialogComponent` untuk memastikan bahwa pengguna (*resource owner*) melakukannya dengan sengaja.
3. `KonfirmasiDialogComponent` mengirimkan jawaban dari pengguna (*resource owner*) atas keputusannya untuk menghapus data.
4. Komponen `ListUserComponent` memanggil `RemoveUserProgressDialogComponent`
5. `RemoveUserProgressDialogComponent` memanggil *method* `removeUser` milik `UserService`.
6. `UserService` melakukan *request* ke *resource server*, dalam hal ini melalui *method* `removeUser` milik kelas `UserController`.
7. `UserController` memanggil *method* `deleteById` milik `UsersRepo` untuk melakukan hapus data di sistem basis data
8. `UsersRepo` melakukan hapus data dan mengembalikan nilai `Unit` yang sebetulnya tidak memiliki nilai apapun.
9. `UserController` mengirimkan *response* ke *front-end* melalui `UserService`.
10. `UserService` mengembalikan hasil ke `RemoveUserProgressDialogComponent` dalam bentuk `Observable`.
11. `RemoveUserProgressDialogComponent` mengembalikan status ke `ListUserComponent` bahwa data telah terhapus.
12. `ListUserComponent` memanggil `InfoDialogComponent` untuk memberitahu pengguna (*resource owner*) bahwa data yang diinginkan telah dihapus

- Fitur Tampilkan Daftar Hak Akses

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna menggunakan fitur untuk menampilkan seluruh hak akses (*role*) yang telah terdaftar melalui aplikasi klien (*user agent*). Berikut adalah diagramnya :



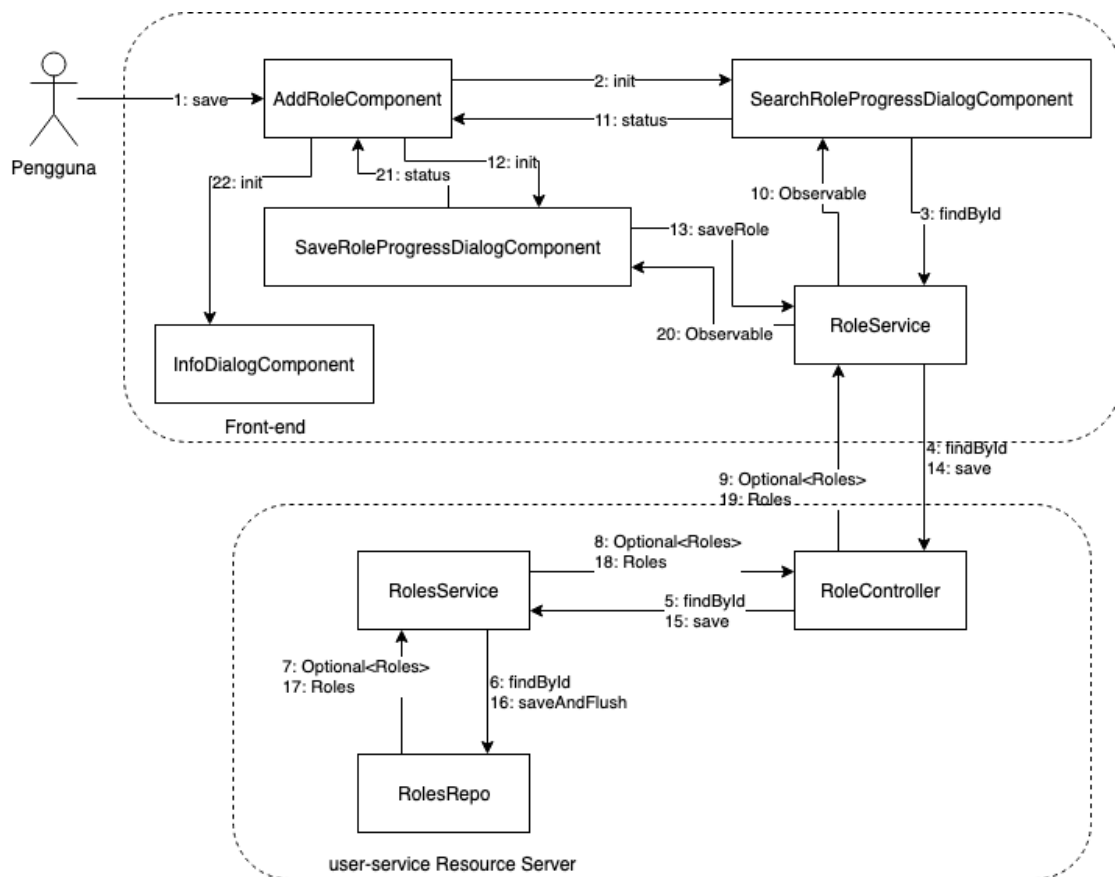
Gambar 33: Diagram *Communication* Untuk Proses Menampilkan Daftar Hak Akses

Penjelasan dari diagram tersebut adalah seperti berikut ini :

1. Pengguna (*resource owner*) melakukan akses ke fitur tampilkan daftar hak akses (*role*) melalui komponen `ListRoleComponent`.

2. `ListRoleComponent` memanggil *method* `getAllRole()` milik `RoleService`
 3. `RoleService` melakukan *request* ke *resource server* melalui *method* `getAllRoles()` milik kelas `RoleController`
 4. `RoleController` melakukan pemanggilan *method* `getAllData()` milik kelas `RolesService`
 5. `RolesService` melakukan pemanggilan *method* `findAll()` milik `RolesRepo` untuk mengambil daftar hak akses dari sistem basis data.
 6. `RolesRepo` mengembalikan daftar hak akses dalam bentuk `List<Roles>` ke `RolesService`
 7. `RolesService` mengembalikan nilai ke `RoleController` dalam bentuk `Flux<Roles>` yang menggunakan fitur *reactive*
 8. `RoleController` akan memberikan *response* ke *front-end* berupa `Flux<Roles>` dalam bentuk JSON, untuk kasus ini, data akan dikirimkan ke `RoleService`.
 9. `RoleService` akan mengirimkan data ke `ListRoleComponent` dalam bentuk `Observable` yang kemudian datanya ditampilkan dalam tabel.
- Fitur Tambah Hak Akses Baru

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna menggunakan fitur tambah hak akses baru (*role*) yang ada pada aplikasi klien (*user agent*). Berikut adalah diagramnya :



Gambar 34: Diagram *Communication* Untuk Proses Tambah Data Hak Akses

Penjelasan untuk diagram tersebut adalah seperti berikut ini :

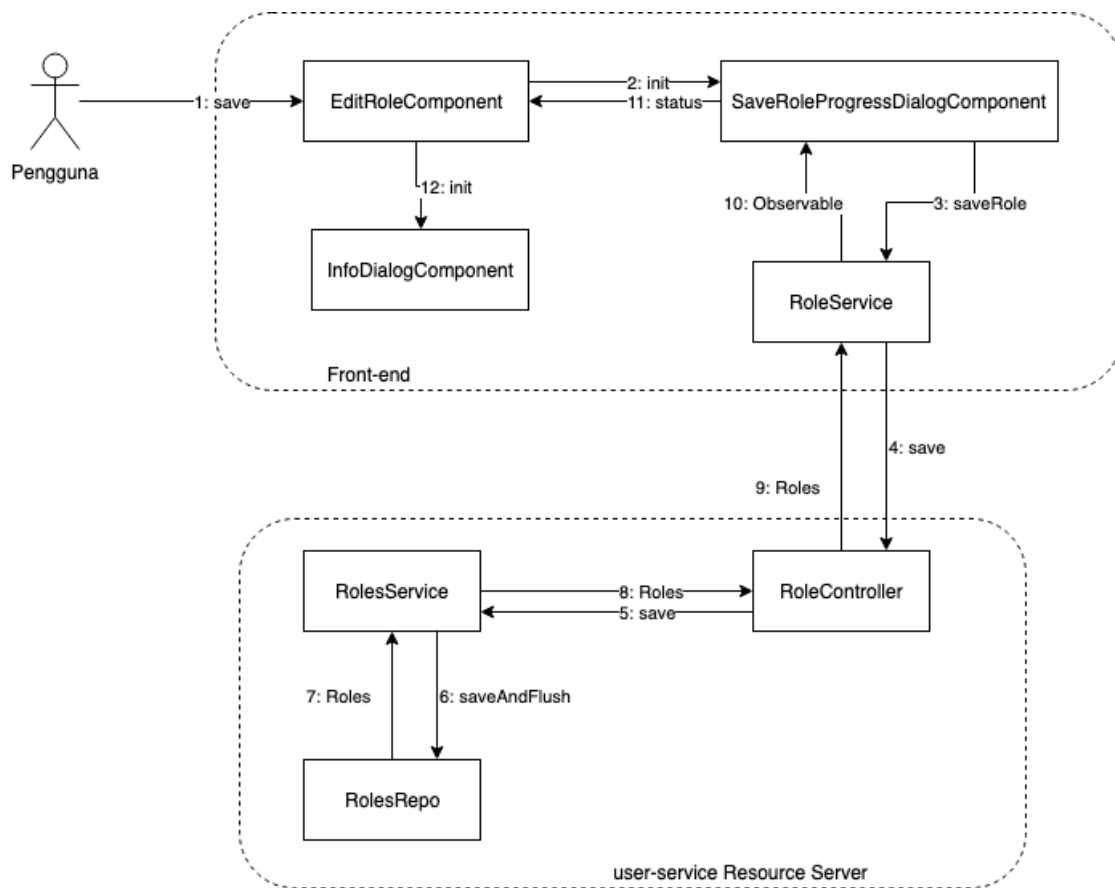
1. Pengguna (*resource owner*) melakukan aksi simpan data di komponen `AddRoleComponent`
2. `AddRoleComponent` mengaktifkan `SearchRoleProgressDialogComponent` untuk memeriksa apakah hak akses sudah pernah terdaftar dengan nama tersebut
3. `SearchRoleProgressDialogComponent` melakukan akses ke *method* `findById()` milik `RoleService`

4. RoleService melakukan *request* ke *resource server*, dalam hal ini melalui *method* `findById` milik kelas RoleController
5. RoleController melakukan pemanggilan *method* `findById()` milik kelas RolesService
6. RolesService melakukan pemanggilan *method* `findById()` milik RolesRepo untuk melakukan simpan data ke sistem basis data.
7. RolesRepo setelah melakukan simpan data ke sistem basis data, akan mengembalikan respon berbentuk `Optional<Roles>` ke RolesService
8. RolesService meneruskan pesan ke RoleController dalam bentuk yang sama, yaitu `Optional<Roles>`
9. RoleController memberikan *response* ke *front-end*, dalam hal ini melalui RoleService berupa data `Optional<Roles>` dalam bentuk JSON.
10. RoleService mengembalikan nilai ke SearchRoleProgressDialogComponent dalam bentuk `Observable`
11. SearchRoleProgressDialogComponent mengembalikan status datanya apakah sudah terdaftar atau belum ke AddRoleComponent
12. AddRoleComponent melakukan pemanggilan ke SaveRoleProgressDialogComponent untuk melakukan simpan data.
13. SaveRoleProgressDialogComponent melakukan pemanggilan *method* `saveRole()` milik RoleService untuk melakukan simpan data hak akses baru
14. RoleService melakukan *request* ke *resource server* melalui *method* `save` milik kelas RoleController
15. RoleController melakukan akses terhadap *method* `save` milik kelas RolesService

16. `RolesService` melakukan simpan data dengan memanggil *method* `saveAndFlush()` milik `RolesRepo`
17. Setelah data tersimpan dalam sistem basis data, `RolesRepo` akan mengembalikan objek dari `Roles` ke `RolesService`
18. `RolesService` meneruskan nilai yang berupa objek dari `Roles` ke `RoleController`
19. `RoleController` memberikan *response* berupa objek `Roles` dalam bentuk JSON ke `RoleService` di *front-end*
20. `RoleService` akan mengembalikan nilai dalam bentuk `Observable` ke `SaveRoleProgressDialogComponent`
21. `SaveRoleProgressDialogComponent` memberikan status ke `AddRoleComponent` apakah datanya tersimpan atau tidak
22. `AddRoleComponent` mengaktifkan `InfoDialogComponent` sebagai informasi bagi pengguna (*resource owner*) apakah datanya telah berhasil tersimpan atau gagal.

- Fitur Ubah Hak Akses

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) menggunakan fitur ubah data hak akses (*role*) yang ada pada aplikasi klien (*user agent*). Berikut adalah diagramnya :



Gambar 35: Diagram *Communication* Untuk Proses Ubah Data Hak Akses

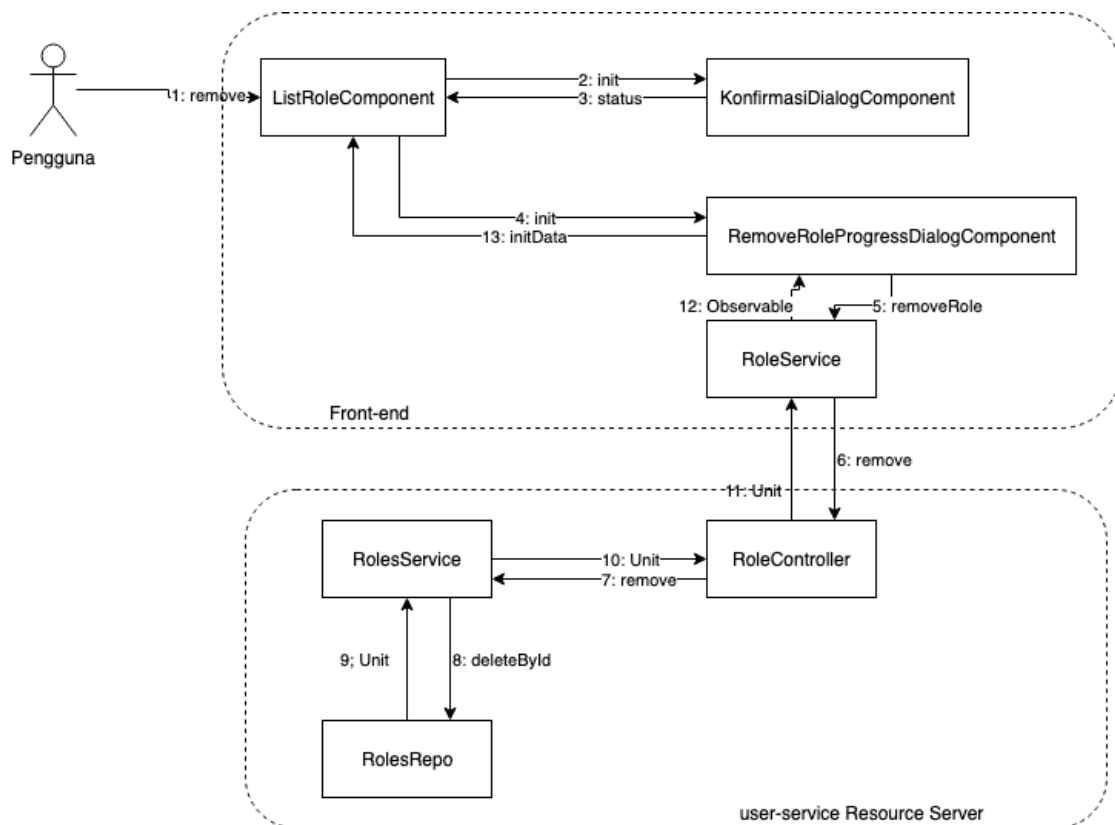
Penjelasan untuk diagram tersebut adalah seperti berikut ini :

1. Pengguna (*resource owner*) melakukan simpan data perubahan melalui `EditRoleComponent`
2. `EditRoleComponent` memanggil komponen `SaveRoleProgressDialogComponent` untuk menunjukkan ke pengguna (*resource owner*) bahwa proses simpan data sedang berjalan.
3. `SaveRoleProgressDialogComponent` melakukan akses ke *method* `saveRole` milik `RoleService` dengan membawa data hasil perubahan

4. `RoleService` melakukan *request* ke *resource server*, dalam hal ini melalui *method save* milik kelas `RoleController`
5. `RoleController` memanggil *method save* milik kelas `RolesService`
6. `RolesService` akan memanggil *method saveAndFlush* milik `RolesRepo` untuk kemudian disimpan dalam sistem basis data
7. `RolesRepo` setelah melakukan simpan data ke sistem basis data, akan mengembalikan sebuah objek `Roles` ke `RolesService`
8. `RolesService` akan meneruskan objek `Roles` ke `RoleController`
9. `RoleController` melakukan *response* ke *front-end* melalui `RoleService` yang sebelumnya melakukan *request*.
10. `RoleService` mengirimkan data dari *response resource server* ke `SaveRoleProgressDialogComponent` dalam bentuk `Observable`
11. `SaveRoleProgressDialogComponent` mengirimkan status ke `EditRoleComponent` sebagai tanda apakah simpan data berhasil dilakukan atau gagal.
12. `EditRoleComponent` melakukan pemanggilan `InfoDialogComponent` untuk memberitahu pengguna (*resource server*) bahwa data berhasil tersimpan atau gagal.

- **Fitur Hapus Hak Akses**

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) menggunakan fitur hapus data hak akses (*role*) yang ada pada aplikasi klien (*user agent*). Berikut adalah diagramnya :



Gambar 36: Diagram *Communication* Untuk Proses Hapus Data Hak Akses

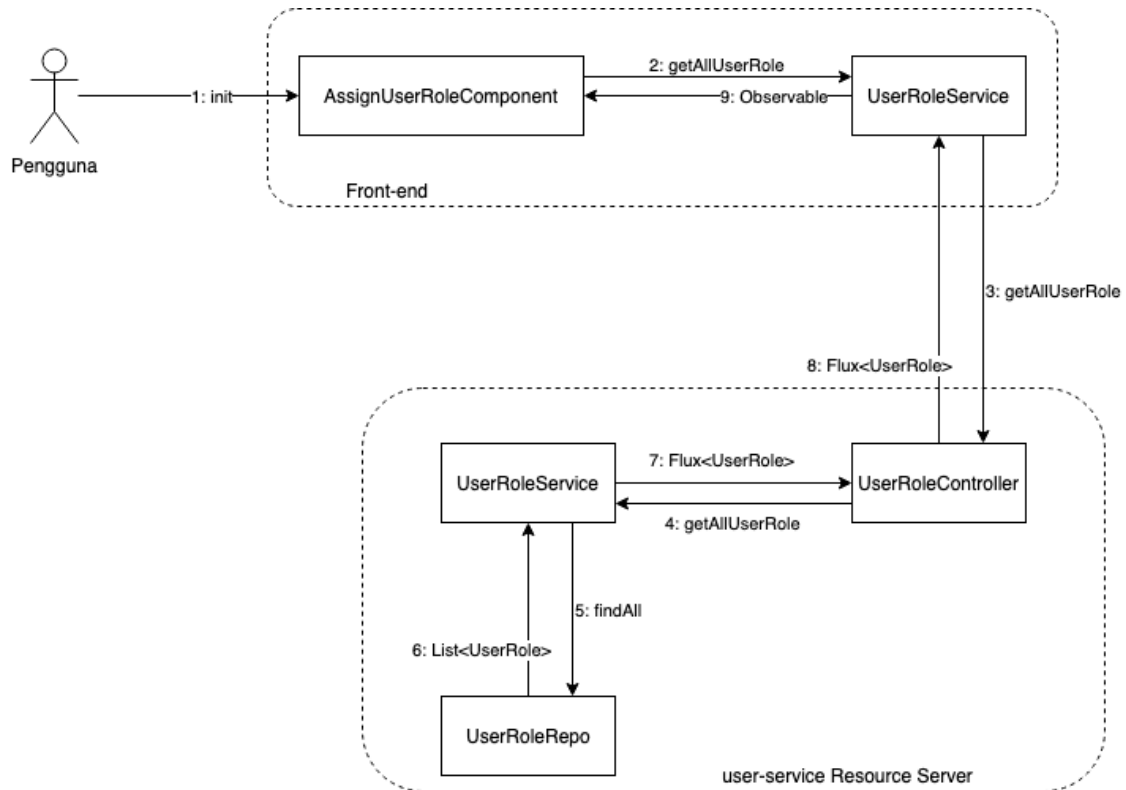
Penjelasan diagram tersebut adalah seperti berikut ini :

1. Pengguna (*resource owner*) melakukan hapus data hak akses melalui `ListRoleComponent`
2. `ListRoleComponent` melakukan pemanggilan `KonfirmasiDialogComponent` untuk memastikan bahwa pengguna (*resource owner*) benar-benar melakukan permintaan hapus data hak akses
3. `KonfirmasiDialogComponent` mengembalikan status pilihan pengguna (*resource owner*) ke `ListRoleComponent` bahwa data benar akan dihapus atau tidak

4. `ListRoleComponent` akan mengaktifkan `RemoveRoleProgressDialogComponent` untuk melakukan proses hapus data hak akses
 5. `RemoveRoleProgressDialogComponent` melakukan akses ke fungsi `removeRole` milik `RoleService`
 6. `RoleService` melakukan *request* ke *resource server* melalui *method* `remove` milik kelas `RoleController`
 7. `RoleController` melakukan akses ke *method* `remove` milik kelas `RolesService`
 8. `RolesService` melakukan akses ke *method* `deleteById` milik `RolesRepo` untuk menghapus data hak akses pada sistem data
 9. Setelah `RolesRepo` menghapus data hak akses pada sistem data, `RolesRepo` mengembalikan nilai kosong dalam bentuk `Unit` ke `RolesService`
 10. `RolesService` akan meneruskan hasil nilai `Unit` ke `RoleController`
 11. `RoleController` akan memberikan *response* berupa `Unit` ke *front-end* melalui kelas `RoleService` yang sebelumnya melakukan *request*.
 12. `RoleService` akan mengembalikan nilai ke `RemoveRoleProgressDialogComponent` dalam bentuk `Observable`
 13. `RemoveRoleProgressDialogComponent` akan mengembalikan nilai ke `ListRoleComponent` untuk melakukan *refresh* tabel daftar hak akses
- Fitur Tampilkan Daftar Pengguna Dengan Hak Akses

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) melakukan akses terhadap daftar hak akses

yang diberikan ke pengguna yang terdaftar. Diagramnya akan terlihat seperti berikut :



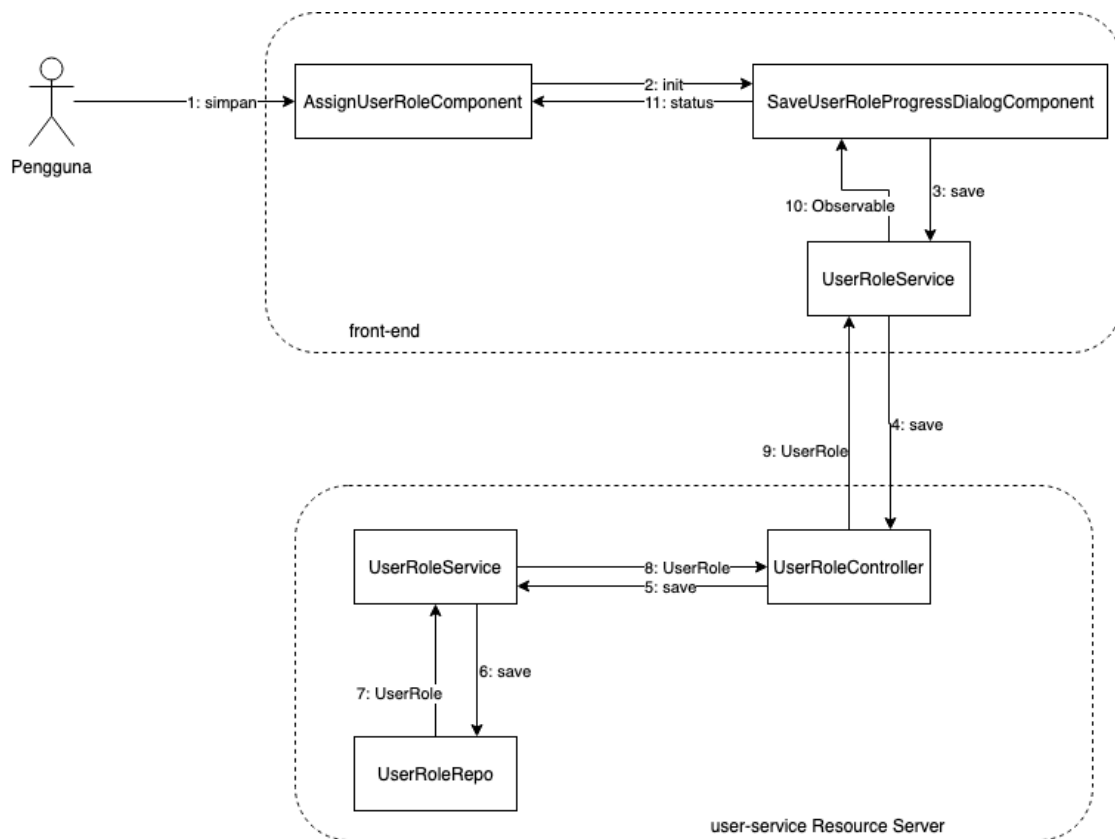
Gambar 37: Diagram *Communication* Untuk Melihat Daftar Hak Akses Yang Diberikan Ke Pengguna

Penjelasan dari diagram tersebut adalah seperti berikut ini :

1. Pengguna melakukan akses ke daftar hak akses yang diberikan ke pengguna melalui `AssignUserRoleComponent`
2. `AssignUserRoleComponent` melakukan akses ke fungsi `getAllUserRole` ke `UserRoleService`
3. `UserRoleService` melakukan *request* ke *resource server* melalui *method* `getAllUserRole` milik kelas `UserRoleController`

4. `UserRoleController` melakukan akses ke *method* `getAllUserRole` milik `UserRoleService`
 5. `UserRoleService` melakukan akses ke *method* `findAll` milik `UserRoleRepo` untuk mengambil seluruh data hak akses yang diberikan ke pengguna yang tercatat dalam sistem basis data
 6. `UserRoleRepo` mengembalikan nilai ke `UserRoleService` dalam bentuk `List<UserRole>`
 7. `UserRoleService` mengembalikan nilai ke `UserRoleController` dalam bentuk `Flux<UserRole>`
 8. `UserRoleController` memberikan *response* ke bagian *front-end* melalui `UserRoleService` yang sebelumnya melakukan *request* dalam bentuk `Flux<UserRole>`
 9. `UserRoleService` akan memberikan nilai ke `AssignUserRoleComponent` dalam bentuk `Observable` untuk kemudian ditampilkan dalam tabel
- Fitur Tambahkan Hak Akses ke Pengguna

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) melakukan penambahan data hak akses untuk pengguna yang terdaftar. Diagramnya akan terlihat seperti berikut :



Gambar 38: Diagram *Communication* Untuk Menambahkan Data Hak Akses Bagi Pengguna

Penjelasan diagram di atas adalah seperti berikut :

1. Pengguna (*resource owner*) melakukan simpan data dari hak akses yang diberikan ke pengguna melalui `AssignUserRoleComponent`.
2. `AssignUserRoleComponent` melakukan pemanggilan ke `SaveUserRoleProgressDialogComponent`
3. `SaveUserRoleProgressDialogComponent` melakukan akses ke *method* `save` milik `UserRoleService`
4. `UserRoleService` melakukan *request* ke *resource server* melalui *method*

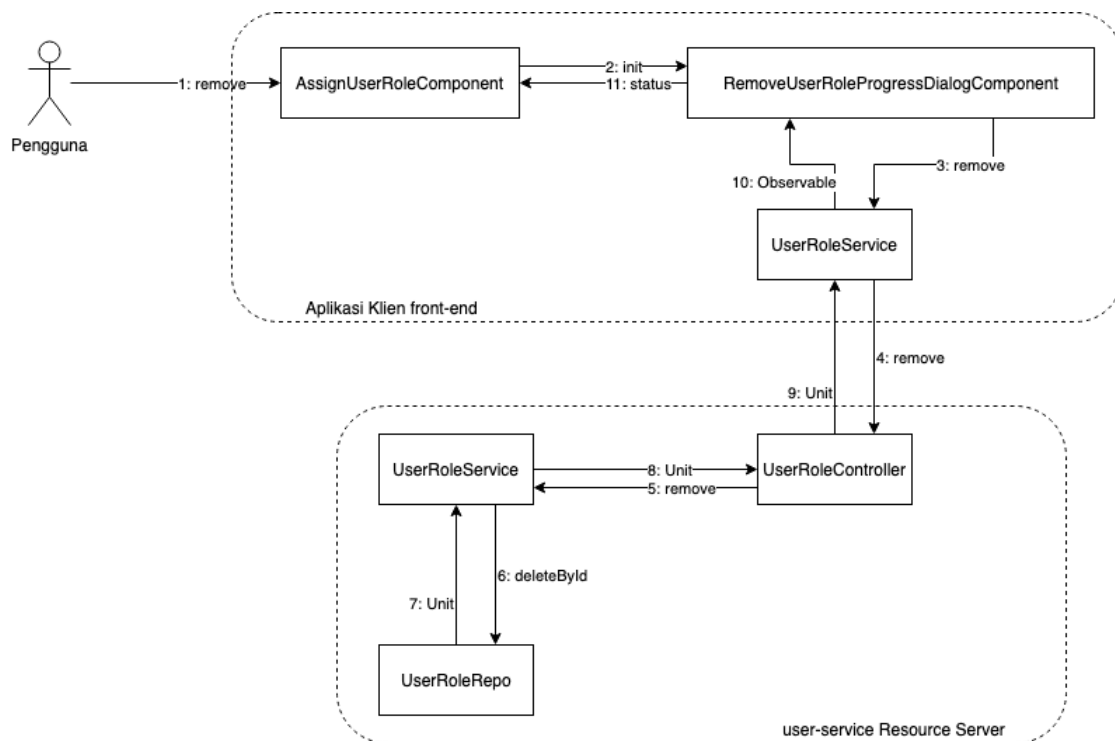
`save` milik `UserRoleController` dengan membawa informasi data hak akses yang diberikan ke pengguna (*resource owner*).

5. `UserRoleController` melakukan akses ke *method* `save` milik `UserRoleService`
6. `UserRoleService` melakukan akses ke *method* `save` milik `UserRoleRepo` untuk menyimpan data ke sistem basis data
7. `UserRoleRepo` setelah melakukan simpan data ke sistem basis data akan mengembalikan nilai berupa objek dari `UserRole` ke `UserRoleService`
8. `UserRoleService` meneruskan nilai dari objek `UserRole` ke `UserRoleController`
9. `UserRoleController` akan memberikan *response* ke *front-end* berupa objek dari kelas `UserRole` atas *request* yang sebelumnya dilakukan oleh `UserRoleService`
10. `UserRoleService` akan mengembalikan nilai ke `SaveUserRoleProgressDialogComponent` dalam bentuk `Observable`
11. `SaveUserRoleProgressDialogComponent` akan mengembalikan sebuah status ke `AssignUserRoleComponent` untuk kemudian `AssignUserRoleComponent` melakukan *refresh* atau pembaruan pada tabel daftar hak akses yang telah diberikan ke pengguna

- Fitur Hapus Hak Akses dari Pengguna

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) melakukan penghapusan data hak akses yang telah diberikan ke pengguna yang terdaftar. Diagramnya akan terlihat seperti berikut :

Penjelasan dari diagram tersebut adalah seperti berikut ini :



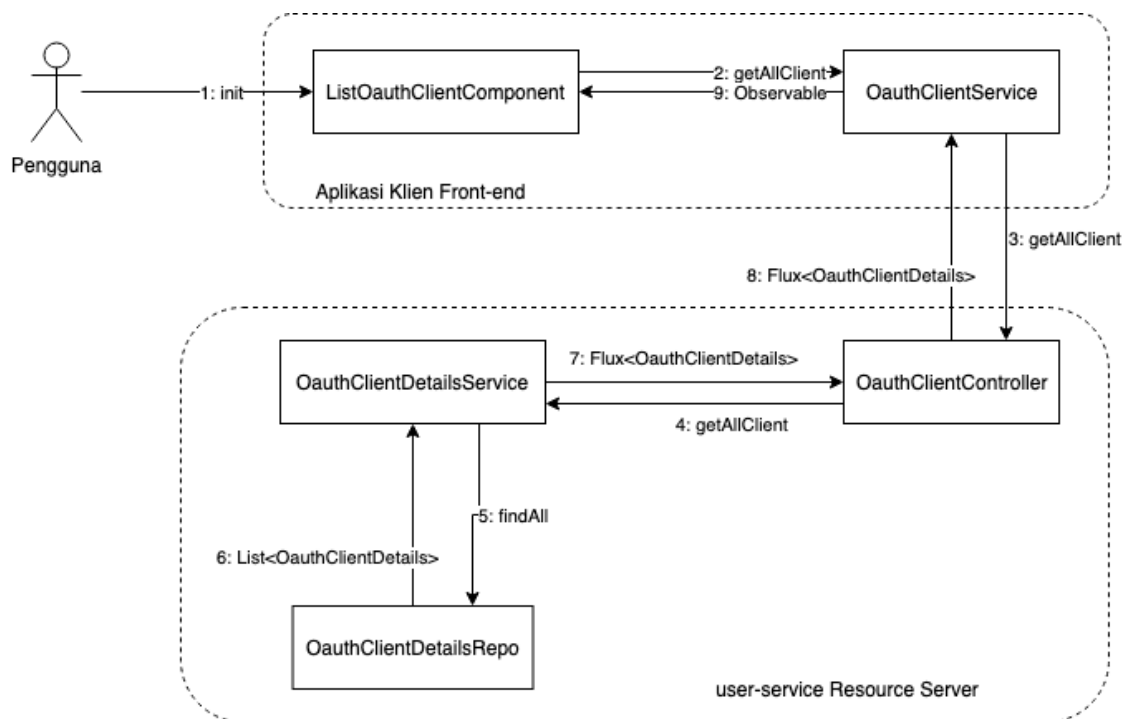
Gambar 39: Diagram *Communication* Untuk Menghapus Data Hak Akses Yang Telah Diberikan Ke Pengguna

1. Pengguna (*resource owner*) melakukan akses hapus data hak akses untuk pengguna yang telah terdaftar melalui `AssignUserRoleComponent`
2. `AssignUserRoleComponent` melakukan pemanggilan ke `RemoveUserRoleProgressDialogComponent` untuk melakukan proses penghapusan hak akses terhadap pengguna yang terpilih
3. `RemoveUserRoleProgressDialogComponent` melakukan akses ke fungsi `remove` milik `UserRoleService`
4. `UserRoleService` melakukan *request* ke *resource server* melalui *method* `remove` milik kelas `UserRoleController`
5. `UserRoleController` melakukan akses ke *method* `remove` milik kelas `UserRoleService`
6. `UserRoleService` melakukan akses ke *method* `deleteById` milik `UserRoleRepo` untuk menghapus data hak akses milik pengguna yang terdaftar di sistem basis data
7. Setelah melakukan hapus data di sistem basis data, `UserRoleRepo` mengembalikan respon berupa kelas `Unit` yang sebetulnya tidak ada nilainya.
8. `UserRoleService` meneruskan respon berupa kelas `Unit` ke `UserRoleController`
9. `UserRoleController` mengirimkan *response* ke aplikasi klien berupa kelas `Unit` dalam bentuk JSON, dalam hal ini ke kelas `UserRoleService`
10. `UserRoleService` meneruskan *response* ke `RemoveUserRoleProgressDialogComponent` dalam bentuk `Observable`

11. `RemoveUserRoleProgressDialogComponent` memberikan status ke `AssignUserRoleComponent` untuk melakukan *refresh* tabel daftar hak akses untuk pengguna.

- Fitur Tampilkan Daftar Aplikasi Klien

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) menggunakan fitur untuk melihat daftar aplikasi klien yang terdaftar. Diagramnya akan terlihat seperti berikut :



Gambar 40: Diagram *Communication* Untuk Menampilkan Daftar Aplikasi Klien Yang Terdaftar

Penjelasan untuk diagram tersebut adalah seperti berikut :

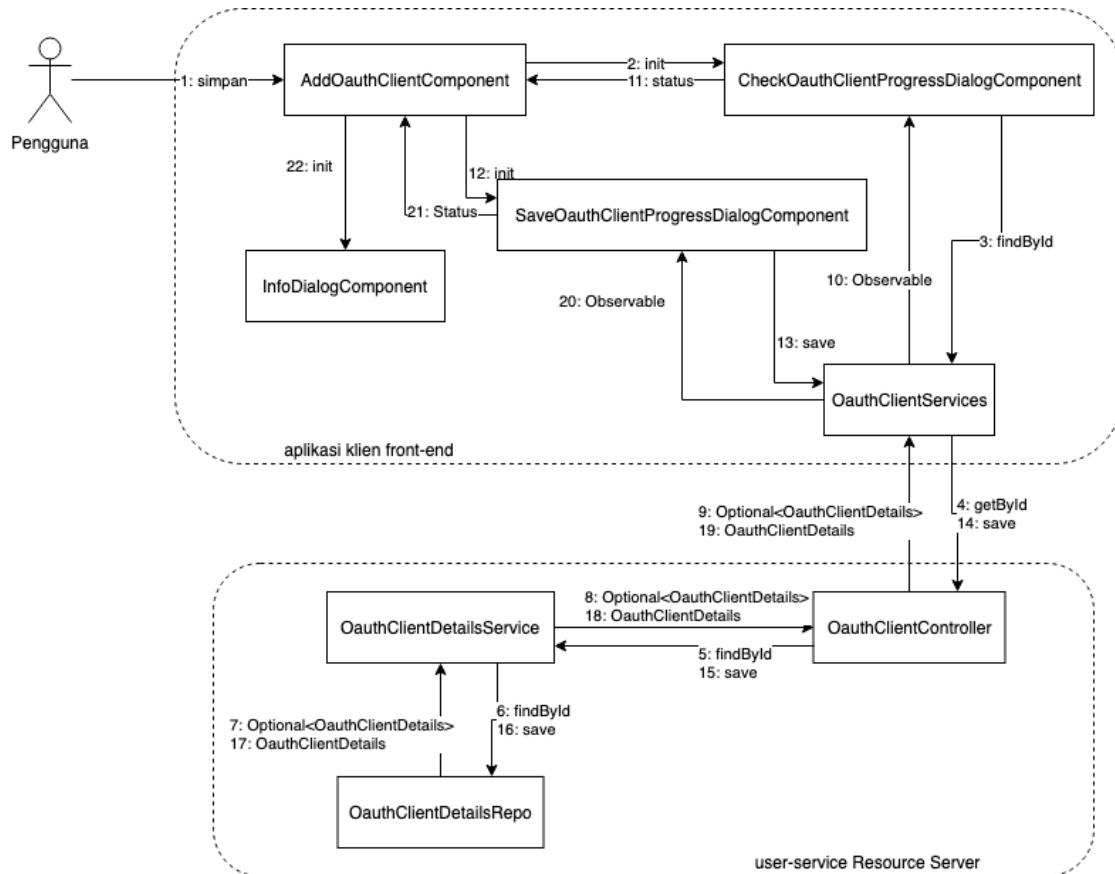
1. Pengguna (*resource owner*) melakukan akses ke fitur tampilkan daftar aplikasi klien melalui `ListOAuthClientComponent`

2. `ListOAuthClientComponent` melakukan akses ke fungsi `getAllClient` milik kelas `OAuthClientService`
3. `OAuthClientService` melakukan *request* ke *resource owner* melalui *method* `getAllClient` milik kelas `OAuthClientController`
4. `OAuthClientController` melakukan akses ke *method* `getAllClient` milik `OAuthClientDetailsService`
5. `OAuthClientDetailsService` melakukan akses ke *method* `findAll` milik `OAuthClientDetailsRepo` untuk melakukan akses ke sistem basis data.
6. Setelah `OAuthClientDetailsRepo` mengambil data dari sistem basis data, `OAuthClientDetailsRepo` akan mengembalikan nilai berupa `List<OAuthClientDetails>` yang berisi daftar aplikasi klien beserta detailnya.
7. `OAuthClientDetailsService` akan meneruskan data ke `OAuthClientController` dalam bentuk objek dari `Flux<OAuthClientDetails>`
8. `OAuthClientController` akan memberikan *response* ke aplikasi klien berupa objek dari `Flux<OAuthClientDetails>`, tentunya melalui `OAuthClientService` karena sebelumnya melakukan *request*
9. `OAuthClientService` mengembalikan nilai ke *ListOAuthClientComponent* untuk ditampilkan dalam tabel nantinya dalam bentuk `Observable`

- Fitur Tambah Aplikasi Klien

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) menggunakan fitur untuk menambahkan

aplikasi klien ke dalam sistem otentikasi. Diagramnya akan terlihat seperti berikut :



Gambar 41: Diagram *Communication* Untuk Menambahkan Data Aplikasi Klien

Penjelasan dari diagram tersebut adalah sebagai berikut :

1. Pengguna (*resource owner*) melakukan aksi simpan data aplikasi klien yang baru melalui **AddOauthClientComponent**
2. **AddOauthClientComponent** melakukan pemanggilan ke **CheckOauthClientProgressDialogComponent** untuk melakukan pemeriksaan terhadap data yang akan disimpan, apakah datanya sudah ada dalam sistem basis data atau belum

3. `CheckOAuthClientProgressDialogComponent` melakukan akses ke fungsi `findById` milik `OAuthClientServices`
4. `OAuthClientServices` melakukan *request* ke *resource server* melalui *method* `getById` milik kelas `OAuthClientController`
5. `OAuthClientController` melakukan akses ke *method* `findById` milik `OAuthClientDetailsService`
6. `OAuthClientDetailsService` melakukan akses ke *method* `findById` milik `OAuthClientDetailsRepo` untuk memeriksa apakah ada aplikasi klien dengan *id* yang diminta, bila datanya ada, itu artinya aplikasi klien tersebut sebetulnya sudah terdaftar.
7. Hasil dari akses ke sistem basis data, `OAuthClientDetailsRepo` mengembalikan nilai ke `OAuthClientDetailsService` dalam bentuk `Optional<OAuthClientDetails>`
8. `OAuthClientDetailsService` meneruskan objek `Optional<OAuthClientDetails>` ke `OAuthClientController`
9. `OAuthClientController` memberikan *response* ke aplikasi klien, dalam hal ini melalui `OAuthClientServices` karena sebelumnya melakukan *request* tersebut berupa objek `Optional<OAuthClientDetails>` dalam bentuk JSON
10. `OAuthClientServices` meneruskan hasil *response* yang dikirimkan oleh *resource server* ke `CheckOAuthClientProgressDialogComponent` dalam bentuk `Observable`
11. `CheckOAuthClientProgressDialogComponent` mengirimkan status ke `AddOAuthClientComponent` yang memberikan informasi apakah data yang akan disimpan sudah pernah terrekam atau belum dalam basis data

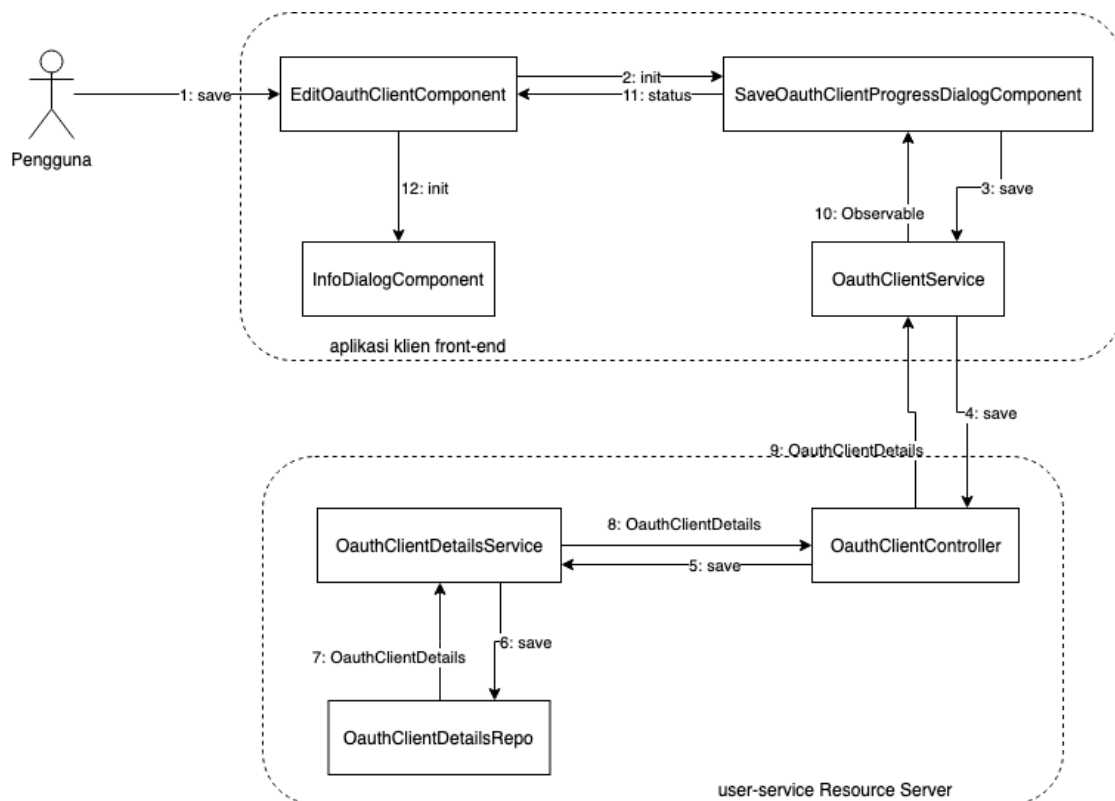
12. `AddOAuthClientComponent` melakukan pemanggilan ke `SaveOAuthClientProgressDialogComponent`
13. `SaveOAuthClientProgressDialogComponent` melakukan akses ke fungsi `save` milik `OAuthClientServices`
14. `OAuthClientServices` melakukan *request* ke *resource server* melalui *method* `save` milik kelas `OAuthClientController`
15. `OAuthClientController` melakukan akses ke `OAuthClientDetailsService` melalui *method* `save`
16. `OAuthClientDetailsService` melakukan akses ke *method* `save` milik `OAuthClientDetailsRepo` untuk melakukan simpan data ke sistem basis data.
17. Setelah melalui proses simpan data ke sistem basis data, `OAuthClientDetailsRepo` mengembalikan nilai ke `OAuthClientDetailsService` dalam bentuk objek `Optional<OAuthClientDetails>`
18. `OAuthClientDetailsService` meneruskan objek `Optional<OAuthClientDetails>` ke `OAuthClientController`
19. `OAuthClientController` memberikan *response* berupa `Optional<OAuthClientDetails>` dalam bentuk JSON ke *front-end*, dalam hal ini ke `OAuthClientServices` yang sebelumnya melakukan *request* ke *resource server*
20. `OAuthClientServices` meneruskan hasil *response* dari *resource server* ke `SaveOAuthClientProgressDialogComponent` dalam bentuk `Observable`
21. `SaveOAuthClientProgressDialogComponent` mengembalikan sebuah status ke `AddOAuthClientComponent` sebagai tanda apakah datanya

berhasil disimpan ke dalam basis data atau tidak

22. `AddOauthClientComponent` memanggil `InfoDialogComponent` untuk memberitahu pengguna (*resource owner*) bahwa data berhasil disimpan atau gagal disimpan.

- Fitur Ubah Detail Aplikasi Klien

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) menggunakan fitur untuk mengubah detail dari data aplikasi klien yang telah terdaftar dalam sistem basis data. Diagramnya akan terlihat seperti berikut :



Gambar 42: Diagram *Communication* Untuk Mengubah Detail Data Aplikasi Klien

Penjelasan dari diagram tersebut adalah seperti berikut :

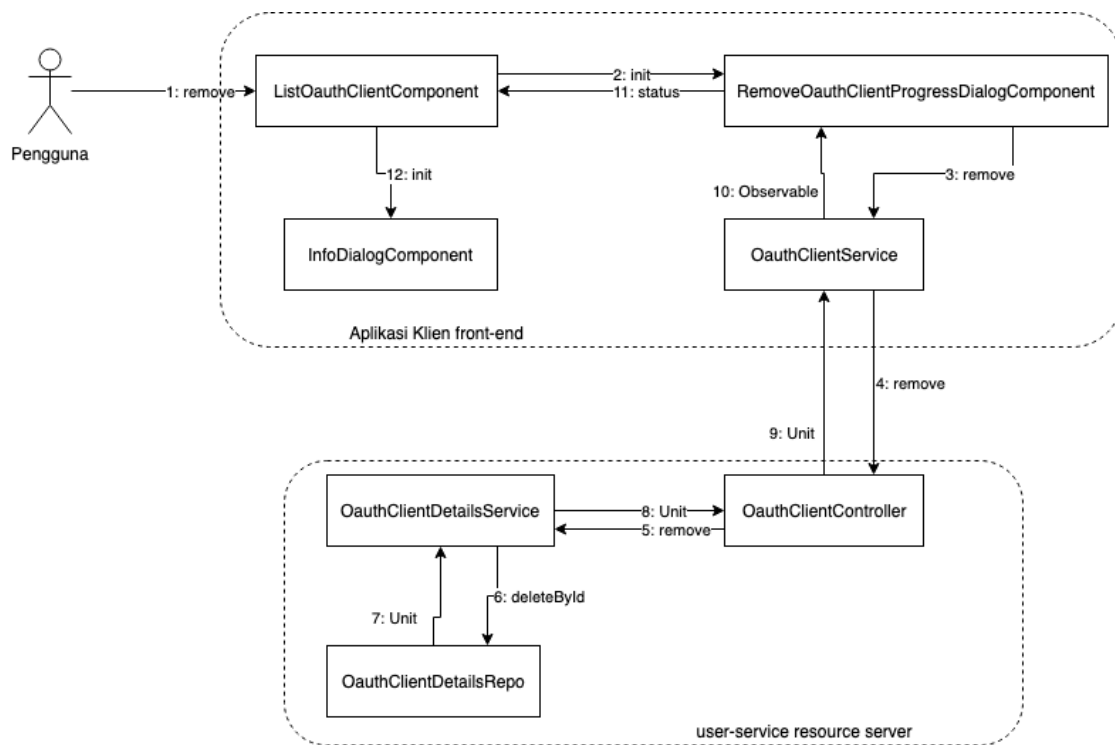
1. Pengguna melakukan proses simpan terhadap perubahan data aplikasi klien dari `EditOAuthClientComponent`
2. `EditOAuthClientComponent` memanggil `SaveOAuthClientProgressDialogComponent` untuk memulai proses simpan perubahan data ke sistem basis data
3. `SaveOAuthClientProgressDialogComponent` melakukan akses ke fungsi `save` milik `OAuthClientService`
4. `OAuthClientService` melakukan *request* simpan data ke *resource server* melalui *method* `save` milik `OAuthClientController`
5. `OAuthClientController` melakukan akses ke *method* `save` milik `OAuthClientDetailsService`
6. `OAuthClientDetailsService` melakukan akses ke *method* `save` milik `OAuthClientDetailsRepo` untuk melakukan simpan data ke sistem basis data
7. Setelah melakukan simpan data ke sistem basis data, `OAuthClientDetailsRepo` mengirimkan objek `OAuthClientDetails` ke `OAuthClientDetailsService`
8. `OAuthClientDetailsService` meneruskan nilai berupa objek `OAuthClientDetails` ke `OAuthClientController`
9. `OAuthClientController` mengirimkan *response* ke *front-end* dalam hal ini adalah `OAuthClientService` yang sebelumnya mengirimkan *request* untuk simpan perubahan data aplikasi klien
10. `OAuthClientService` meneruskan hasil *response* dari *resource server* ke `SaveOAuthClientProgressDialogComponent` dalam bentuk

Observable

11. `SaveOAuthClientProgressDialogComponent` akan mengembalikan status ke `EditOAuthClientComponent` apakah hasil perubahan data berhasil disimpan ke sistem basis data atau tidak.
12. `EditOAuthClientComponent` memanggil `InfoDialogComponent` untuk memberitahu pengguna (*resource owner*) bahwa proses simpan data perubahan berhasil atau tidak.

- Fitur Hapus Aplikasi Klien

Diagram *communication* untuk fitur ini menjelaskan proses yang terjadi pada saat pengguna (*resource owner*) menggunakan fitur untuk menghapus detail data dari aplikasi klien yang telah terdaftar dalam sistem basis data. Diagramnya akan terlihat seperti berikut :



Gambar 43: Diagram *Communication* Untuk Menghapus Detail Data Aplikasi Klien

Penjelasan dari diagram tersebut adalah seperti berikut ini :

1. Pengguna melakukan aksi hapus data aplikasi klien melalui `ListOAuthClientComponent`
2. `ListOAuthClientComponent` memanggil `RemoveOAuthClientProgressDialogComponent` untuk memulai proses hapus data aplikasi klien dan pengguna (*resource owner*) mengetahui prosesnya masih berjalan
3. `RemoveOAuthClientProgressDialogComponent` melakukan akses ke fungsi `remove` milik `OAuthClientService`

4. `OAuthClientService` melakukan *request* ke *resource server*, dalam hal ini melalui *method remove* milik kelas `OAuthClientController`
5. `OAuthClientController` melakukan akses ke *method remove* milik `OAuthClientDetailsService`
6. `OAuthClientDetailsService` melakukan akses ke *method deleteById* milik `OAuthClientDetailsRepo` agar data klien yang dimaksud terhapus dari sistem basis data.
7. Setelah `OAuthClientDetailsRepo` melakukan penghapusan data aplikasi klien di sisi sistem basis data, `OAuthClientDetailsRepo` mengirimkan objek berupa `Unit` yang sebetulnya tidak ada nilainya.
8. `OAuthClientDetails Service` meneruskan objek `Unit` ke `OAuthClientController`
9. `OAuthClientController` memberikan *response* ke *front-end* melalui `OAuthClientService` karena sebelumnya melakukan *request*
10. `OAuthClientService` meneruskan hasil *response* ke `RemoveOAuthClientProgressDialogComponent` dalam bentuk `Observable`
11. `RemoveOAuthClientProgressDialogComponent` memberikan sebuah status ke `ListOAuthClientComponent` bahwa data berhasil dihapus atau tidak
12. `ListOAuthClientComponent` akan memanggil `InfoDialogComponent` untuk memberikan informasi kepada pengguna (*resource owner*) bahwa data berhasil dihapus dari sistem basis data atau tidak.

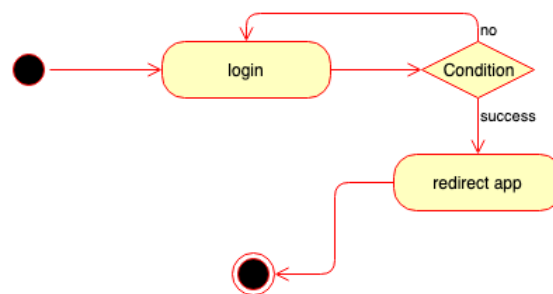
5.2.7 Diagram *Activity*

Diagram ini masuk dalam kategori diagram *behavior* yang menunjukkan alur kontrol atau alur objek yang dipertegas dalam urutan aktivitas dan kondisi pada alur yang terjadi.

Seperti halnya diagram *communication*, akan lebih mudah apabila diagram ini dibentuk dari beberapa skenario yang telah ditetapkan seperti berikut :

- Diagram *Activity Login*

Diagram *activity* untuk kasus *login* adalah seperti berikut :



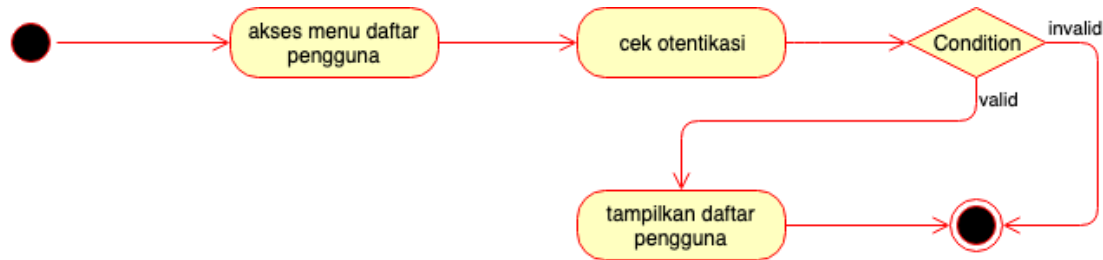
Gambar 44: Diagram *Activity* Untuk *Login*

Awalnya saat pengguna (*resource owner*) mencoba melakukan akses yang membutuhkan *login* dari aplikasi klien (*user agent*), maka halaman *login* akan muncul, kemudian pengguna mengisi *username* dan *password* untuk melakukan otentikasi.

Apabila kondisi otentikasi berhasil dilakukan, maka proses berikutnya pengguna akan diarahkan ke halaman data yang menjadi kewenangannya atau hak aksesnya (*role*), namun bila gagal, pengguna akan tetap berada pada halaman *login*.

- Diagram *Activity* Menampilkan Daftar Pengguna (*Resource Owner*)

Diagram *activity* untuk menampilkan daftar pengguna yang telah terdaftar dalam sistem otentikasi adalah seperti berikut :

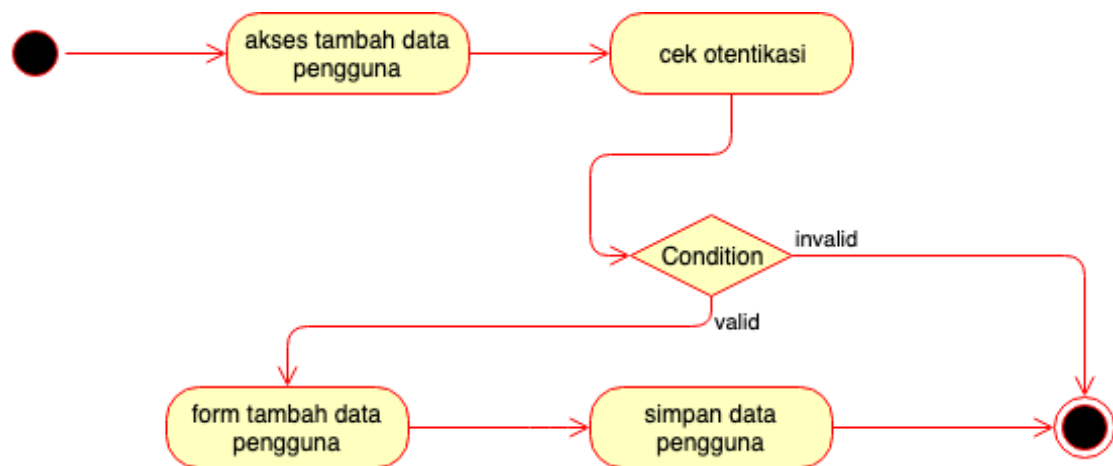


Gambar 45: Diagram *Activity* Untuk Menampilkan Daftar Pengguna (*Resource Owner*)

Diagram tersebut menggambarkan pada saat akan melakukan akses ke menu untuk menampilkan daftar pengguna, prosesnya akan melalui pemeriksaan otentikasi, apakah pengguna (*resource owner*) dalam kewenangannya melakukan akses terhadap halaman tersebut, bila ya, maka daftar pengguna akan ditampilkan, namun bila tidak, halaman tersebut tidak akan ditampilkan.

- Diagram *Activity* Menambah Data Pengguna (*Resource Owner*)

Diagram *activity* untuk menambah data pengguna (*resource owner*) baru pada sistem otentikasi adalah seperti berikut :

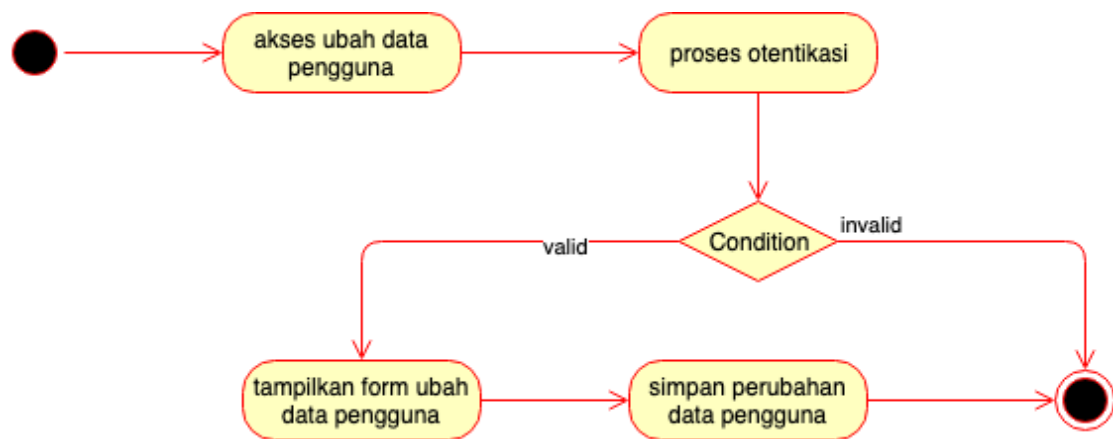


Gambar 46: Diagram *Activity* Untuk Menambah Data Pengguna (*Resource Owner*) Baru

Diagram tersebut menjelaskan bahwa ketika ada akses ke fitur tambah data pengguna, maka akan dilakukan otentikasi terlebih dahulu, apabila hasil dari otentikasi menyatakan bahwa pengguna (*resource owner*) tersebut memiliki hak akses (*role*) atau kewenangan pada fitur ini, maka formulir tambah data akan ditampilkan kemudian dapat dilakukan simpan data padanya, namun bila otentikasi gagal, maka proses tidak akan menampilkan formulir tambah data pengguna.

- Diagram *Activity* Mengubah Data Pengguna (*Resource Owner*)

Diagram *activity* untuk mengubah data pengguna (*resource owner*) yang telah terrekam pada sistem otentikasi adalah seperti berikut ini :

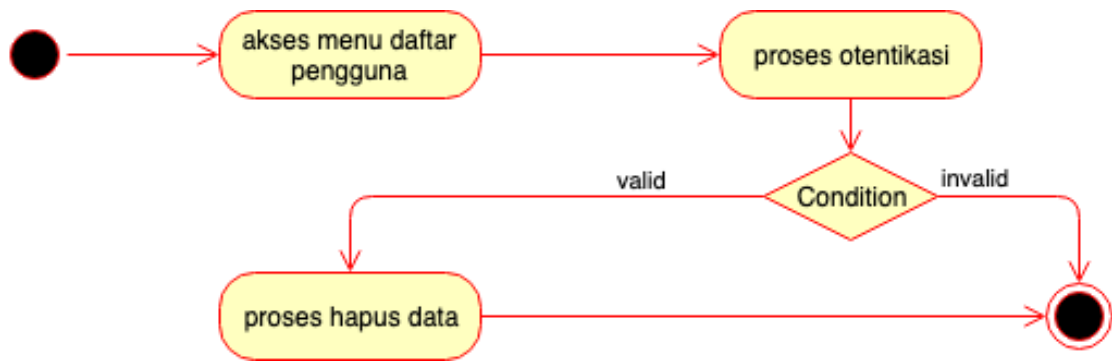


Gambar 47: Diagram *Activity* Untuk Mengubah Data Pengguna (*Resource Owner*)

Diagram tersebut menjelaskan pada saat pengguna (*resource owner*) melakukan akses ke fitur ubah data pengguna, maka proses otentikasi akan dijalankan, apabila pengguna (*resource owner*) memiliki hak akses atau kewenangan untuk menggunakan fitur ini, maka formulir ubah data pengguna akan ditampilkan dan fitur untuk simpan hasil dari perubahan data pengguna dapat dilakukan, namun apabila otentikasi gagal, maka formulir tidak akan ditampilkan

- Diagram *Activity* Menghapus Data Pengguna (*Resource Owner*)

Diagram *activity* untuk menghapus data pengguna (*resource owner*) yang telah terrekam pada sistem otentikasi adalah seperti berikut :

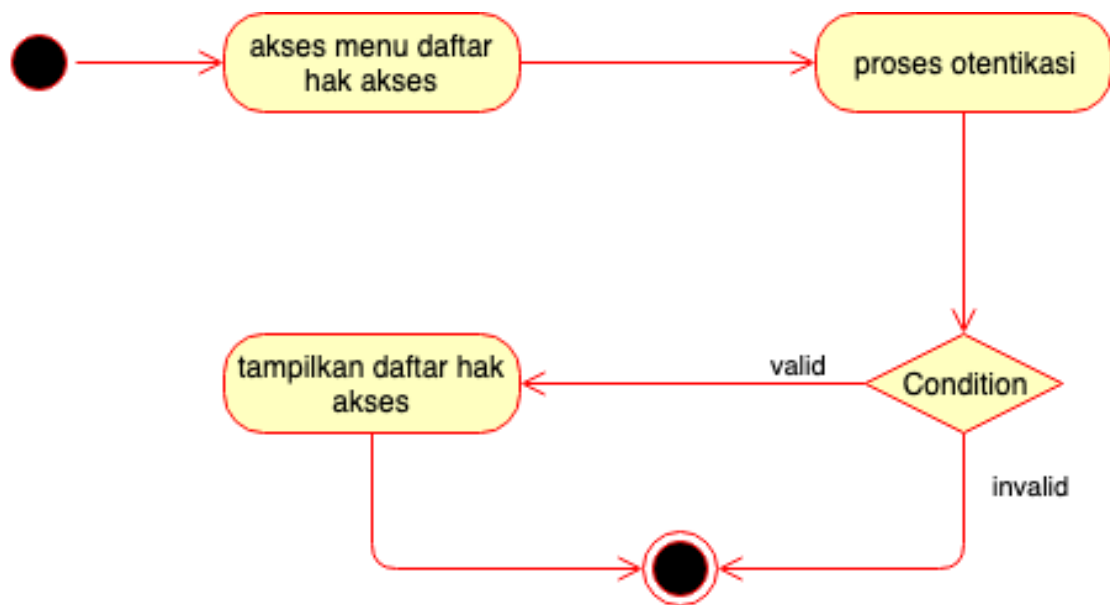


Gambar 48: Diagram *Activity* Untuk Menghapus Data Pengguna (*Resource Owner*)

Diagram tersebut menjelaskan pada saat pengguna akan melakukan proses hapus data pengguna (*resource owner*), maka pengguna melakukan akses terhadap menu daftar pengguna terlebih dahulu, kemudian proses otentikasi akan terjadi, apabila pengguna (*resource owner*) memiliki kewenangan akses terhadap fitur ini, maka pengguna dapat melakukan proses hapus data, apabila tidak memiliki kewenangan, maka fitur hapus data tidak dapat diakses oleh pengguna (*resource owner*).

- Diagram *Activity* Menampilkan Daftar Hak Akses (*Role*)

Diagram *activity* untuk menampilkan daftar hak akses / kewenangan (*role*) yang telah terrekam pada sistem otentikasi adalah seperti berikut :

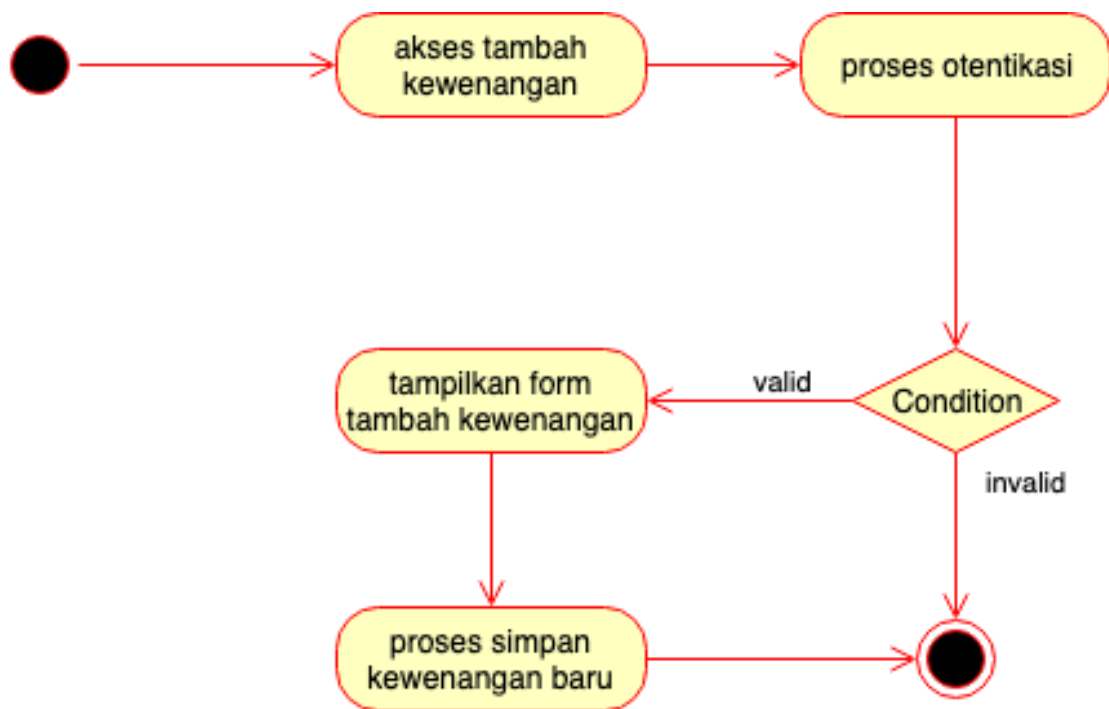


Gambar 49: Diagram *Activity* Untuk Menampilkan Hak Akses (*Role*)

Diagram tersebut menjelaskan pada saat pengguna (*resource owner*) akan melakukan akses terhadap daftar hak akses atau kewenangan (*role*) yang kemudian akan dilakukan otentikasi terlebih dahulu apakah pengguna telah sesuai dengan kewenangannya melakukan akses terhadap daftar hak akses, apabila dinyatakan sesuai, maka aplikasi klien akan menampilkan daftar hak akses / kewenangan (*role*), namun bisa tidak maka prosesnya akan dilewati.

- Diagram *Activity* Menambahkan Hak Akses (*Role*)

Diagram *activity* untuk menambahkan data kewenangan (*role*) dalam sistem otentikasi adalah seperti berikut :

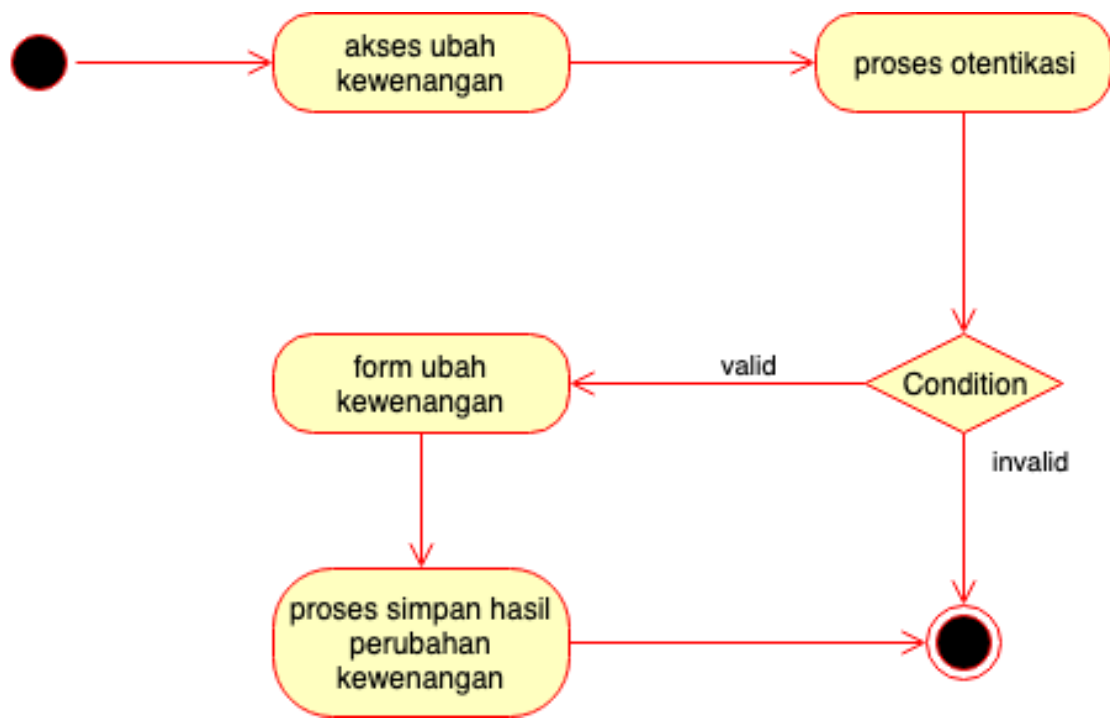


Gambar 50: Diagram *Activity* Untuk Menambahkan Data Kewenangan (*Role*)

Diagram tersebut menjelaskan pada saat pengguna (*resource owner*) akan melakukan akses ke fitur tambah kewenangan, maka proses otentikasinya berjalan untuk memastikan apakah pengguna (*resource owner*) dalam kewenangannya untuk menambahkan data hak akses, bila ditemukan bahwa pengguna (*resource owner*) memiliki kewenangan tersebut, maka formulir untuk menambahkan hak akses akan muncul dan proses simpan data kewenangan dapat dilakukan, setelah itu proses selesai, namun bila pengguna tidak memiliki kewenangan tersebut, maka prosesnya akan dilewati.

- Diagram *Activity* Mengubah Hak Akses (*Role*)

Diagram *activity* untuk mengubah data kewenangan (*role*) dalam sistem otentikasi adalah seperti berikut :

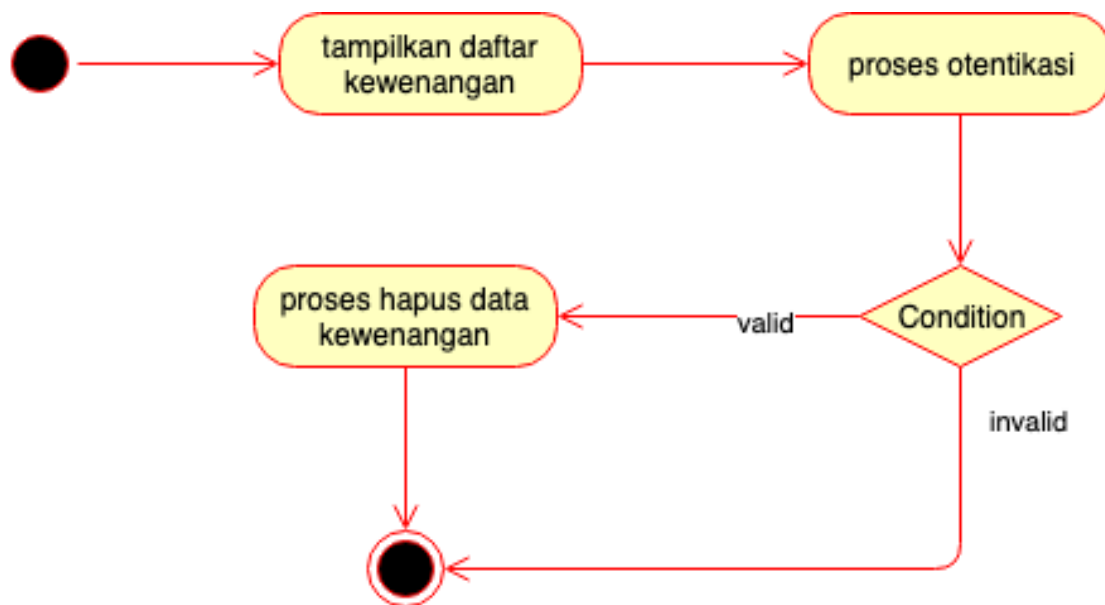


Gambar 51: Diagram *Activity* Untuk Mengubah Data Kewenangan (*Role*)

Diagram tersebut menjelaskan pada saat pengguna (*resource owner*) akan melakukan akses ke ubah kewenangan, proses otentikasi akan dilakukan terhadap pengguna (*resource owner*), bila pengguna (*resource owner*) memiliki kewenangan untuk mengubah data, maka formulir untuk mengubah kewenangan (*role*) akan ditampilkan, dan proses simpan perubahan data kewenangan dapat dilakukan, kemudian prosesnya selesai, namun bila pengguna (*resource owner*) tidak memiliki hak atas perubahan tersebut, maka proses menampilkan form untuk mengubah data kewenangan dan proses simpan data akan dilewati.

- Diagram *Activity* Menghapus Hak Akses (*Role*)

Diagram *activity* untuk menghapus data kewenangan (*role*) dalam sistem otentikasi adalah seperti berikut :

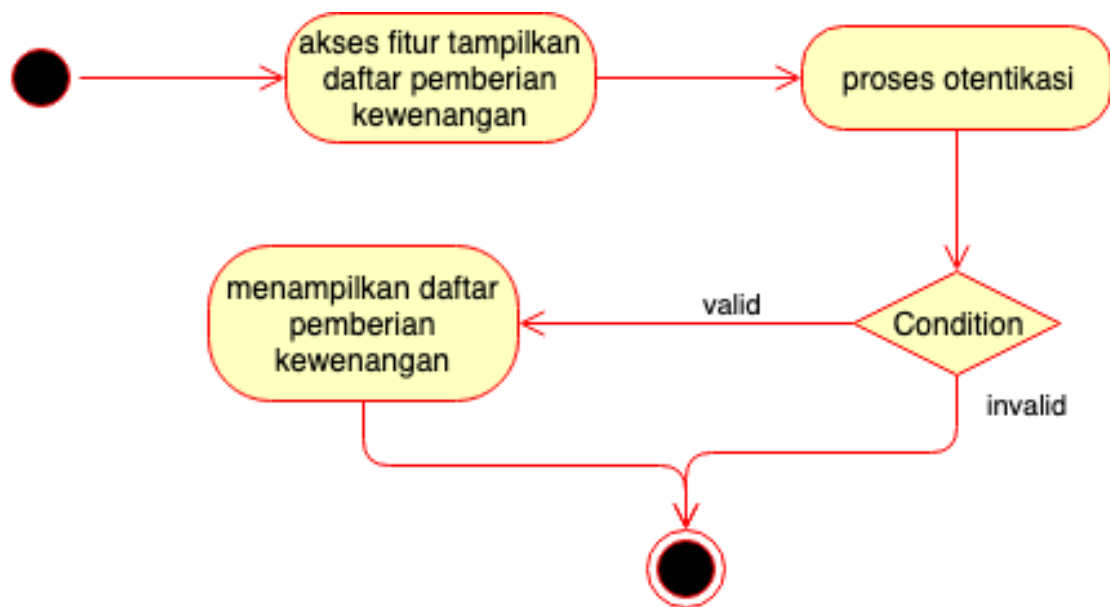


Gambar 52: Diagram *Activity* Untuk Menghapus Data Kewenangan (*Role*)

Diagram tersebut menjelaskan pada saat pengguna (*resource owner*) akan melakukan proses hapus data, pengguna (*resource owner*) melakukan akses dulu ke tampilan daftar kewenangan, setelah itu proses otentikasi akan berjalan untuk memastikan bahwa pengguna (*resource owner*) dalam kewenangannya untuk melakukan hapus data kewenangan, bila pengguna (*resource owner*) memiliki kewenangan (*role*) untuk melakukan hapus data hak akses, maka fitur untuk menghapus data kewenangan dapat dilakukan, namun bila tidak prosesnya akan terlewati.

- Diagram *Activity* Menampilkan Daftar Pemberian Hak Akses (*Role*)

Diagram *activity* untuk menampilkan daftar pemberian hak akses (*role*) dalam sistem otentikasi akan terlihat seperti berikut :

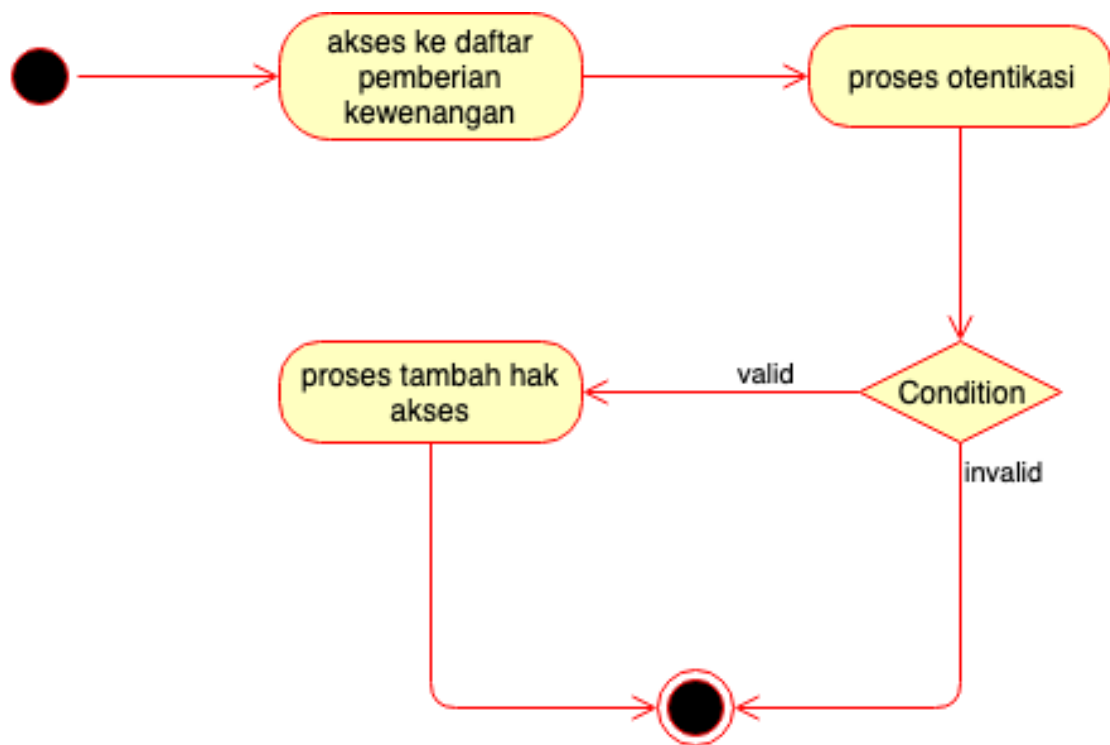


Gambar 53: Diagram *Activity* Untuk Menampilkan Daftar Pemberian Hak Akses (*Role*)

Diagram tersebut menjelaskan pada saat pengguna (*resource owner*) melakukan akses untuk menampilkan daftar pemberian kewenangan, maka proses otentikasi akan terjadi, bila pengguna (*resource owner*) memiliki kewenangan itu, maka daftar pemberian kewenangan akan ditampilkan, namun bila tidak, prosesnya akan dilewati.

- Diagram *Activity* Menambahkan Pemberian Hak Akses (*Role*)

Diagram *activity* untuk menambahkan pemberian hak akses (*role*) ke pengguna adalah seperti gambar berikut ini :

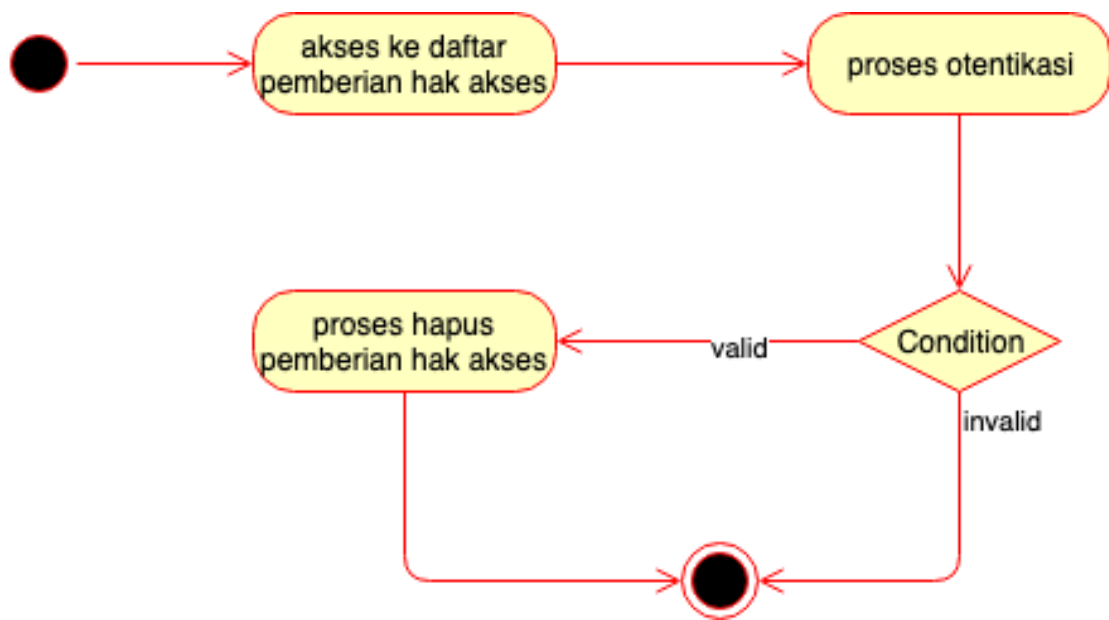


Gambar 54: Diagram *Activity* Untuk Menambahkan Pemberian Hak Akses (*Role*)

Diagram tersebut menjelaskan pada saat akan memberikan penambahan hak akses baru, pengguna (*resource owner*) akan melakukan akses ke daftar pemberian hak akses terlebih dahulu, sampai sini proses otentikasi akan berjalan, bila pengguna (*resource owner*) memiliki kewenangan itu, proses tambah hak akses (*role*) akan dilakukan, namun bila tidak, maka prosesnya akan dilewati.

- Diagram *Activity* Menghapus Pemberian Hak Akses (*Role*)

Diagram *activity* untuk menghapus pemberian hak akses (*role*) yang telah tersimpan dalam sistem otentikasi adalah seperti gambar berikut :

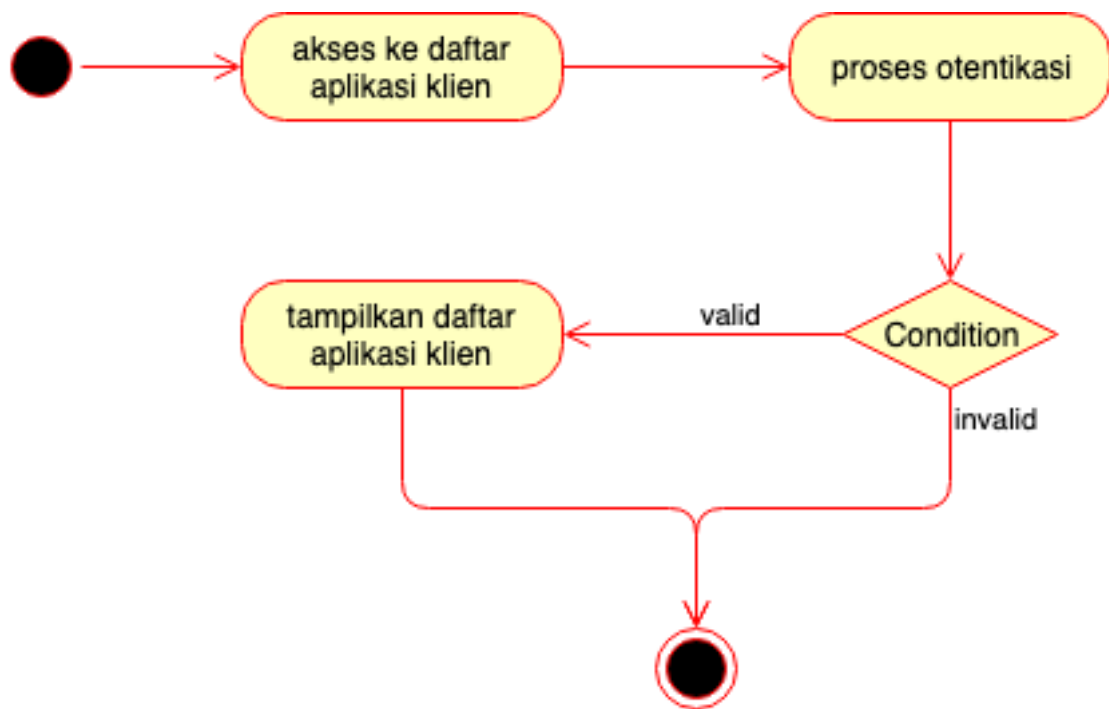


Gambar 55: Diagram *Activity* Untuk Menghapus Pemberian Hak Akses (*Role*)

Diagram tersebut menjelaskan pada saat pengguna (*resource owner*) akan melakukan penghapusan terhadap pemberian kewenangan yang telah tersimpan dalam sistem otentikasi, pengguna (*resource owner*) akan melakukan akses pada daftar pemberian hak akses (*role*) terlebih dahulu, kemudian proses otentikasi akan berjalan, bila pengguna memiliki kewenangan untuk melakukan penghapusan pemberian kewenangan (*role*) terhadap suatu pengguna (*resource owner*), maka proses penghapusan dapat dilakukan, namun bila tidak, maka proses tersebut akan dilewati.

- Diagram *Activity* Menampilkan Daftar Aplikasi Klien (*User Agent*)

Diagram *activity* untuk menampilkan daftar aplikasi klien (*user agent*) yang telah terdaftar dalam sistem otentikasi ini adalah seperti pada gambar berikut :

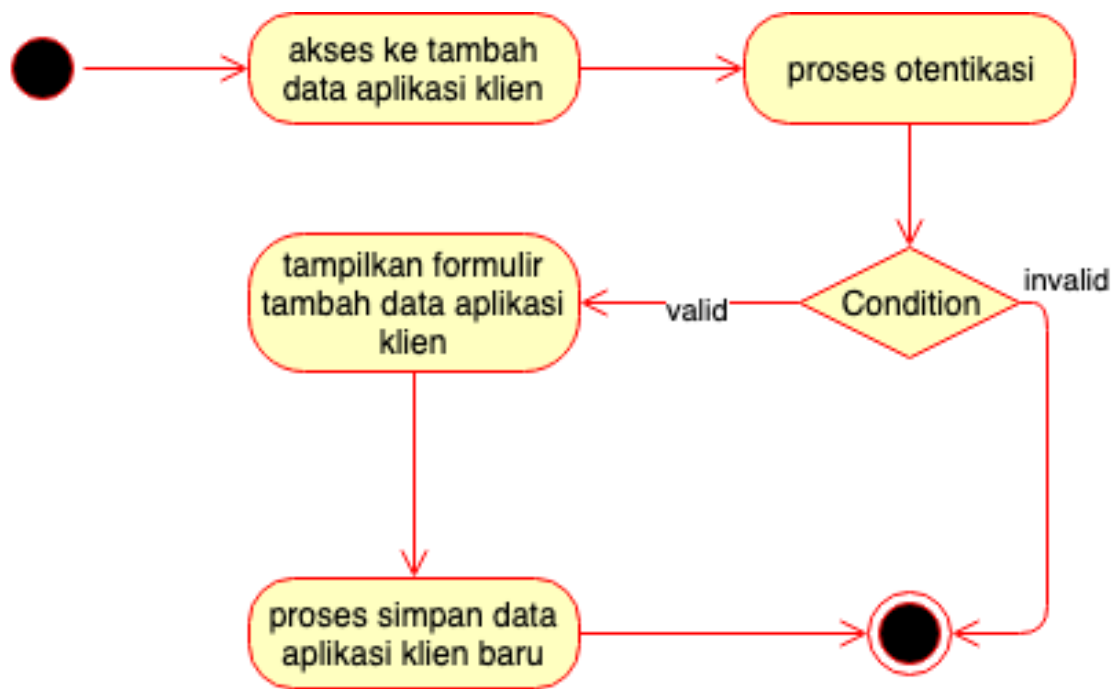


Gambar 56: Diagram *Activity* Untuk Menampilkan Daftar Aplikasi Klien (*User Agent*)

Diagram tersebut menjelaskan pada saat pengguna (*resource owner*) akan melihat daftar aplikasi klien yang telah tersimpan dalam sistem otentikasi, aplikasi akan melalui proses otentikasi pengguna, apakah pengguna (*resource owner*) dalam kapasitas atau kewenangannya memiliki hak untuk melihat daftar aplikasi klien (*user agent*) atau tidak, bila ya (*valid*), maka aplikasi akan menampilkan daftar aplikasi klien, namun bila tidak maka prosesnya akan dilewati.

- Diagram *Activity* Menambah Data Aplikasi Klien (*User Agent*)

Diagram *activity* untuk menambah data aplikasi klien (*user agent*) yang baru ke dalam sistem otentikasi adalah seperti gambar berikut :

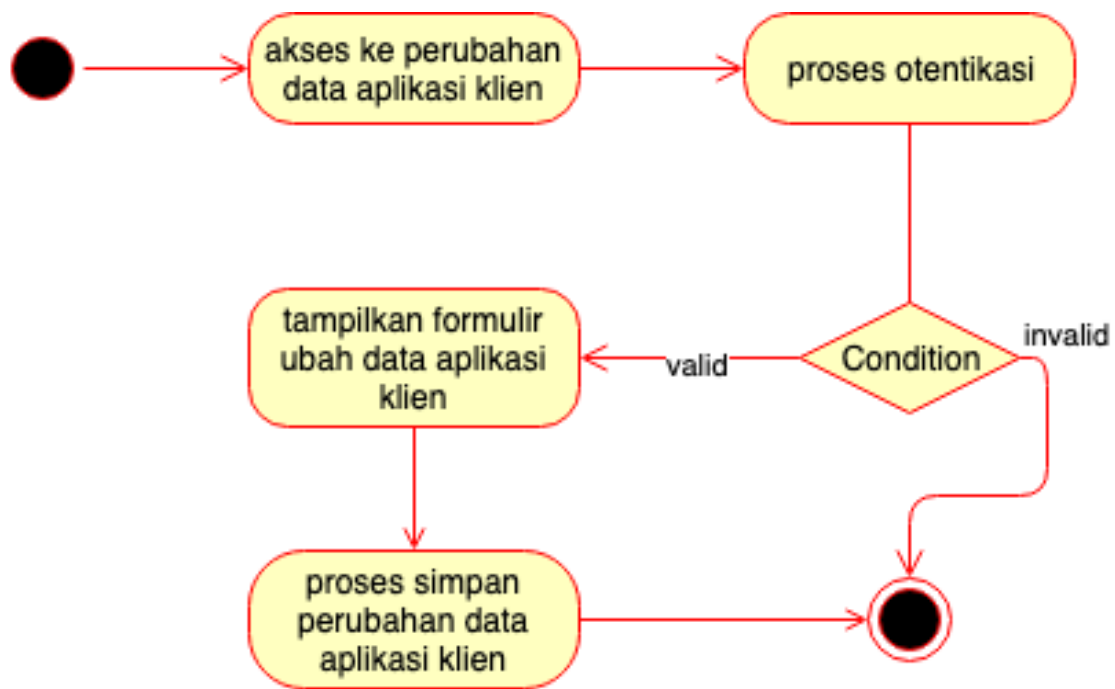


Gambar 57: Diagram *Activity* Untuk Menambahkan Data Aplikasi Klien (*User Agent*)

Diagram tersebut menjelaskan pada saat pengguna melakukan akses ke fitur tambah data aplikasi klien (*user agent*), proses otentikasi akan berjalan untuk memastikan bahwa pengguna (*resource owner*) memiliki hak akses terhadap fitur ini, bila ya, maka aplikasi akan menampilkan formulir untuk mengisi data dari aplikasi klien yang akan didaftarkan, kemudian proses simpan dapat dilakukan, namun bila proses otentikasi gagal dan menyatakan bahwa pengguna (*resource owner*) tidak memiliki hak akses, maka prosesnya akan dilewati.

- Diagram *Activity* Mengubah Data Aplikasi Klien (*User Agent*)

Diagram *activity* untuk mengubah data aplikasi klien (*user agent*) yang telah terrekam dalam sistem otentikasi adalah seperti pada gambar berikut :



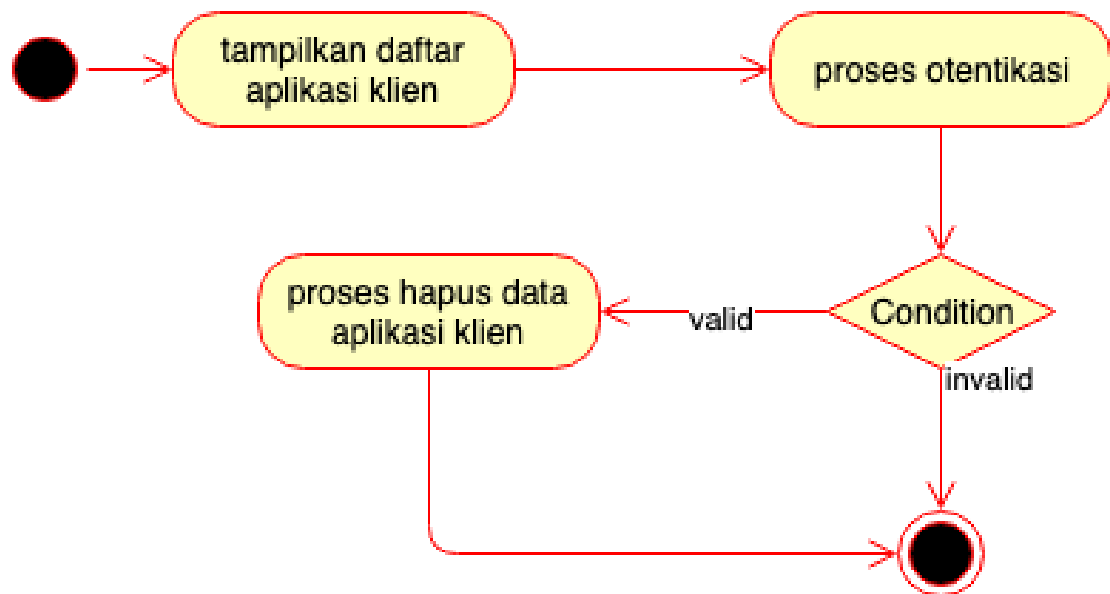
Gambar 58: Diagram *Activity* Untuk Mengubah Data Aplikasi Klien (*User Agent*)

Diagram tersebut akan menjelaskan proses simpan perubahan data dari aplikasi klien yang telah terrekam dalam sistem otentikasi, pertama pengguna (*resource owner*) akan melakukan akses ke fitur perubahan data aplikasi klien, kemudian proses otentikasi akan berjalan dan memastikan apakah pengguna (*resource owner*) memiliki hak akses untuk mengubah data aplikasi klien atau tidak, bila ya, maka aplikasi akan menampilkan formulir untuk menubah data aplikasi klien, kemudian proses simpan perubahan data aplikasi klien dapat dilakukan, namun bila pengguna (*resource owner*) tidak memiliki hak akses, maka proses menampilkan formulir perubahan data aplikasi klien dan proses simpan data perubahan akan dilewati.

- Diagram *Activity* Menghapus Data Aplikasi Klien (*User Agent*)

Diagram *activity* untuk menghapus data aplikasi klien (*user agent*) yang

telah terrekam dalam sistem otentikasi adalah seperti gambar berikut :



Gambar 59: Diagram *Activity* Untuk Menghapus Data Aplikasi Klien (*User Agent*)

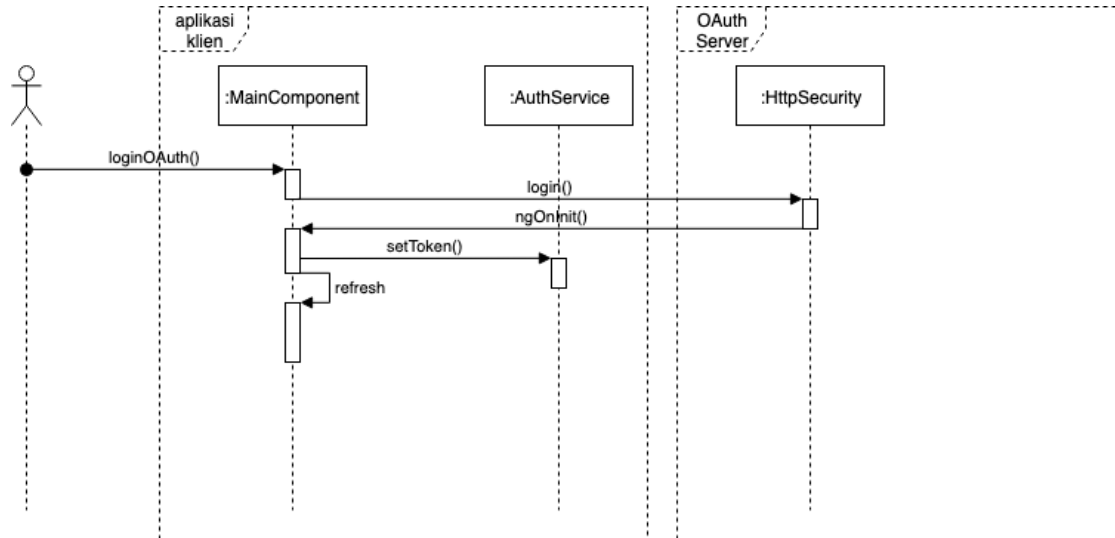
Diagram tersebut akan menjelaskan proses hapus data dari aplikasi klien yang terrekam dalam sistem otentikasi, pertama pengguna perlu melakukan akses ke tampilkan daftar aplikasi klien, setelah itu proses otentikasi akan berjalan untuk menentukan apakah pengguna memiliki kewenangan untuk menghapus data aplikasi klien yang telah terrekam atau tidak, bila ya, maka proses hapus data aplikasi klien dapat dilakukan, namun bila tidak, proses akan dilewatkan.

5.2.8 Diagram *Sequence*

Diagram ini akan menggambarkan alur pertukaran pesan dari beberapa objek pada rentang siklus hidupnya. Diagram ini pun akan kita bagi menjadi beberapa bagian sesuai dengan *case* atau fitur yang dimiliki oleh sistem otentikasi, berikut daftarnya :

- Diagram *Sequence Login*

Diagram *sequence* untuk proses *login* akan digambarkan seperti berikut ini :



Gambar 60: Diagram *Sequence* Untuk Proses *Login*

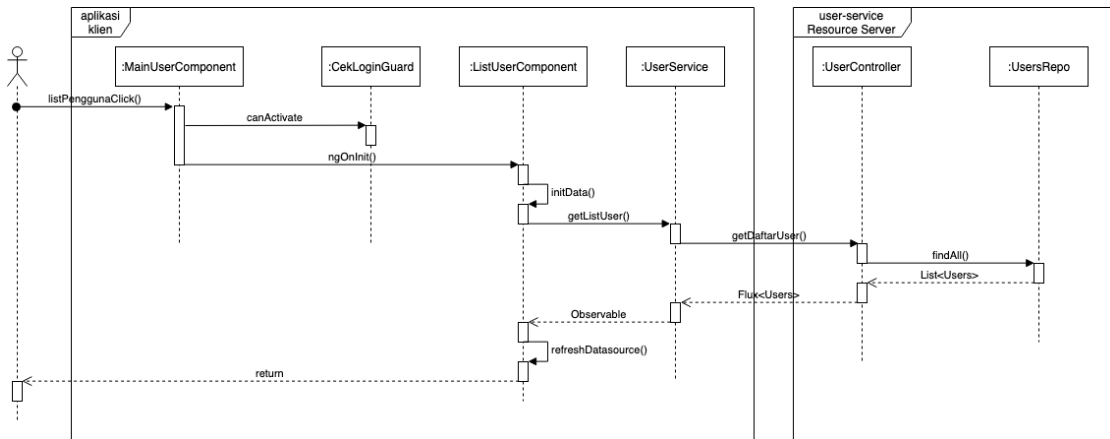
Pertama pengguna (*resource owner*) melakukan aksi untuk melakukan *login* melalui **MainComponent**, *method* yang dipanggil adalah *method* `loginOAuth()`.

Selanjutnya dari **MainComponent** mengarahkan pengguna ke halaman *login* pada *OAuth Server*, setelah proses *login* berhasil, *OAuth Server* akan mengembalikan sebuah token ke **MainComponent** yang kemudian token ini disimpan pada **AuthService**.

Tahap berikutnya **MainComponent** melakukan *refresh* halaman untuk membuka beberapa menu sesuai dengan kewenangan pengguna yang sebelumnya disembunyikan.

- Diagram *Sequence* Untuk Menampilkan Daftar Pengguna (*Resource Owner*)

Diagram *sequence* untuk menampilkan daftar pengguna (*resource owner*) adalah seperti gambar berikut :



Gambar 61: Diagram *Sequence* Untuk Menampilkan Daftar Pengguna

Pertama pengguna (*resource owner*) melakukan akses ke fitur menampilkan daftar pengguna melalui **MainUserComponent**. Kemudian aplikasi klien melakukan pemeriksaan kewenangan, apakah hak aksesnya memenuhi syarat atau tidak, apabila memenuhi syarat, maka akan ditampilkan daftar pengguna melalui **ListUserComponent**.

Berikutnya **ListUserComponent** melakukan akses ke fungsi **getListUser()** milik **UserService**, yang kemudian diteruskan dalam bentuk *request* ke *resource server* oleh **UserService** melalui akses ke *method* **getDaftarUser()** milik kelas **UserController**.

UserController kemudian melakukan akses ke sistem basis data melalui *method* **findAll()** milik **UsersRepo**. Hasil dari *method* **findAll()** milik **UsersRepo** berupa **List<Users>** yang dikembalikan ke **UserController**.

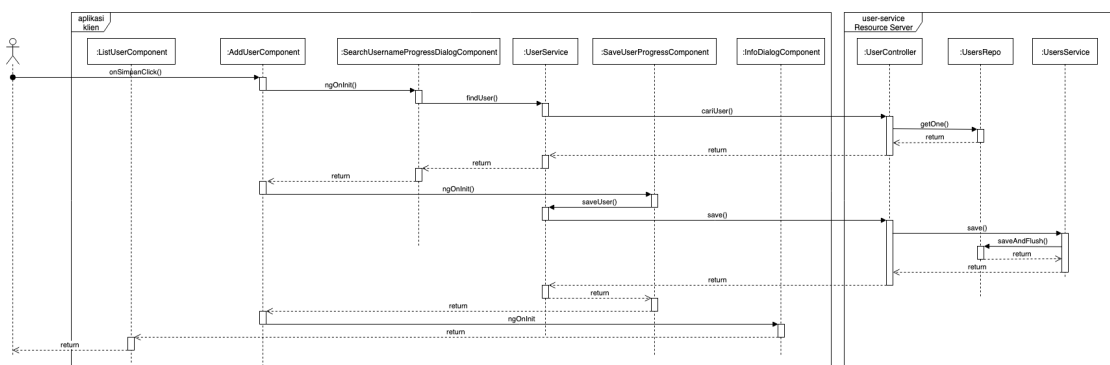
UserController kemudian mengirimkan *response* ke aplikasi klien (*user agent*) berupa **Flux<Users>** melalui **UserService** yang sebelumnya mela-

kukan *request*.

Kemudian *UserService* meneruskan *response* berupa *Observable* ke *ListUserComponent*, yang akhirnya digunakan untuk melakukan pembaruan data pada tabel yang dapat dikonsumsi oleh pengguna.

- Diagram *Sequence* Untuk Menambah Data Pengguna (*Resource Owner*)

Diagram *sequence* untuk proses menambahkan data pengguna (*Resource Owner*) adalah seperti berikut :



Gambar 62: Diagram *Sequence* Untuk Menambahkan Data Pengguna

Penjelasan dari diagram tersebut yaitu pertama, pengguna (*resource owner*), dalam hal ini yang memiliki kewenangan tentu saja, melakukan akses ke formulir tambah data pengguna (*resource owner*) melalui *AddUserComponent*, kemudian prosesnya berlanjut menampilkan *SearchUsernameProgressDialogComponent*, dari komponen ini melakukan akses ke fungsi *findUser()* milik *UserService*, dari *UserService* mencoba melakukan *request* ke *resource server*, dalam hal ini melalui *method* *cariUser()* milik kelas *UserController*.

Method dari kelas *UserController* kemudian melakukan akses ke *method* *getOne()* milik *UsersRepo*, yang kemudian akses ke sistem basis data ter-

jadi. Apabila data yang dicari ada dalam sistem basis data, maka *method* `getOne()` akan mengembalikan detail dari pengguna tersebut, namun bila tidak ditemukan, maka data yang dikembalikan adalah data kosong.

Setelah `UserController` menerima *response* atau data dari `UsersRepo`, `UserController` melakukan *response* ke aplikasi klien, dalam hal ini melalui `UserService`, kemudian proses datanya berlanjut ke `SearchUsernameProgressDialogComponent` sampai kembali ke `AddUserComponent`.

Bila data yang akan disimpan belum terdaftar dalam sistem basis data, maka dari `AddUserComponent` akan memanggil `SaveUserProgressComponent`, kemudian dari komponen `SaveUserProgressComponent` akan melakukan akses ke fungsi `saveUser()` milik `UserService`, yang kemudian dari fungsi tersebut akan berlanjut melakukan *request* ke *resource server* melalui *method* `save` milik `UserController`.

Dari *method* `save` milik `UserController`, kemudian akan melakukan akses ke *method* `save` milik `UsersService`, yang kemudian dari sini akan melakukan akses ke *method* `saveAndFlush()` milik `UsersRepo`.

Dari *method* `saveAndFlush()` milik `UsersRepo` proses pencatatan ke sistem basis data terjadi, kemudian *method* ini akan mengembalikan objek berupa `Users` ke `UsersService`, yang dilanjutkan dikirim ke `UserController`.

Dari `UserController` kemudian akan melakukan *response* ke aplikasi klien berupa objek `Users` dalam bentuk JSON, yang diterima di `UserService`.

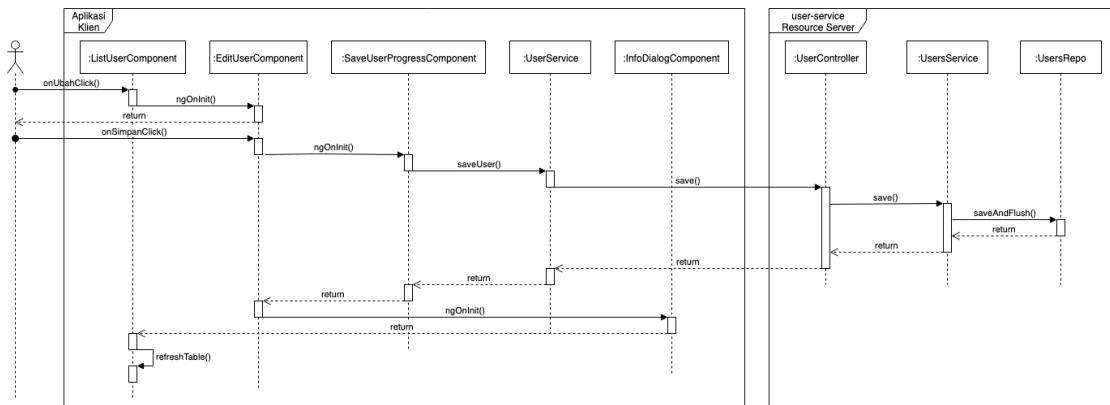
Dari `UserService` kemudian akan mengirimkan datanya ke `SaveUserProgressComponent` dalam bentuk `Observable` yang kemudian diteruskan pesannya dalam bentuk status ke `AddUserComponent`.

Setelah `AddUserComponent` menangkap pesan dari

`SaveUserProgressComponent`, kemudian memanggil `InfoDialogComponent` untuk memberitahu pengguna bahwa data yang diproses untuk disimpan telah tersimpan dalam sistem basis data.

- Diagram *Sequence* Untuk Mengubah Data Pengguna (*Resource Owner*)

Diagram *sequence* untuk proses mengubah data pengguna (*resource owner*) adalah seperti gambar berikut :



Gambar 63: Diagram *Sequence* Untuk Mengubah Data Pengguna

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna (*resource owner*) melakukan akses ke fitur Ubah Data Pengguna pada daftar pengguna seperti ditunjukkan pada akses ke fungsi `onUbahClick()` milik `ListUserComponent`. Kemudian `ListUserComponent` melakukan pemanggilan ke `EditUserComponent` sehingga muncul jendela formulir untuk mengubah data pengguna (*resource owner*).

Setelah mengisi perubahan informasi data pengguna, maka pengguna (*resource owner*) melakukan simpan perubahan data melalui fungsi `onSimpanClick()` milik `EditUserComponent`, kemudian komponen ini melakukan pemanggilan ke `SaveUserProgressComponent` untuk melanjutkan prosesnya menyimpan perubahan data pengguna.

Kemudian `SaveUserProgressComponent` melakukan akses ke fungsi `saveUser()` milik `UserService` yang dilanjutkan dengan aksi *request* dari `UserService` ke *resource server*, dalam hal ini adalah *method* `save()` milik `UserController`.

Prosesnya kemudian berlanjut dari `UserController` melakukan akses ke *method* `save()` milik `UsersService`, dan di dalamnya melakukan akses ke *method* `saveAndFlush()` milik `UsersRepo`.

Sampai sini proses interaksi dengan sistem basis data terjadi, setelah perubahan berhasil disimpan dalam sistem basis data, maka `UsersRepo` akan mengembalikan sebuah objek ke `UsersService` berbentuk kelas `Users` yang artinya data telah tersimpan.

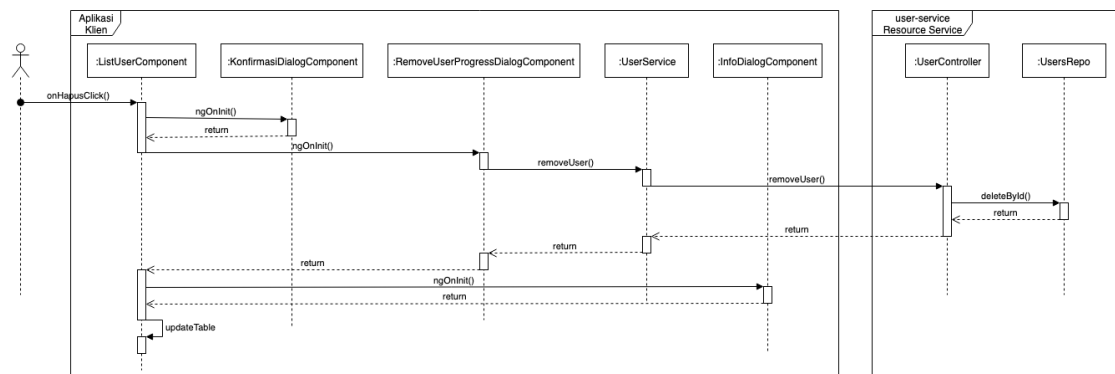
Objek `Users` akan diteruskan dari `UsersService` ke `UserController`, yang kemudian `UserController` akan memberikan *response* ke aplikasi klien (*user agent*), dalam hal ini ke `UserService` karena sebelumnya melakukan *request*.

Dari `UserService` kemudian akan mengembalikan data ke `SaveUserProgressComponent` dalam bentuk `Observable`, yang kemudian akan diteruskan ke `EditUserComponent`. Dari komponen `EditUserComponent` ini kemudian memanggil `InfoDialogComponent` untuk memberitahukan status berhasil atau tidaknya hasil perubahan disimpan dalam sistem basis data ke pengguna (*resource owner*).

Terakhir dari `InfoDialogComponent` kemudian akan mengembalikan halaman aktif ke `ListUserComponent` yang kemudian akan melakukan pembaruan data pada tabel agar data yang telah diubah diketahui perubahannya.

- Diagram *Sequence* Untuk Menghapus Data Pengguna (*Resource Owner*)

Diagram *sequence* untuk proses penghapusan data pengguna (*resource owner*) dari sistem otentikasi adalah seperti gambar berikut :



Gambar 64: Diagram *Sequence* Untuk Menghapus Data Pengguna

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna (*resource server*) menggunakan fitur hapus data pengguna melalui fungsi `onHapusClick()` milik `ListUserComponent`, kemudian `ListUserComponent` melakukan pemanggilan ke `KonfirmasiDialogComponent` untuk melakukan konfirmasi apakah pengguna (*resource owner*) akan menghapus data tersebut.

Bila pengguna (*resource owner*) melakukan konfirmasi untuk hapus data, prosesnya akan berlanjut ke pemanggilan `RemoveUserProgressDialogComponent`, yang di dalamnya akan memanggil fungsi `removeUser()` milik `UserService`.

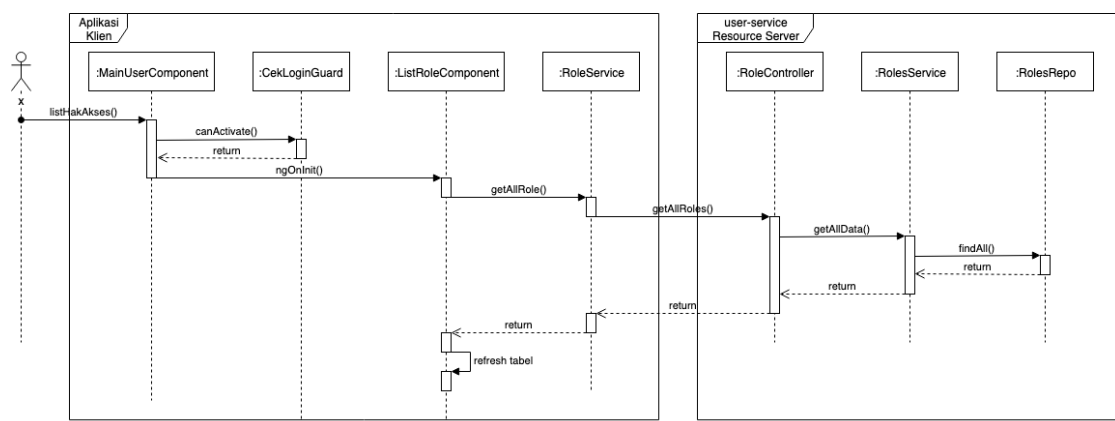
Selanjutnya `UserService` melakukan *request* ke *resource server* melalui *method* `removeUser()` milik `UserController`. Setelah itu `UserController` akan melakukan akses ke *method* `deleteById()` milik `UsersRepo` yang kemudian melakukan operasi hapus data pada sistem basis data.

Kemudian *method* `deleteById()` mengembalikan sebuah nilai ke `UserController` dan diteruskan menjadi *response* ke aplikasi klien, dalam hal ini melalui `UserService` yang sebelumnya melakukan *request*.

Hasil *response* ini kemudian diteruskan ke `RemoveUserProgressDialogComponent` dalam bentuk `Observable`, yang kemudian dilanjutkan ke `ListUserComponent` dalam bentuk status. Dari status tersebut kemudian `ListUserComponent` melakukan pemanggilan ke `InfoDialogComponent` untuk memberitahu pengguna (*resource owner*) apakah datanya telah terhapus atau belum yang kemudian aplikasi klien akan melakukan pembaruan data pada tabelnya.

- Diagram *Sequence* Untuk Menampilkan Daftar Hak Akses (*Role*)

Diagram *sequence* untuk menampilkan daftar hak akses (*role*) yang telah terrekam dalam sistem otentikasi adalah seperti gambar berikut :



Gambar 65: Diagram *Sequence* Untuk Menampilkan Daftar Hak Akses

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna (*resource owner*) melakukan akses ke fitur menampilkan hak akses melalui fungsi `listHakAkses()` milik `MainUserComponent`, pada proses ini, fungsi `canActivate()` milik `CekLoginGuard` akan terpicu untuk melakukan verifikasi hak akses pengguna apakah memiliki kewenangan (*role*) tersebut atau tidak.

Bila pengguna memang memiliki hak akses, maka halaman akan diarahkan ke `ListRoleComponent` tempat daftar hak akses akan ditampilkan, di awalnya `ListRoleComponent` akan melakukan akses ke fungsi `getAllRole()` milik `RoleService`, kemudian `RoleService` melakukan *request* ke *resource server* melalui *method* `getAllRoles()` milik `RoleController`.

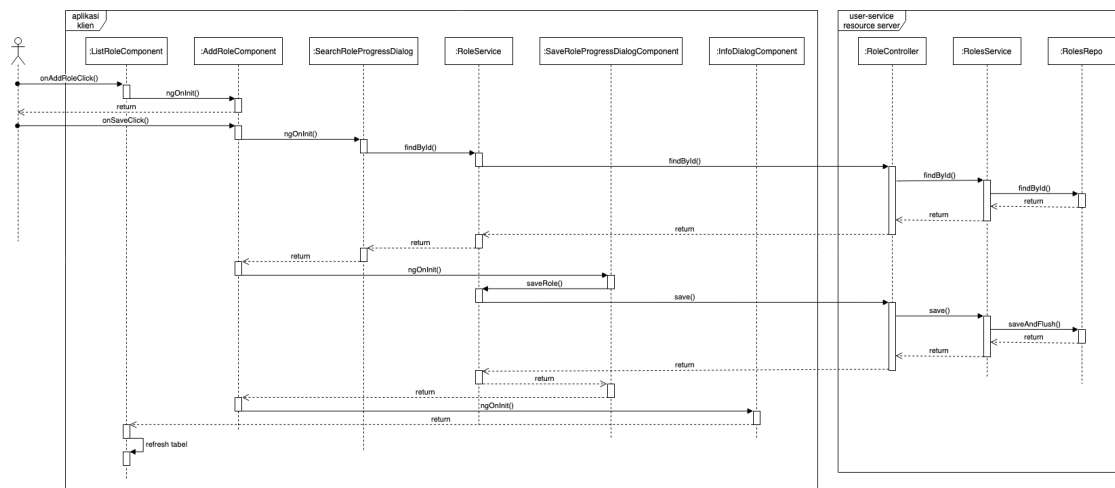
Kemudian `RoleController` akan melakukan akses ke *method* `getAllData()` milik `RolesService`, yang kemudian dari `RolesService` melakukan akses ke *method* `findAll()` milik `RolesRepo`. Dari `RolesRepo` inilah akses data ke sistem basis data terjadi yang kemudian akan mengembalikan nilai ke `RolesService` dalam bentuk `List<Roles>`.

Dari `RolesService` kemudian mengembalikan nilai ke `RoleController` dalam bentuk `Flux<Roles>`, yang kemudian digunakan sebagai bahan *response* ke aplikasi klien yang sebelumnya melakukan *request*, data akan diterima di aplikasi klien melalui `RoleService`.

Hasil yang diterima `RoleService` akan diteruskan ke `ListRoleComponent` dalam bentuk `Observable` yang kemudian digunakan sebagai bahan untuk melakukan pembaruan isi tabel daftar hak akses.

- Diagram *Sequence* Untuk Menambahkan Hak Akses (*Role*)

Diagram *sequence* untuk menambahkan data hak akses (*role*) adalah seperti pada gambar berikut :



Gambar 66: Diagram *Sequence* Untuk Menambahkan Data Hak Akses

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna melakukan akses ke fitur `ListRoleComponent`, kemudian melakukan akses ke fungsi `onAddRoleClick()` milik `ListRoleComponent`. Dari `ListRoleComponent` kemudian memanggil `AddRoleComponent` yang ditampilkan kepada pengguna untuk kemudian mengisi data mengenai hak akses yang akan dibuatkan.

Setelah pengguna (*resource owner*) mengisi data hak akses (*role*), pengguna akan menekan tombol simpan yang akan memicu akses ke fungsi `onSaveClick()` milik `AddRoleComponent`. Dari dalam fungsi `onSaveClick()` ini akan memanggil `SearchRoleProgressDialog` melalui fungsi `ngOnInit()`, kemudian dari fungsi `ngOnInit()` milik `SearchRoleProgressDialog` melakukan akses ke fungsi `findById()` ke `RoleService`.

Pada tahap ini, `RoleService` melakukan *request* ke *resource server* melalui *method* `findById()` milik kelas `RoleController`, kemudian di dalamnya akan melakukan akses ke *method* `findById()` milik `RolesService`, yang di

dalamnya akan melakukan akses ke *method* `findById()` milik `RolesRepo`.

Pada operasi di *method* `findById()` milik `RolesRepo` akan terjadi proses pencarian data pada sistem basis data, yang kemudian akan memberikan sebuah nilai balik berupa objek dari kelas `Optional<Roles>`, nilai ini dikirimkan ke `RolesService`.

Dari `RolesService`, objek dari `Optional<Roles>` akan dikirimkan ke `RoleController`. Kemudian `RoleController` akan mengirimkan objek tersebut sebagai *response* ke aplikasi klien melalui `RoleService` yang sebelumnya telah melakukan *request*.

Dari `RoleService` kemudian data hasil *response resource server* akan diteruskan ke `SearchRoleProgressDialog` dalam bentuk `Observable` yang kemudian diteruskan ke `AddRoleComponent`. Di `AddRoleComponent` kemudian akan melakukan seleksi apakah data hak akses sudah pernah dibuat atau belum, bila belum, maka proses berikutnya adalah memanggil `SaveRoleProgressDialogComponent` untuk melakukan simpan data.

Di awal pemanggilan `SaveRoleProgressDialogComponent` akan melakukan pemanggilan ke fungsi `saveRole()` milik `RoleService`, yang kemudian diteruskan menjadi *request* ke *resource server* melalui *method* `save()` milik `RoleController`.

Dari `RoleController` kemudian melakukan akses ke *method* `save()` milik `RolesService`, yang di dalamnya melakukan akses ke *method* `saveAndFlush()`, disini akses simpan data terjadi pada sistem basis data.

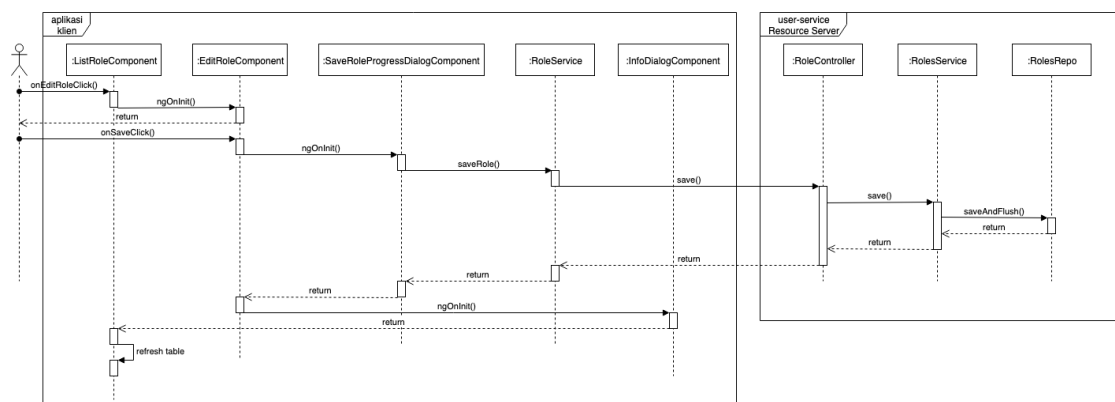
Setelah operasi simpan data pada sistem basis data selesai, `RolesRepo` mengembalikan sebuah nilai berupa objek `Roles` ke `RolesService`, yang kemudian diteruskan oleh `RolesService` ke `RoleController`.

Setelah itu `RoleController` akan mengirimkan sebuah *response* ke

aplikasi klien melalui `RoleService` karena sebelumnya melakukan *request* simpan data. Dari `RoleService` datanya diteruskan ke `SaveRoleProgressDialogComponent` dalam bentuk `Observable`, yang kemudian diolah dan mengirimkan status ke `AddRoleComponent`, dari status ini kemudian `AddRoleComponent` melakukan pemanggilan ke `InfoDialogComponent`, yang kemudian akan melakukan pengalihan tampilan ke `ListRoleComponent` yang secara otomatis akan melakukan pembaruan isi tabel daftar hak akses.

- Diagram *Sequence* Untuk Mengubah Hak Akses (*Role*)

Diagram *sequence* untuk mengubah data hak akses (*role*) adalah seperti pada gambar berikut :



Gambar 67: Diagram *Sequence* Untuk Mengubah Data Hak Akses

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna melakukan akses ke fitur ubah data melalui fungsi `onEditRoleClick()` milik `ListRoleComponent`, kemudian `EditRoleComponent` akan dipanggil dan pengguna dapat mengisi perubahan data pada komponen ini.

Selanjut pada saat pengguna (*user agent*) melakukan simpan data perubahan, prosesnya akan mengakses fungsi `onSaveClick()` milik

`EditRoleComponent`, di dalam fungsi `onSaveClick()` ini kemudian akan memanggil `SaveRoleProgressDialogComponent` untuk memulai proses simpan data perubahannya.

Di dalam `SaveRoleProgressDialogComponent` akan melakukan akses ke fungsi `saveRole()` milik `RoleService`, yang kemudian dari `RoleService` akan melakukan *request* ke *resource server* melalui *method* `save` milik kelas `RoleController`.

Dari kelas `RoleController`, kemudian akan melakukan akses ke *method* `save()` milik `RolesService`, yang diteruskan di dalam `RolesService` untuk melakukan akses ke *method* `saveAndFlush()` milik `RolesRepo`. Pada tahap inilah proses simpan data ke sistem basis data terjadi.

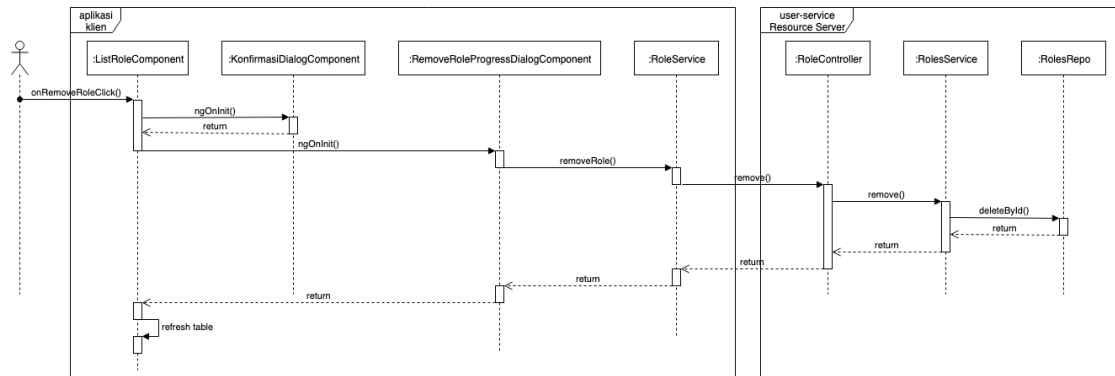
Setelah proses simpan data ke sistem basis data berhasil, `RolesRepo` akan mengembalikan sebuah objek dalam bentuk `Roles` ke `RolesService`, yang kemudian dari `RolesService` akan diteruskan ke `RoleController`.

Nilai dari objek `Roles` kemudian dijadikan bahan *response* data ke aplikasi klien melalui `RoleService` karena sebelumnya melakukan *request*. Dari `RoleService` kemudian hasil *response* akan dikirimkan ke `SaveRoleProgressDialogComponent` dalam bentuk `Observable`, yang kemudian diteruskan ke `EditRoleComponent` dalam bentuk status.

Nilai kembalian dalam bentuk status ini akan menjadi pemicu untuk kemudian memanggil `InfoDialogComponent` untuk memberikan informasi ke pengguna (*resource owner*) apakah data perubahan berhasil tersimpan atau tidak ke sistem basis data, yang pada akhirnya akan menjadi pemicu untuk melakukan pembaruan data tabel pada `ListRoleComponent`.

- Diagram *Sequence* Untuk Menghapus Hak Akses (*Role*)

Diagram *sequence* untuk menghapus hak akses (*role*) adalah seperti pada gambar berikut :



Gambar 68: Diagram *Sequence* Untuk Menghapus Data Hak Akses

Penjelasan dari diagram tersebut adalah seperti berikut ini, pertama pengguna berada pada `ListRoleComponent` dan memilih salah satu data untuk dihapus melalui fungsi `onRemoveRoleClick()` milik `ListRoleComponent`. Dari dalam fungsi ini kemudian memanggil `KonfirmasiDialogComponent` untuk memastikan bahwa data yang terpilih untuk dihapus adalah benar.

Setelah pengguna (*user agent*) memastikan data tersebut akan dihapus, kemudian prosesnya berlanjut dengan memanggil `RemoveRoleProgressDialogComponent`, yang di dalamnya akan menggunakan fungsi `removeRole()` milik `RoleService`.

Di dalam fungsi `removeRole()` milik `RolesService`, prosesnya akan melakukan *request* ke *resource server* melalui *method* `remove()` milik `RoleController`. Kemudian dari `RoleController` akan melakukan akses ke *method* `remove()` milik `RolesService`, dan di dalam *method* ini akan melakukan akses ke *method* `deleteById()` milik `RolesRepo`.

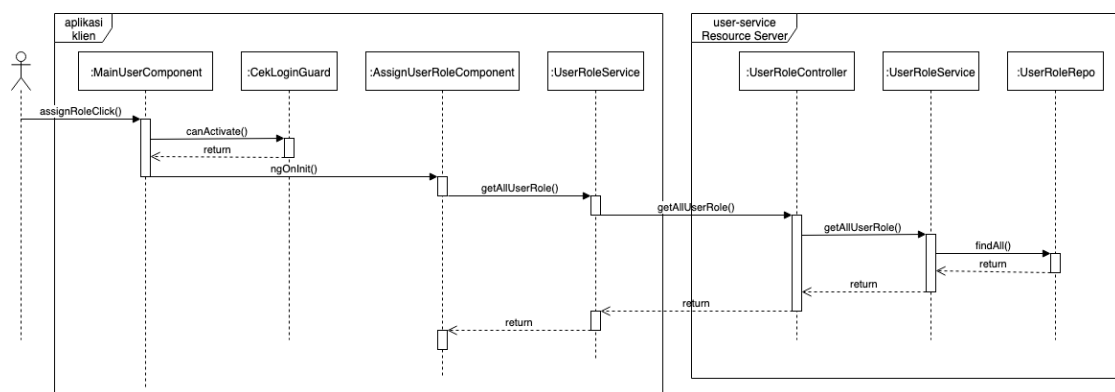
Di `RolesRepo` proses interaksi dengan sistem basis data terjadi, data pada sistem basis data akan dihapus dan proses berikutnya akan mengembalikan nilai ke `RolesService` yang kemudian diteruskan dari `RolesService` ke `RoleController`.

Nilai kembalian yang diterima di `RoleController` digunakan sebagai bahan *response* ke aplikasi klien yang kemudian diterima di `RoleService` karena sebelumnya melakukan *request*. Setelah itu dari `RoleService` kemudian meneruskan hasilnya ke `RemoveRoleProgressDialogComponent` dalam bentuk `Observable`.

Hasil yang diterima `RemoveRoleProgressDialogComponent` akan diteruskan dalam bentuk status ke `ListRoleComponent`, yang dengan nilai status ini kemudian `ListRoleComponent` melakukan pembaruan data pada tabel.

- Diagram *Sequence* Untuk Menampilkan Daftar Pemberian Hak Akses (*Role*)

Diagram *sequence* untuk menampilkan pemberian hak akses adalah seperti pada gambar berikut :



Gambar 69: Diagram *Sequence* Untuk Menampilkan Daftar Pemberian Hak Akses

Penjelasan dari diagram tersebut adalah sebagai berikut, pertama pengguna melakukan akses dari `MainUserComponent` ke menu Pemberian Hak Akses,

menu ini akan melakukan akses ke fungsi `assignRoleClick()` yang di dalamnya akan memicu pemanggilan fungsi `canActivate()` milik `CekLoginGuard`, untuk memastikan bahwa pengguna (*resource owner*) memiliki kewenangan untuk melakukan akses ke fitur tersebut.

Setelah pengguna dinyatakan berhak atas akses tersebut, maka halaman akan dialihkan ke `AssignUserRoleComponent` yang di dalamnya akan melakukan akses ke fungsi `getAllUserRole()` milik `UserRoleService`.

Dari dalam fungsi `getAllUserRole()` kelas `UserRoleService` melakukan *request* ke *resource server* melalui *method* `getAllUserRole()` milik kelas `UserRoleController`, yang kemudian di dalam *method* `getAllUserRole()` ditindak lanjuti dengan akses ke *method* `getAllUserRole()` milik `UserRoleService`, yang di dalamnya melakukan akses ke *method* `findAll()` milik `UserRoleRepo`. Pada tahap ini akses ke sistem basis data terjadi untuk mengambil daftar pemberian hak akses yang tercatat.

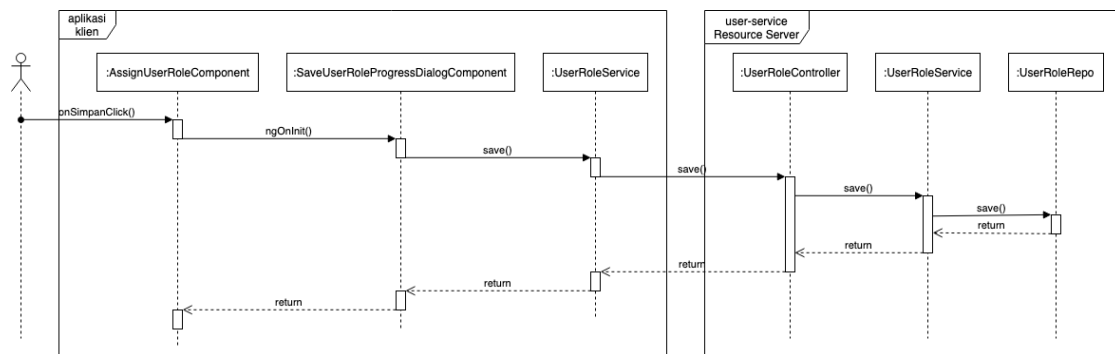
Hasil dari pengambilan data pada sistem basis data oleh `UserRoleRepo` akan mengembalikan nilai berupa `List<UserRole>` ke `UserRoleService` yang kemudian diteruskan ke `UserRoleController` dalam bentuk `Flux<UserRole>`.

Pada tahap ini `UserRoleController` akan mengirimkan sebuah *response* ke aplikasi klien berupa objek `Flux<UserRole>` yang kemudian diterima oleh `UserRoleService` yang sebelumnya melakukan *request*.

Hasil *response* dari *resource server* kemudian dikirimkan oleh `UserRoleService` ke `AssignUserRoleComponent` dalam bentuk `Observable` yang kemudian digunakan untuk mengisi tabel daftar pemberian hak akses.

- Diagram *Sequence* Untuk Menambahkan Pemberian Hak Akses (*Role*)

Diagram *sequence* untuk menambah pemberian hak akses adalah seperti pada gambar berikut :



Gambar 70: Diagram *Sequence* Untuk Menambah Pemberian Hak Akses

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna melakukan proses simpan data melalui fungsi `onSimpanClick()` milik `AssignUserRoleComponent`, yang di dalamnya kemudian memanggil `SaveUserRoleProgressDialogComponent`.

Proses inisiasi yang terjadi pada `SaveUserRoleProgressDialogComponent` ketika dipanggil akan melakukan akses ke fungsi `save` milik `UserRoleService`, yang kemudian dari fungsi `save` ini, `UserRoleService` melakukan *request* ke *resource server* melalui *method* `save` milik `UserRoleController`.

Di dalam *method* `save` milik `UserRoleController` kemudian melakukan akses ke *method* `save` milik `UserRoleService`, yang di dalam *method* ini akan melakukan akses ke *method* `save` milik `UserRoleRepo`. Sampai sini proses interaksi dengan sistem basis data terjadi, data yang disertakan akan tersimpan dalam sistem basis data.

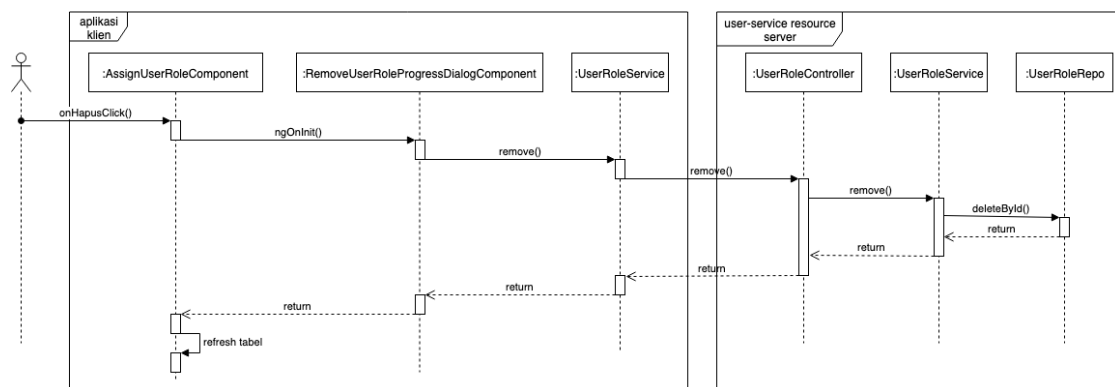
Setelah proses simpan data berhasil tercatat dalam sistem basis data, `UserRoleRepo` akan mengembalikan nilai dalam bentuk objek `UserRole` ke `UserRoleService` yang kemudian hasilnya diteruskan ke `UserRoleController`.

Setelah `UserRoleController` menerima data dari `UserRoleService`, `UserRoleController` akan mengirimkan *response* ke aplikasi klien, dalam hal ini ke `UserRoleService` karena sebelumnya melakukan *request* simpan data.

Dari `UserRoleService` kemudian hasil *response* akan diteruskan ke `SaveUserRoleProgressDialogComponent` dalam bentuk `Observable` yang kemudian dari `SaveUserRoleProgressDialogComponent` akan diteruskan ke `AssignUserRoleComponent` untuk melakukan pembaruan data pada tabel.

- Diagram *Sequence* Untuk Menghapus Pemberian Hak Akses (*Role*)

Diagram *sequence* untuk menghapus pemberian hak akses adalah seperti pada gambar berikut ini :



Gambar 71: Diagram *Sequence* Untuk Menghapus Pemberian Hak Akses

Penjelasan untuk diagram tersebut adalah seperti berikut, pertama pengguna (*resource owner*) melakukan akses ke hapus data melalui fungsi `onHapusClick()` milik `AssignUserRoleComponent`, di dalamnya akan memanggil `RemoveUserRoleProgressDialogComponent` untuk memulai proses hapus data pemberian hak akses.

Pada saat inisiasi `RemoveUserRoleProgressDialogComponent`, di dalamnya akan melakukan akses ke fungsi `remove` milik `UserRoleService`, dari dalam fungsi `remove` milik `UserRoleService` kemudian melakukan *request* ke *resource server* melalui *method* `remove()` milik `UserRoleController`.

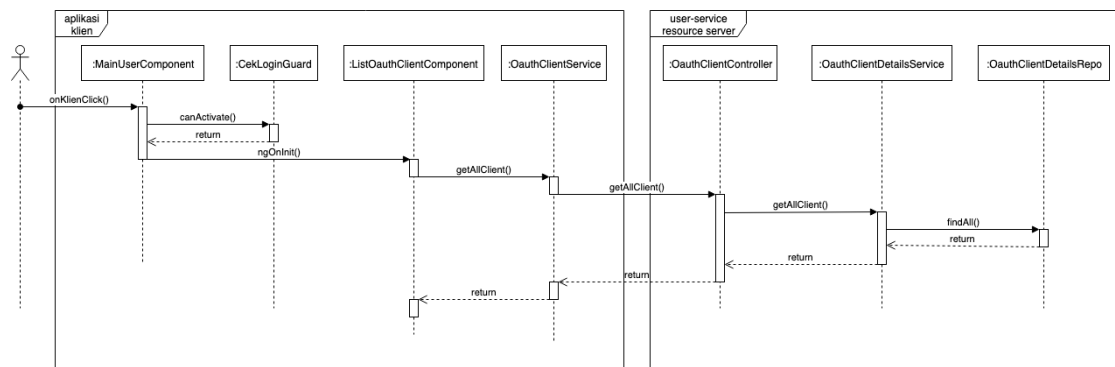
Lebih dalam, pada *method* `remove()` milik `UserRoleController` akan melakukan akses ke *method* `remove()` milik `UserRoleService`, lebih ke dalam lagi, dari `UserRoleService` akan melakukan akses ke *method* `remove` milik `UserRoleRepo`. Di sini interaksi dengan sistem basis data terjadi, data akan dihapuskan dari sistem basis data.

Proses selanjutnya `UserRoleRepo` akan mengembalikan sebuah nilai dalam bentuk objek `Unit` yang sebetulnya nilai kosong ke `UserRoleService`, yang kemudian diteruskan ke `UserRoleController`.

Kemudian `UserRoleController` akan memberikan *response* ke aplikasi klien, melalui `UserRoleService` karena sebelumnya melakukan *request*. Dari `UserRoleService`, data akan diteruskan ke `RemoveUserRoleProgressDialogComponent` dalam bentuk `Observable`, yang kemudian diolah dan diteruskan ke `AssignUserRoleComponent` dalam bentuk status, yang pada akhirnya statusnya digunakan untuk memicu pembaruan data pada tabel.

- Diagram *Sequence* Untuk Menampilkan Daftar Aplikasi Klien (*User Agent*)

Diagram *sequence* untuk menampilkan daftar aplikasi klien adalah seperti pada gambar berikut :



Gambar 72: Diagram *Sequence* Untuk Menampilkan Daftar Aplikasi Klien

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna (*resource owner*) melakukan akses ke fitur Daftar Aplikasi Klien dengan cara melakukan akses melalui fungsi `onKlienClick()` milik `MainUserComponent`, yang kemudian akan memicu fungsi `canActivate()` milik `CekLoginGuard` tereksekusi untuk memeriksa apakah pengguna yang sedang melakukan akses berada dalam kewenangannya untuk melakukan hal tersebut.

Bila kewenangannya memang memiliki hak akses untuk fitur tersebut, maka proses berikutnya adalah memanggil `ListOAuthClientComponent`, yang pada tahap inisiasinya akan melakukan akses ke fungsi `getAllClient()` milik `OAuthClientService`.

Dari `OAuthClientService` kemudian melakukan *request* ke *resource server* melalui *method* `getAllClient()` milik `OAuthClientController`. Dari `OAuthClientController` kemudian melakukan akses ke *method* `getAllClient()` milik `OAuthClientDetailsService`, selanjutnya `OAuthClientDetailsService` melakukan akses ke *method* `findAll()` milik `OAuthClientDetailsRepo`. Sampai sini proses pengambilan seluruh data aplikasi klien terjadi pada lapisan sistem basis data.

Setelah pengambilan data dari sistem basis data berhasil, ke-

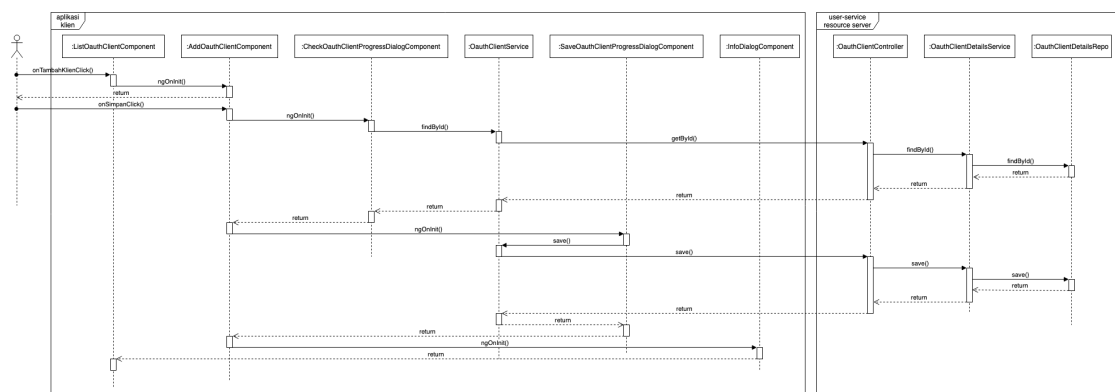
mudian nilai akan dikembalikan dari `OAuthClientDetailsRepo` ke `OAuthClientDetailsService` dalam bentuk objek `List<OAuthClientDetails>`, yang kemudian diteruskan oleh `OAuthClientDetailsService` ke `OAuthClientController`.

Dari `OAuthClientController` kemudian melakukan *response* ke aplikasi klien melalui `OAuthClientService` karena sebelumnya melakukan *request*, nilai yang diberikan ke aplikasi klien berbentuk objek `Flux<OAuthClientDetails>`.

Dari `OAuthClientService`, hasil *response* diteruskan ke `ListOAuthClientComponent` dalam bentuk `Observable` yang kemudian menjadi bahan data untuk mengisi tabel daftar aplikasi klien.

- Diagram *Sequence* Untuk Menambahkan Data Aplikasi Klien (*User Agent*)

Diagram *sequence* untuk menambahkan data aplikasi klien adalah seperti pada gambar berikut :



Gambar 73: Diagram *Sequence* Untuk Menambahkan Daftar Aplikasi Klien

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna (*resource owner*) melakukan akses ke fitur tambah aplikasi klien melalui fungsi `onTambahKlienClick()` milik `ListOAuthClientComponent`. Dari fungsi

ini kemudian melakukan panggilan ke `AddOAuthClientComponent` sehingga pengguna (*resource owner*) dapat melakukan pengisian data aplikasi klien.

Setelah pengguna (*resource owner*) melakukan pengisian data aplikasi klien kemudian menyimpannya, maka fungsi `onSimpanClick()` pada `AddOAuthClientComponent` akan terpicu, kemudian memanggil `CheckOAuthClientProgressDialogComponent` untuk melakukan pemeriksaan data apakah aplikasi klien yang akan disimpan sudah pernah disimpan sebelumnya.

Dari `CheckOAuthClientProgressDialogComponent` kemudian melakukan akses ke fungsi `findById()` milik `OAuthClientService`. Dari `OAuthClientService` kemudian melakukan *request* ke *resource server* melalui *method* `findById()` milik `OAuthClientController`.

Dari `OAuthClientController` kemudian melakukan akses ke *method* `findById()` milik `OAuthClientDetailsService` yang di dalamnya melakukan akses ke *method* `findById()` milik `OAuthClientDetailsRepo`. Di sini proses pencarian data pada sistem basis data terjadi.

Setelah proses pencarian data pada sistem basis data selesai, maka `OAuthClientDetailsRepo` akan mengembalikan nilai dalam bentuk objek `Optional<OAuthClientDetails>` ke `OAuthClientDetailsService`, yang nilainya kemudian diteruskan ke `OAuthClientController`.

Kemudian `OAuthClientController` akan melakukan *response* ke aplikasi klien melalui `OAuthClientService` karena sebelumnya melakukan *request*. Dari `OAuthClientService` kemudian hasil *response* diteruskan ke `CheckOAuthClientProgressDialogComponent` dalam bentuk `Observable` dan diteruskan ke `AddOAuthClientComponent` dalam bentuk status.

Bila status menunjukkan data yang akan disimpan belum pernah terca-

tat dalam sistem basis data, maka proses akan berlanjut dengan memanggil `SaveOAuthClientProgressDialogComponent`, yang di dalamnya akan melakukan akses ke fungsi `save()` milik `OAuthClientService`. Dari `OAuthClientService` kemudian melakukan *request* kembali ke *resource server*, kali ini melalui *method* `save()` milik `OAuthClientController`.

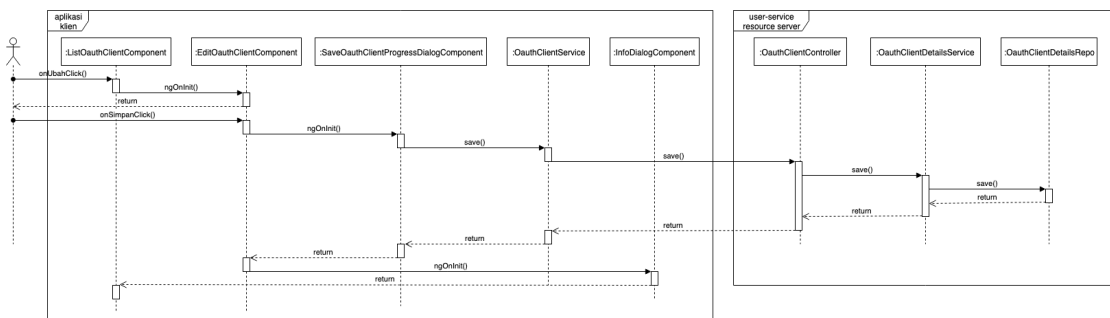
Dari `OAuthClientController` kemudian melakukan akses ke *method* `save()` milik `OAuthClientDetailsService` yang di dalamnya melakukan akses ke *method* `save()` milik `OAuthClientDetailsRepo`. Sampai sini proses simpan data ke sistem basis data terjadi.

Setelah data tersimpan pada sistem basis data, `OAuthClientDetailsRepo` akan mengembalikan nilai dalam bentuk objek `OAuthClientDetails` ke `OAuthClientDetailsService`, yang kemudian diteruskan ke `OAuthClientController`.

Sampai tahap ini `OAuthClientController` akan memberikan *response* ke aplikasi klien melalui `OAuthClientService` karena sebelumnya melakukan *request*. Dari `OAuthClientService`, *response* akan diteruskan ke `SaveOAuthClientProgressDialogComponent` dalam bentuk `Observable`. Dari `SaveOAuthClientProgressDialogComponent` meneruskan pesan ini dalam bentuk status ke `AddOAuthClientComponent`, hasilnya kemudian akan ditampilkan ke pengguna dengan memanggil `InfoDialogComponent` yang pada akhirnya akan melakukan pembaruan data tabel pada `ListOAuthClientComponent`.

- Diagram *Sequence* Untuk Mengubah Data Aplikasi Klien (*User Agent*)

Diagram *sequence* untuk mengubah data aplikasi klien adalah seperti pada gambar berikut :



Gambar 74: Diagram *Sequence* Untuk Mengubah Data Aplikasi Klien

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna melakukan akses ke fitur ubah data melalui fungsi `onUbahClick()` milik `ListOAuthClientComponent` yang di dalamnya memanggil `EditOAuthClientComponent`, sampai sini pengguna (**resource owner**) dapat mengubah rincian data dari aplikasi klien.

Setelah pengguna (*resource owner*) selesai mengubah beberapa rincian data untuk aplikasi klien, kemudian pengguna (*resource owner*) menyimpan perubahan tersebut dengan melakukan akses ke fungsi `onSimpanClick()` milik `EditOAuthClientComponent`. Di dalam `EditOAuthClientComponent` kemudian memanggil `SaveOAuthClientProgressDialogComponent` untuk memulai proses simpan data dan pengguna (*resource owner*) mengetahui bahwa proses tersebut berjalan.

Dari `SaveOAuthClientProgressDialogComponent` kemudian melakukan akses ke fungsi `save` milik `OAuthClientService`, setelah itu `OAuthClientService` melakukan *request* data ke *resource server* melalui *method* `save()` milik `OAuthClientController`.

Atas dasar *request* yang datang, `OuathClientController` melakukan akses ke *method* `save()` milik `OauthClientDetailsService`, yang di dalamnya

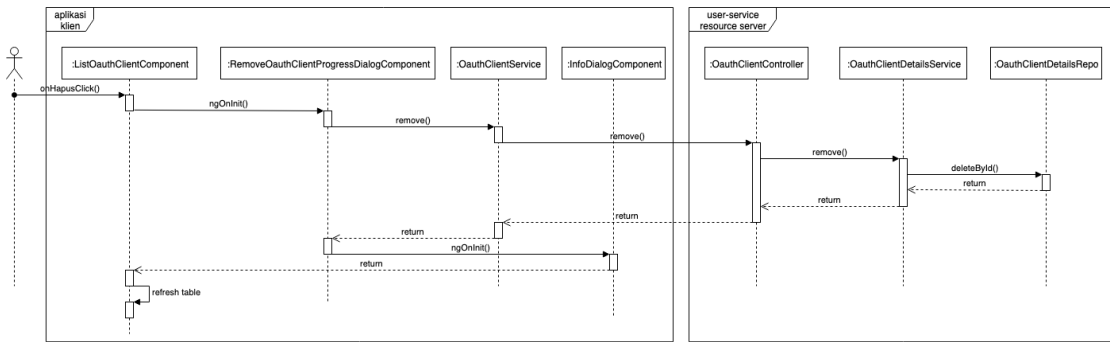
akan melakukan akses ke *method* `save()` milik `OAuthClientDetailsRepo`. Di sini proses simpan data ke dalam sistem basis data terjadi.

Setelah proses simpan data ke sistem basis data berhasil dilakukan, kemudian `OAuthClientDetailsRepo` akan mengembalikan sebuah nilai berbentuk objek dari `OAuthClientDetails` ke `OAuthClientDetailsService`, yang kemudian diteruskan ke `OAuthClientController`. Dari `OAuthClientController` kemudian memberikan *response* ke aplikasi klien melalui `OAuthClientService` karena sebelumnya melakukan *request* simpan data.

Dari `OAuthClientService` kemudian meneruskan hasil *response* dari *resource server* ke `SaveOAuthClientProgressDialogComponent` dalam bentuk `Observable`, yang kemudian diteruskan oleh `SaveOAuthClientProgressDialogComponent` ke `EditOAuthClientComponent` dalam bentuk sebuah status, dengan dasar status ini, `EditOAuthClientComponent` akan memanggil `InfoDialogComponent` untuk memberikan informasi kepada pengguna (*resource owner*) apakah proses simpan perubahan datanya berhasil atau gagal, yang pada akhirnya akan memicu `ListOAuthClientController` untuk melakukan pembaruan pada tabel daftar aplikasi klien.

- Diagram *Sequence* Untuk Menghapus Data Aplikasi Klien (*User Agent*)

Diagram *sequence* untuk menghapus data aplikasi klien dari sistem otentikasi adalah seperti pada gambar berikut :



Gambar 75: Diagram *Sequence* Untuk Menghapus Data Aplikasi Klien

Penjelasan dari diagram tersebut adalah seperti berikut, pertama pengguna melakukan akses ke fitur hapus data aplikasi klien melalui fungsi `onHapusClick()` milik `ListOAuthClientComponent`, kemudian dari `ListOAuthClientComponent` melakukan pemanggilan ke `RemoveOAuthClientProgressDialogComponent` untuk menginformasikan ke pengguna (*resource owner*) bahwa proses hapus data aplikasi klien sedang berjalan.

Dari `RemoveOAuthClientProgressDialogComponent`, kemudian melakukan akses ke fungsi `remove()` milik `OAuthClientService` yang di dalamnya akan melakukan *request* ke *resource server* melalui *method* `remove()` milik `OAuthClientController`.

Kemudian proses berlanjut, setelah `OAuthClientController` menerima *request* dari aplikasi klien, kemudian `OAuthClientController` melakukan akses ke *method* `remove` milik `OAuthClientDetailsService` yang di dalamnya melakukan akses ke *method* `deleteById()` milik `OAuthClientDetailsRepo`. Pada tahap ini proses penghapusan pada sistem basis data berjalan.

Setelah proses penghapusan data pada sistem basis data selesai, kemudian `OAuthClientDetailsRepo` akan mengembalikan nilai berbentuk objek

Unit yang sebetulnya tidak ada isinya ke `OAuthClientDetailsService`, yang kemudian dari `OAuthClientDetailsService` hasilnya akan diteruskan ke `OAuthClientController`.

Dari `OAuthClientController` kemudian memberikan *response* ke aplikasi klien melalui `OAuthClientService` karena sebelumnya melakukan *request* hapus data aplikasi klien, dari *response* tersebut, `OAuthClientService` akan meneruskan hasilnya ke `RemoveOAuthClientProgressDialogComponent` dalam bentuk `Observable`, yang kemudian akan memanggil `InfoDialogComponent` untuk memberikan informasi apakah datanya berhasil dihapus atau gagal ke pengguna (*resource owner*).

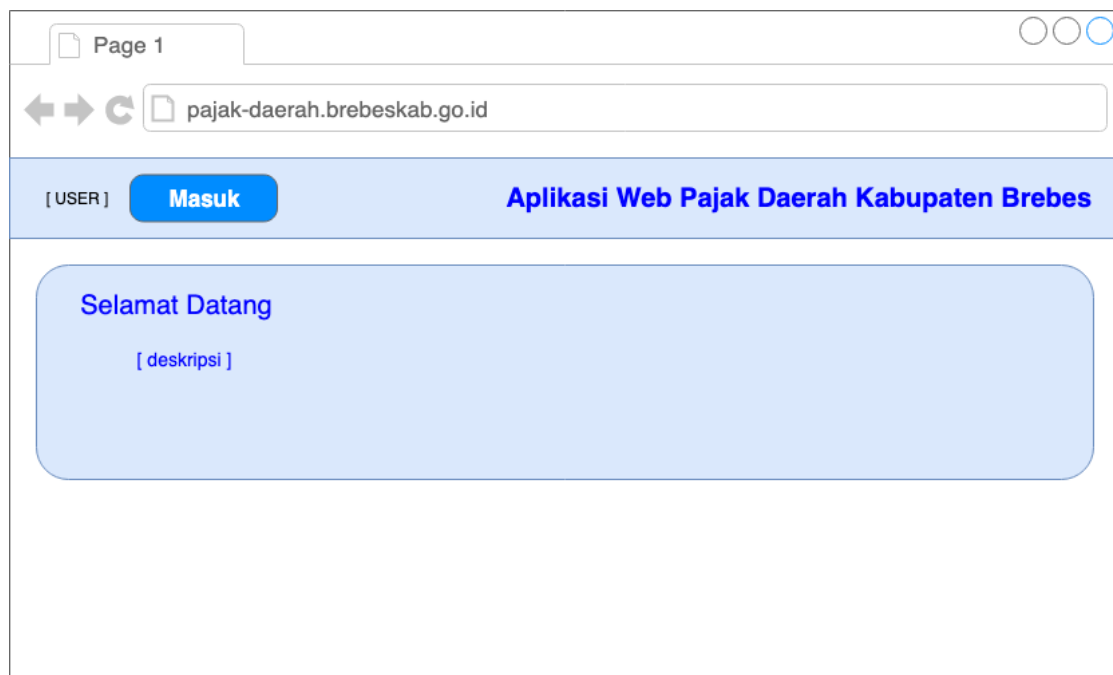
Setelah informasinya berhasil diterima pengguna, kemudian `InfoDialogComponent` akan memicu perubahan isi tabel daftar aplikasi klien pada `ListOAuthClientComponent`.

5.3 Bagian Tampilan Aplikasi

Bagian aplikasi nantinya akan terbagi sesuai dengan banyaknya fitur yang disediakan, berikut adalah daftar desain tampilan yang akan ada dalam aplikasi yang dibangun :

- Tampilan Jendela Utama

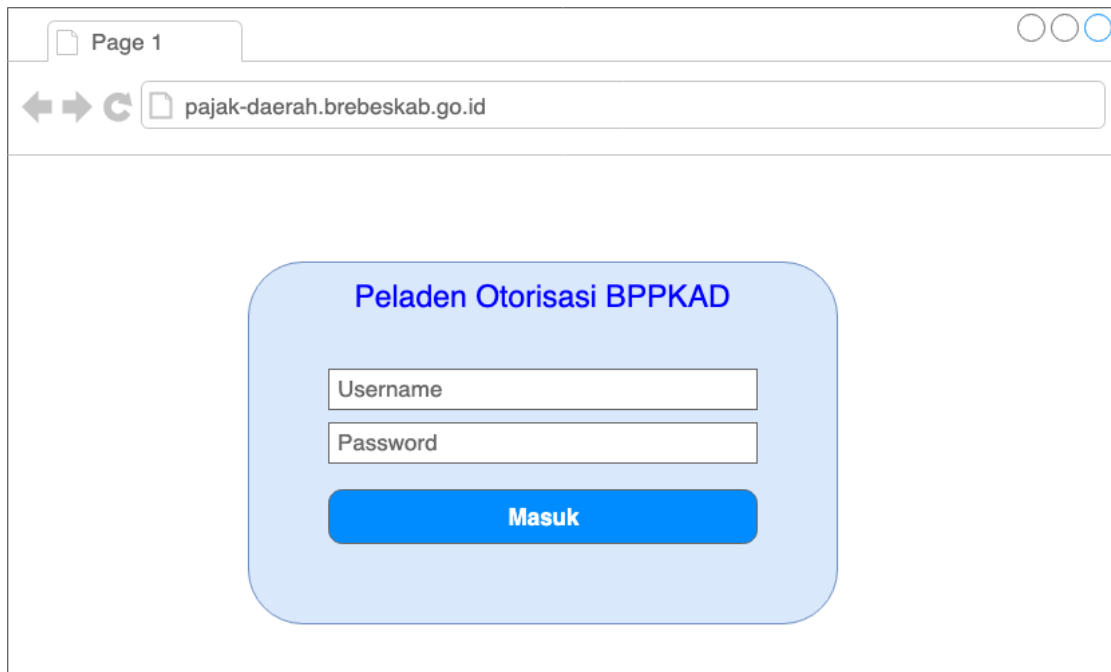
Jendela utama ini akan terlihat pada saat pengguna (*resource owner*) melakukan akses pertama kali ke aplikasi. Tampilannya akan terlihat seperti pada gambar berikut :



Gambar 76: Tampilan Jendela Utama

- Tampilan Halaman *Login*

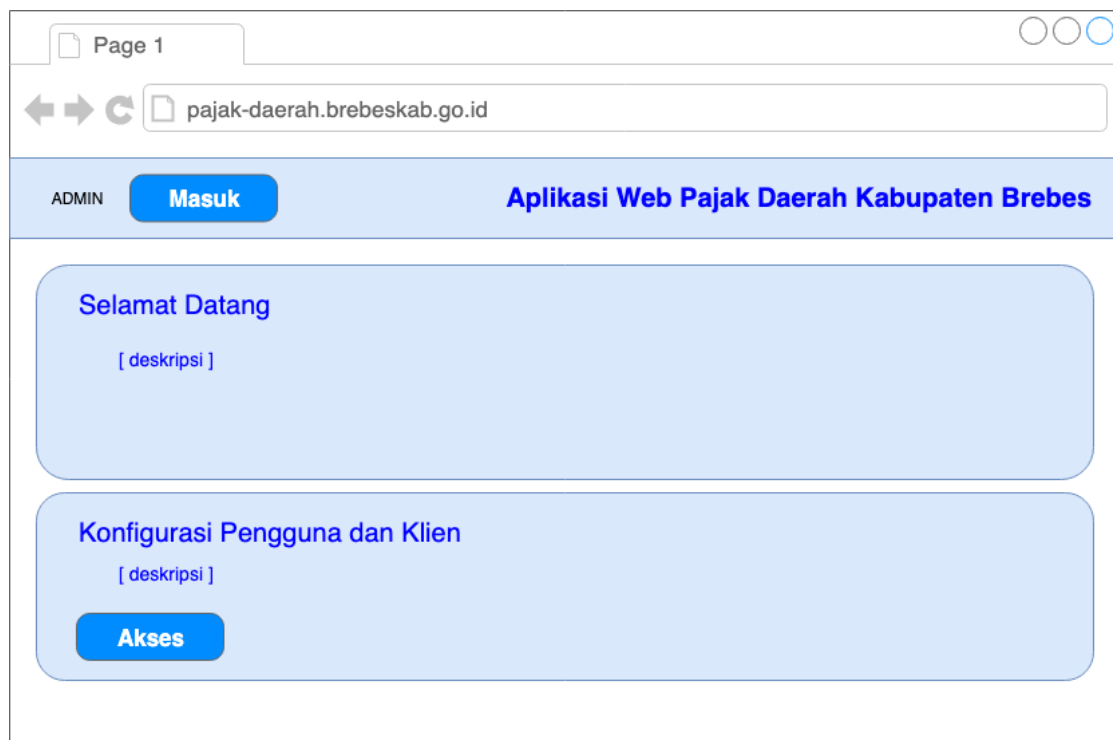
Pada saat pengguna (*resource owner*) ingin melakukan akses ke informasi yang sensitif, maka halaman ini akan muncul terlebih dahulu untuk melakukan otentikasi pengguna, tampilannya akan terlihat seperti pada gambar berikut :



Gambar 77: Tampilan Halaman *Login*

- Tampilan Jendela Administrator

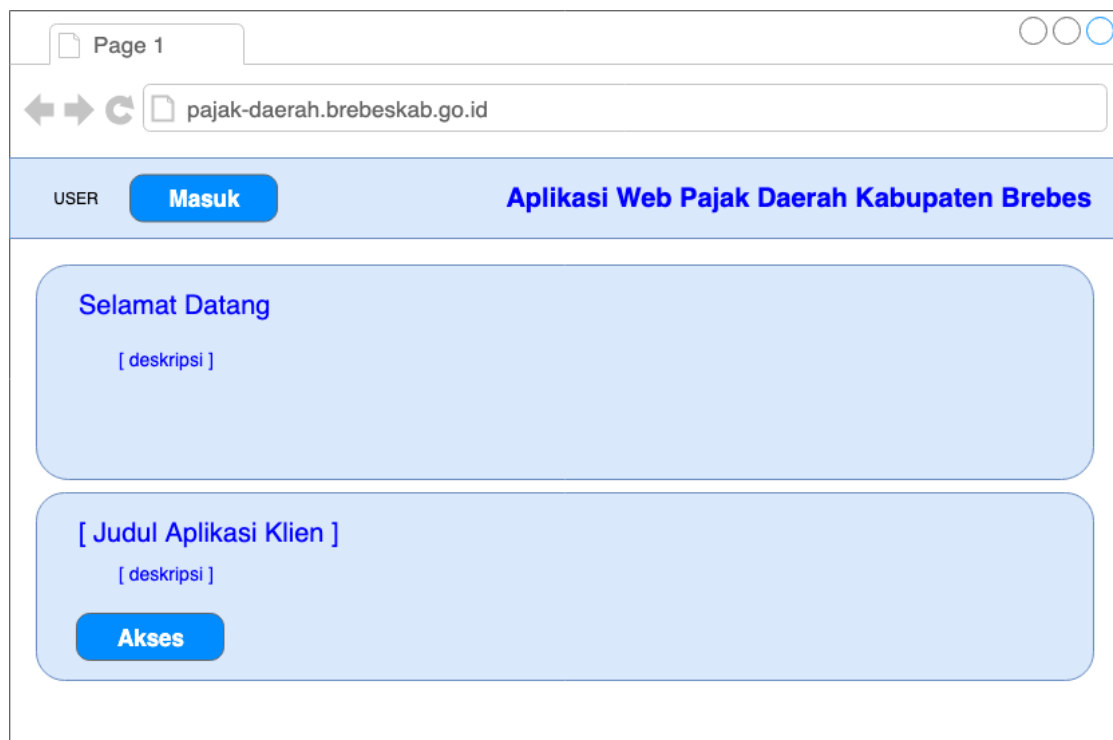
Jendela ini akan muncul setelah pengguna (*resource owner*) melakukan otentikasi di halaman *login* dan memiliki hak akses sebagai administrator. Tampilannya akan terlihat seperti gambar berikut :



Gambar 78: Tampilan Jendela Administrator

- Tampilan Jendela Pengguna Pada Umumnya

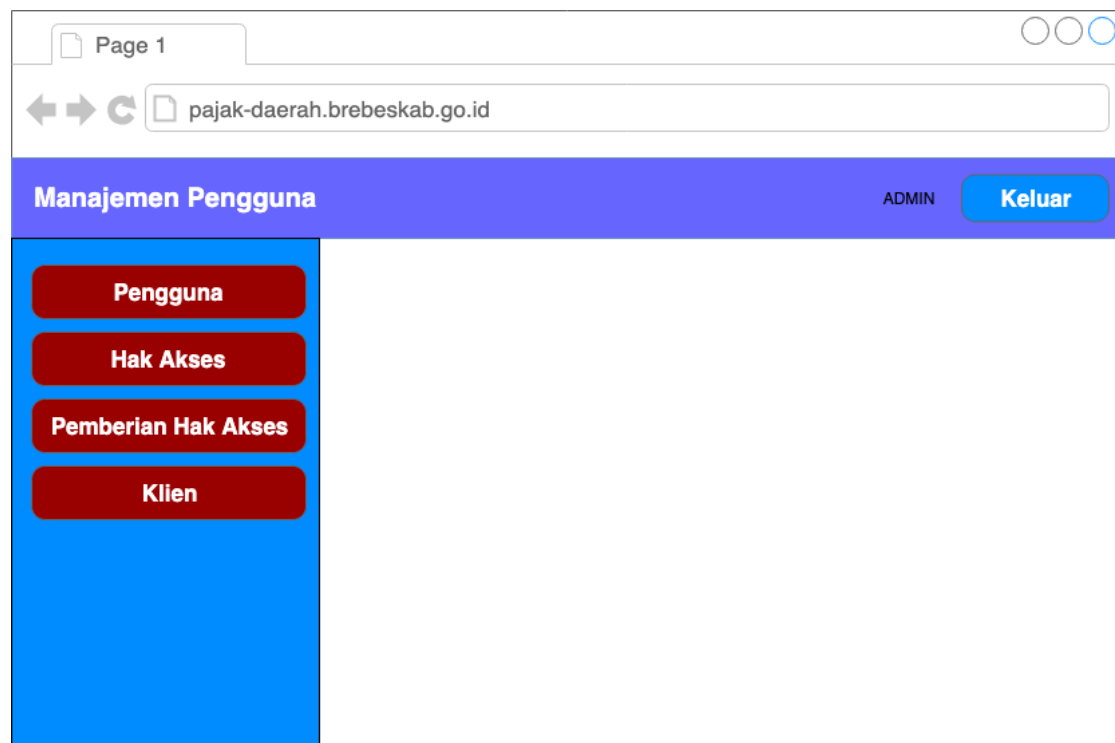
Jendela ini akan muncul setelah pengguna (*resource owner*) melakukan otentikasi di halaman *login* dan memiliki hak akses sebagai pengguna biasa. Tampilannya akan terlihat seperti pada gambar berikut :



Gambar 79: Tampilan Jendela Pengguna Umum

- Tampilan Jendela Manajemen Pengguna

Jendela ini hanya dapat diakses oleh pengguna dengan kewenangan administrator, yang fungsinya untuk melakukan manajemen pengguna dan aplikasi klien, tampilannya akan terlihat seperti pada gambar berikut :

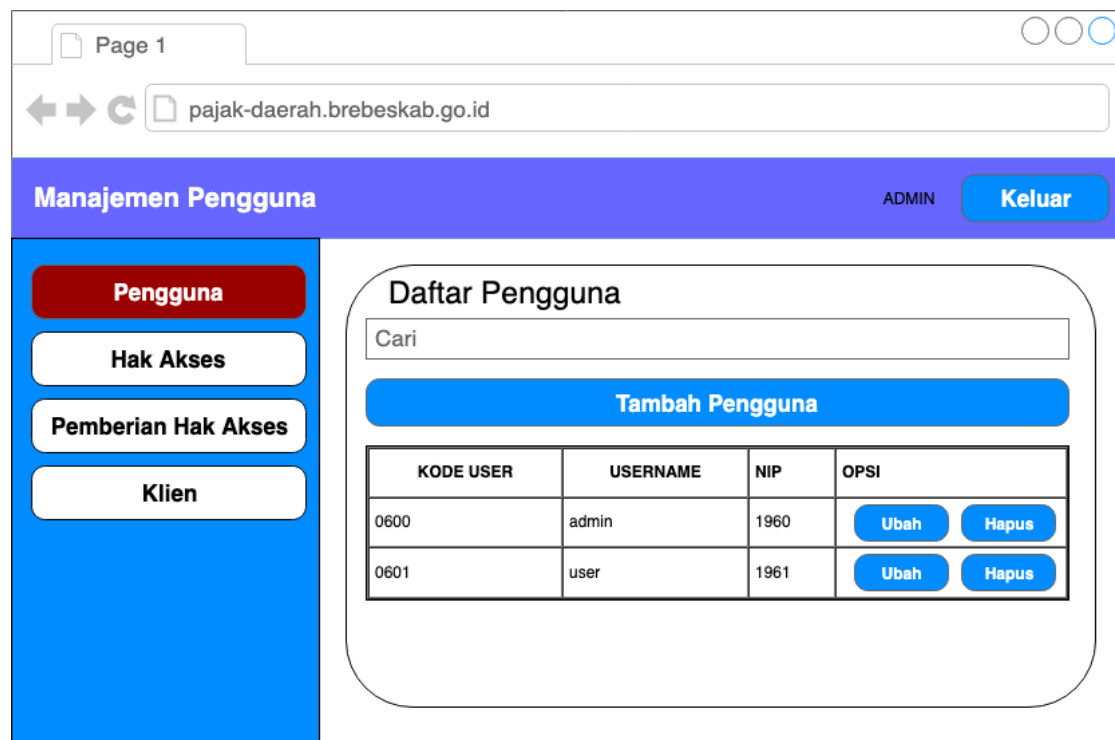


Gambar 80: Tampilan Jendela Manajemen Pengguna

- Tampilan Jendela Daftar Pengguna

Jendela ini digunakan untuk melihat seluruh daftar pengguna yang terdaftar.

Tampilannya akan terlihat seperti pada gambar berikut :



Gambar 81: Tampilan Jendela Daftar Pengguna

- Tampilan Jendela Tambah Data Pengguna Baru

Jendela ini digunakan oleh administrator untuk melakukan penambahan data pengguna baru pada sistem otentikasi, tampilannya akan terlihat seperti berikut :

Page 1

pajak-daerah.brebeskab.go.id

Manajemen Pengguna ADMIN **Keluar**

Pengguna

Hak Akses

Pemberian Hak Akses

Klien

Tambah Pengguna Baru

Nama Pengguna

Kata Kunci

Status Aktif ☐ Tidak Aktif

Kode Pengguna

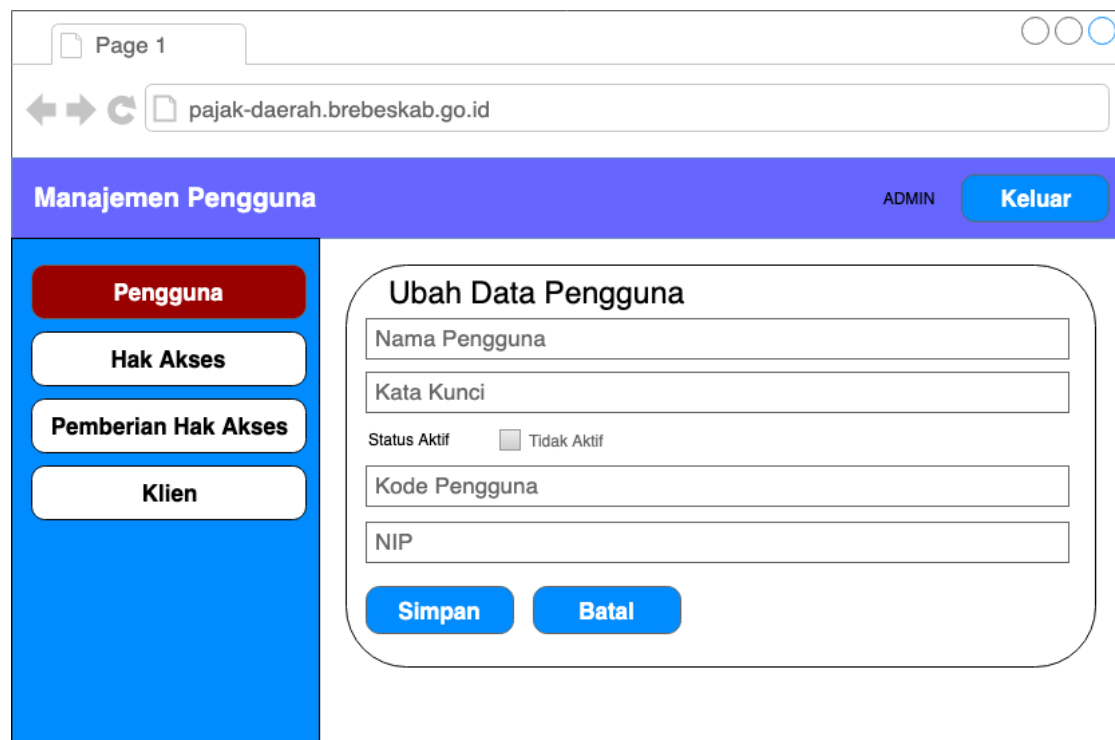
NIP

Simpan **Batal**

Gambar 82: Tampilan Jendela Tambah Data Pengguna Baru

- Tampilan Jendela Ubah Data Pengguna

Jendela ini digunakan oleh administrator untuk melakukan perubahan data pengguna yang telah terdaftar sebelumnya, tampilannya akan terlihat seperti pada gambar berikut :



Page 1

pajak-daerah.brebeskab.go.id

Manajemen Pengguna ADMIN **Keluar**

Pengguna

Hak Akses

Pemberian Hak Akses

Klien

Ubah Data Pengguna

Nama Pengguna

Kata Kunci

Status Aktif ☐ Tidak Aktif

Kode Pengguna

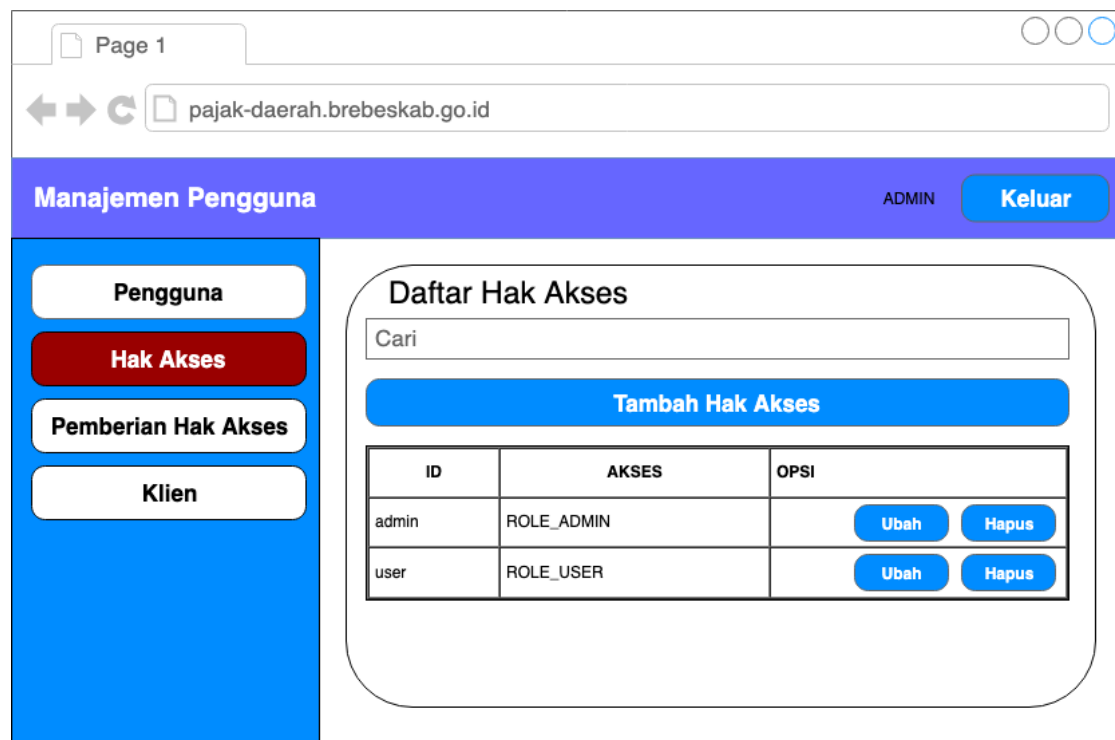
NIP

Simpan **Batal**

Gambar 83: Tampilan Jendela Ubah Data Pengguna

- Tampilan Jendela Daftar Hak Akses

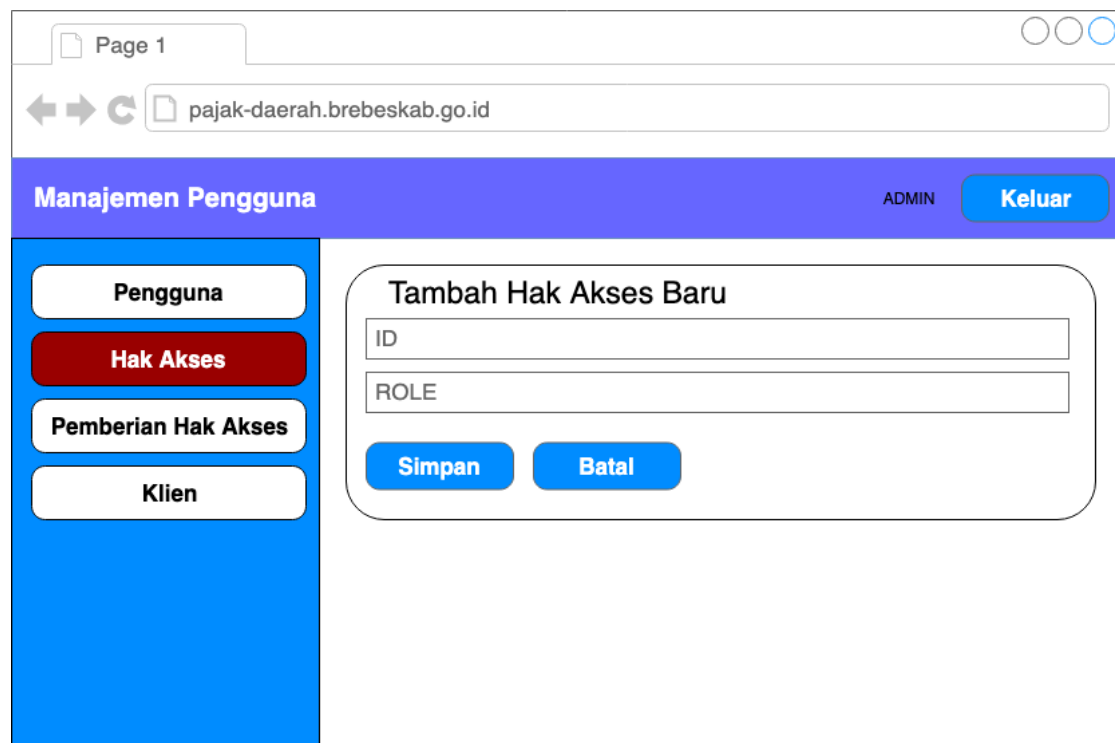
Jendela ini digunakan oleh administrator untuk menampilkan daftar hak akses. Tampilannya akan terlihat seperti gambar berikut :



Gambar 84: Tampilan Jendela Daftar Hak Akses

- Tampilan Jendela Tambah Hak Akses Baru

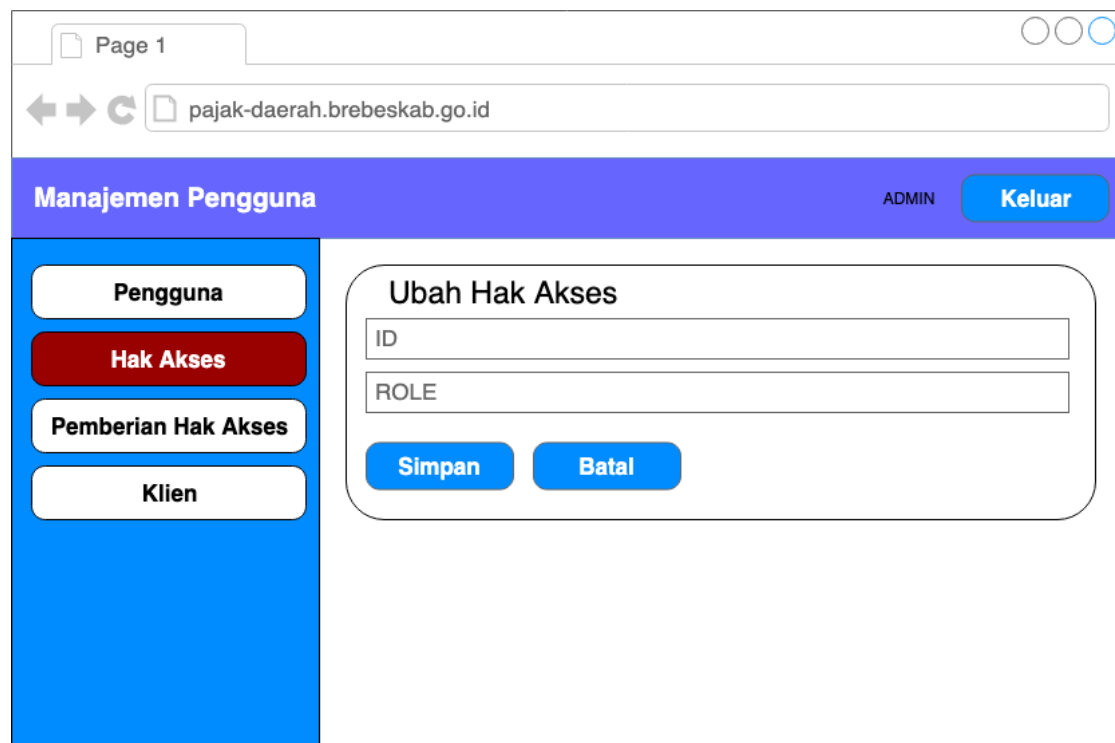
Jendela ini digunakan oleh administrator untuk menambahkan hak akses baru bila diperlukan, tampilannya akan terlihat seperti pada gambar berikut :



Gambar 85: Tampilan Jendela Tambah Hak Akses Baru

- Tampilan Jendela Ubah Hak Akses

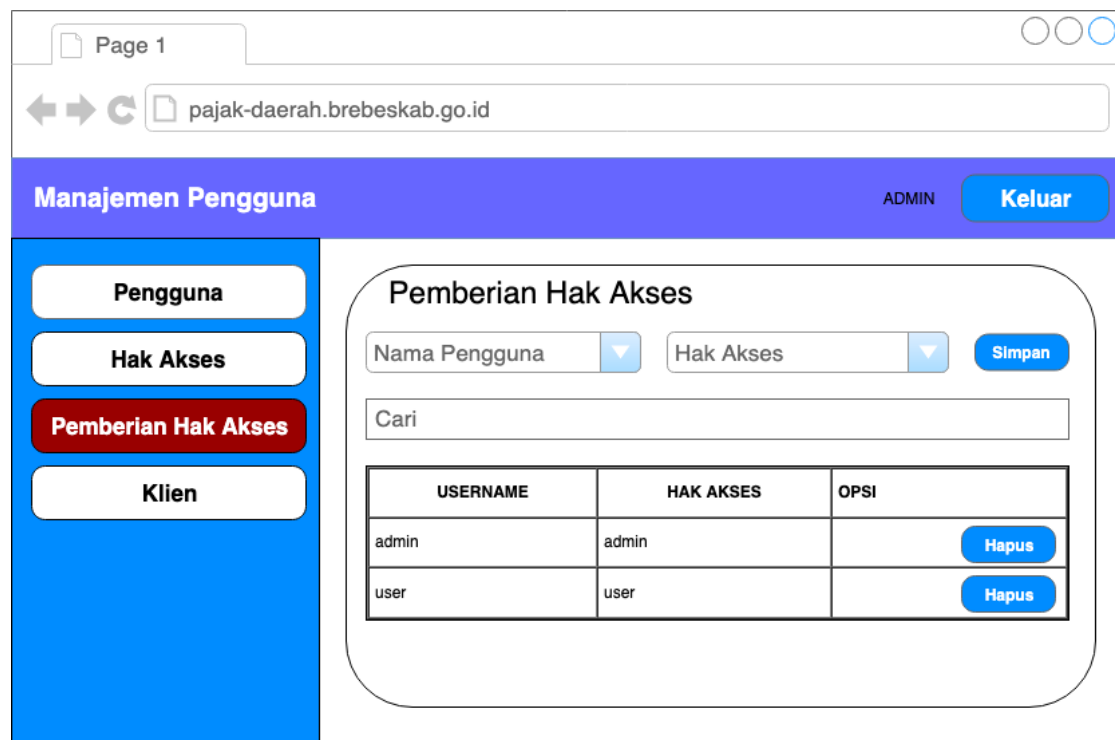
Jendela ini digunakan oleh administrator untuk mengubah isian hak akses yang mungkin terjadi karena kesalahan ketik, tampilannya akan terlihat seperti pada gambar berikut :



Gambar 86: Tampilan Jendela Ubah Hak Akses

- Tampilan Jendela Daftar Pemberian Hak Akses

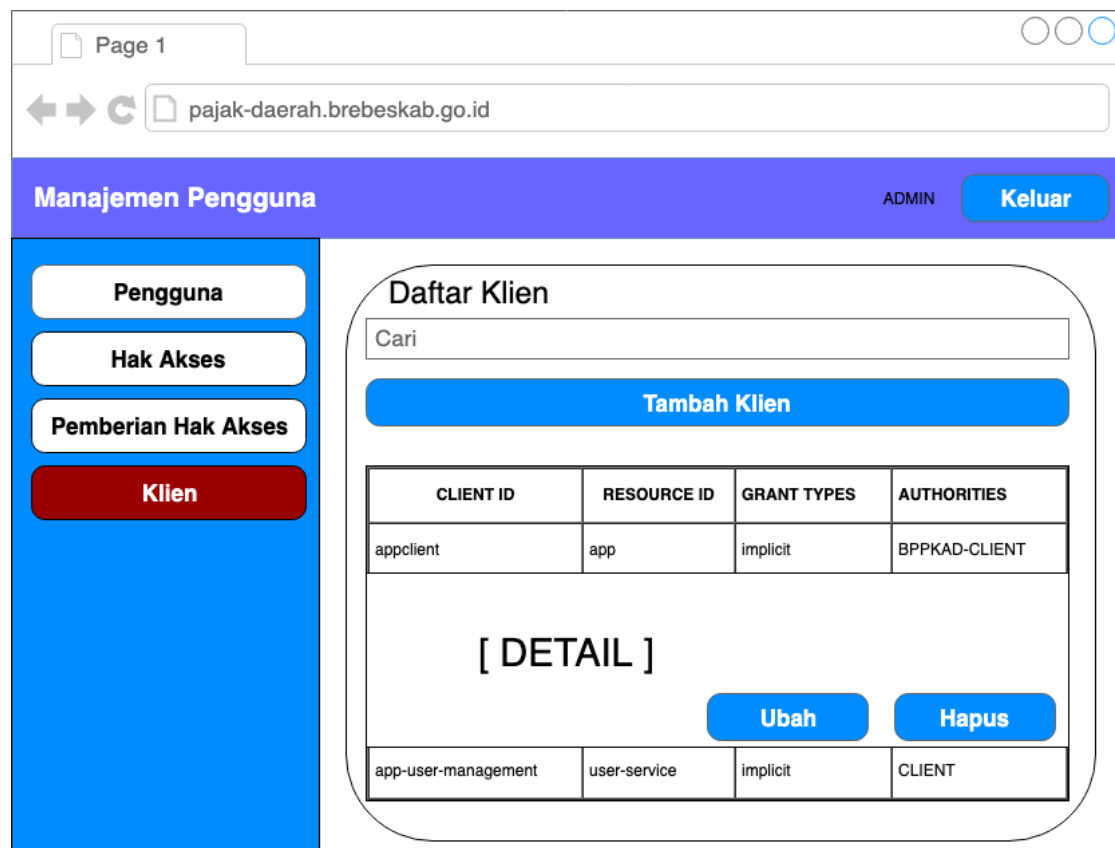
Jendela ini digunakan untuk menampilkan daftar pemberian hak akses pada tiap pengguna, pada jendela ini pun diberikan fitur untuk menambahkan pemberian hak akses dan menghapus pemberian hak akses, tampilannya akan terlihat seperti berikut :



Gambar 87: Tampilan Jendela Daftar Pemberian Hak Akses

- Tampilan Jendela Daftar Aplikasi Klien

Jendela ini digunakan untuk menampilkan daftar aplikasi klien (*user agent*) yang terdaftar pada sistem otentikasi, tampilannya akan terlihat seperti pada gambar berikut :



Gambar 88: Tampilan Jendela Daftar Aplikasi Klien

- Tampilan Jendela Tambah Aplikasi Klien Baru

Jendela ini digunakan untuk menampilkan formulir penambahan aplikasi klien ke sistem otentikasi, tampilannya akan terlihat seperti gambar berikut :

Page 1

pajak-daerah.brebeskab.go.id

Manajemen Pengguna ADMIN **Keluar**

Pengguna

Hak Akses

Pemberian Hak Akses

Klien

Tambah Klien OAuth Baru

Client ID

Resource ID

Client Secret

Scope

Grant Types

Web Server Redirect URI

Authorities

Access Token Validity

Refresh Token Validity

Informasi Tambahan

Simpan **Batal**

Gambar 89: Tampilan Jendela Tambah Aplikasi Klien Baru

- Tampilan Jendela Ubah Data Aplikasi Klien

Jendela ini digunakan untuk menampilkan formulir perubahan data dari aplikasi klien yang telah terdaftar, tampilannya akan terlihat seperti gambar berikut :

Gambar 90: Tampilan Jendela Ubah Data Aplikasi Klien

6 BIAYA DAN JADWAL PENGEMBANGAN

1. Biaya Pengembangan

Pengembangan aplikasi ini memerlukan beberapa perangkat dan kelengkapannya seperti berikut ini :

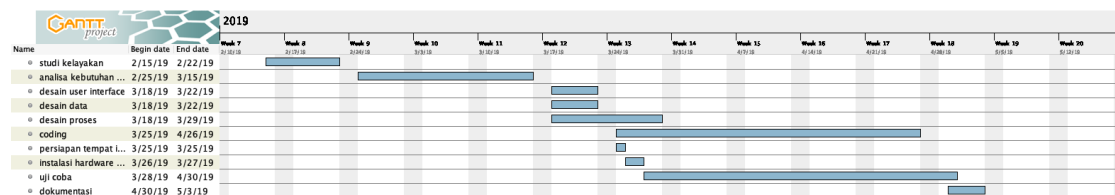
- (a) Peladen Sistem Basis Data
- (b) Peladen Aplikasi *Web*

- (c) Aplikasi Peladen *Servlet*, dalam hal ini menggunakan Apache Tomcat yang tersedia gratis.
- (d) Docker sebagai *container* tempat aplikasi dan layanan berada.
- (e) *Driver* JDBC yang tersedia gratis.
- (f) Akses Internet
- (g) IDE, menggunakan IntelliJ IDEA versi *Community Edition*

Melihat ketersediaan perangkat keras dan perangkat lunak tersebut di atas sudah ada, hanya tinggal digunakan dan beberapa perangkat lunak hanya tinggal di unduh secara bebas dan gratis, maka tidak ada biaya yang diperlukan untuk pengembangannya.

2. Jadwal Pengembangan

Jadwal pengembangan untuk membangun aplikasi atau sistem informasi ini tertuang pada diagram *gantt* seperti pada gambar 91 berikut ini :



Gambar 91: Diagram *Gantt* Untuk Jadwal Pengembangan Aplikasi