BAB III
ANALISIS DAN PERANCANGAN

1.1 Analisis

1.1.1 Identifikasi Masalah

Dengan mempertimbangkan informasi sebelumnya, masalah yang dapat diidentifikasi adalah sebagai berikut:

1. Penggunaan sistem yang masih menggunakan pencacatan manual untuk mendapatkan rekapan data harian, sehingga menyulitkan admin untuk mendapatkan data secara rinci dan lengkap.
2. Penggunaan arsitektur monolitik yang dimanfaatkan dirasa kurang efektif ketika transaksi dalam jumlah yang banyak dalam sekali waktu.
3. Penggunaan sistem yang masih belum *up-to-date* dengan perusahaan PLN sehingga ketika ada pengguna baru belum bisa integrasi dengan sistem.

1.1.2 Pemecahan Masalah

Berdasarkan uraian identifikasi masalah di atas, dapat dilakukan dengan menggunakan sistem *microservice* pada aplikasi saat ini. Adapun langkah-langkah penyelesaian masalah sebagai berikut:

1. Mengamati permasalahan yang dirasakan oleh admin dan karyawan.
2. Observasi untuk penyesuaian sistem yang akan digunakan menggunakan metode baru yang akan dirancangkan.

3. Melakukan identifikasi masalah, penentuan batasan masalah, tujuan beserta manfaat.
4. Mengumpulkan data maupun informasi sejelas-jelasnya.
5. Melakukan pengolahan data.
6. Melakukan analisis data dan perancangan sistem yang akan digunakan dan dibuat.
7. Mengimplementasikan rancangan sistem yang sudah dibuat dalam bentuk *prototype*, hingga menjadi sistem pengembangan lokal.
8. Menarik sebuah kesimpulan dan saran dari penelitian yang sudah dilakukan.

Menggunakan *microservices* dibandingkan dengan arsitektur monolitik menawarkan beberapa keuntungan yang signifikan, terutama dalam hal skalabilitas, fleksibilitas, dan pemeliharaan aplikasi. Berikut adalah beberapa alasan spesifik mengapa *microservices* yang dipilih:

1. Skalabilitas Terpisah

   Dalam arsitektur monolitik, seluruh aplikasi dikemas menjadi satu unit besar, sehingga sulit untuk menskalakan bagian tertentu dari aplikasi secara terpisah. Sebaliknya, dengan *microservices*, setiap layanan dapat diskalakan secara independen berdasarkan kebutuhan. Misalnya, jika modul pembayaran membutuhkan lebih banyak sumber daya, hanya layanan tersebut yang perlu diskalakan, tanpa harus memengaruhi layanan lainnya.
2. Fleksibilitas Teknologi

Microservices memungkinkan pengembang menggunakan teknologi yang berbeda untuk setiap layanan sesuai kebutuhan. Jika satu tim merasa lebih nyaman dengan Python untuk layanan tertentu dan Java untuk yang lain, mereka bisa melakukannya. Di sisi lain, arsitektur monolitik mengharuskan seluruh aplikasi menggunakan satu set teknologi yang sama, yang dapat membatasi inovasi dan fleksibilitas.

3. Pengembangan dan Penyebaran Mandiri

   Dalam arsitektur monolitik, perubahan pada satu bagian aplikasi sering memerlukan penerapan ulang seluruh aplikasi, yang bisa memperlambat siklus pengembangan dan mengakibatkan risiko *downtime*. Dengan *microservices*, setiap layanan dapat dikembangkan, diuji, dan diterapkan secara mandiri tanpa memengaruhi layanan lainnya. Ini memungkinkan siklus pengembangan yang lebih cepat dan penerapan perubahan yang lebih aman.

4. Ketahanan dan Isolasi Kegagalan

   Dalam arsitektur monolitik, jika satu bagian dari aplikasi gagal, seluruh aplikasi mungkin terpengaruh. Sebaliknya, dalam arsitektur *microservices*, kegagalan dalam satu layanan tidak akan secara langsung memengaruhi layanan lainnya. Misalnya, jika layanan pencarian gagal, layanan pembayaran tetap bisa berjalan, yang meningkatkan ketahanan aplikasi secara keseluruhan.

5. Pemeliharaan dan Pengembangan Berkelanjutan

Aplikasi monolitik cenderung menjadi lebih kompleks dan sulit untuk dipelihara seiring bertambahnya fitur dan ukuran aplikasi. Setiap perubahan atau penambahan fitur baru bisa menyebabkan masalah yang tak terduga di bagian lain dari aplikasi. *Microservices*, dengan layanan yang lebih kecil dan terpisah, lebih mudah untuk dipelihara dan diperbarui, sehingga mengurangi risiko kesalahan dan mempercepat pengembangan berkelanjutan.

6. Tim yang Terdesentralisasi

   Dengan arsitektur *microservices*, tim dapat bekerja secara lebih independen pada layanan masing-masing. Ini memungkinkan organisasi untuk membentuk tim yang terdesentralisasi, di mana setiap tim bertanggung jawab atas pengembangan dan operasi satu atau beberapa *microservices*. Ini mempercepat waktu pengembangan dan mempermudah koordinasi antar tim.

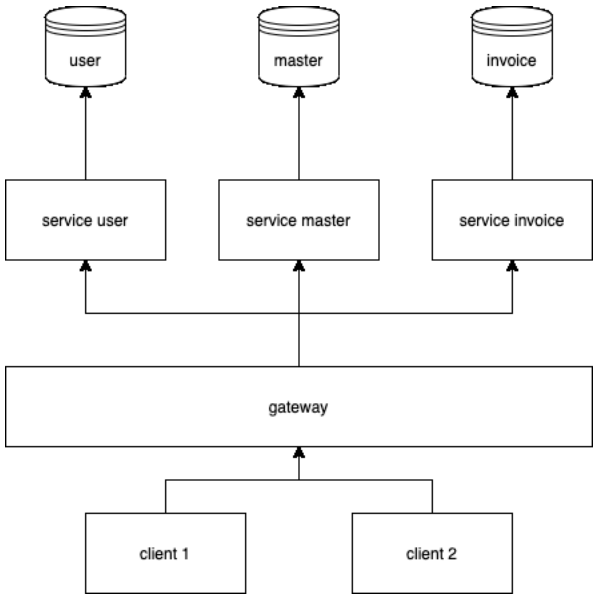
Secara keseluruhan, arsitektur *microservices* menawarkan lebih banyak fleksibilitas, skalabilitas, dan ketahanan dibandingkan dengan arsitektur monolitik, menjadikannya pilihan yang lebih baik untuk aplikasi yang kompleks dan berkembang.

1.2 Perancangan

1.2.1 Perancangan Sistem

Agar permintaan tidak berubah selama tahap pengembangan, kebutuhan fungsional harus diidentifikasi terlebih dahulu. Setelah persetujuan dicapai, tahap

desain sistem dapat dimulai. Ini akan memberikan pemahaman yang lebih luas tentang sistem *microservice* pembelian.



Gambar 3. 1 Arsitektur *Microservice*

Pada gambar di atas dijelaskan bahwa sistem arsitektur *microservices* memecah menjadi 3 bagian *service*, diantaranya *service user*, *service master* dan *service invoice*. Kemudian diikuti dengan *gateway* yang menghubungkan dengan *frontend*. Pembagian ini diharapkan ketika salah satu *service* bermasalah tidak mengganggu atau mematikan semua *service*.

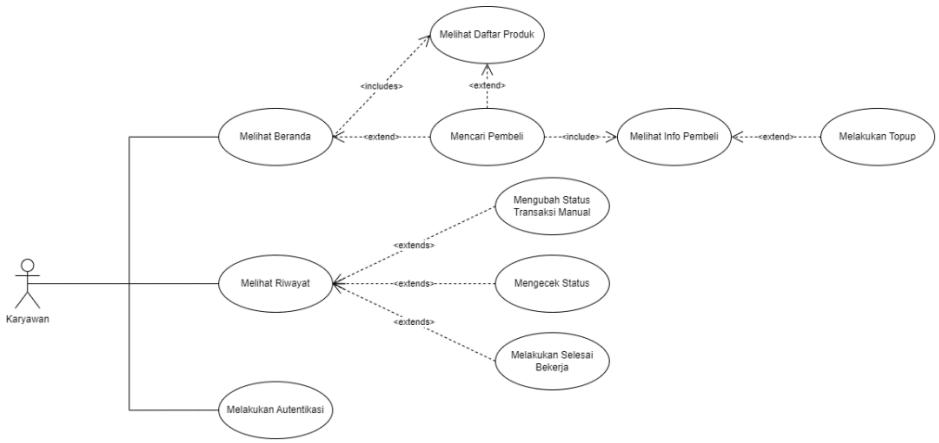
Gambaran awal sistem diberikan melalui proses perancangan. Pada penilitan ini dibagi menjadi 3 *diagram*:

a. *Use Case Diagram*

   Untuk tujuan penelitian ini, kami menggunakan diagram *use case* yang terdiri dari *use case* admin dan karyawan. Berikut adalah rancangan *use case* admin:

Gambar 3. 2 *Use Case Diagram* Admin

Dalam sistem ini admin memiliki hak untuk melakukan manajemen data karyawan dan data jam kerja, mendapatkan informasi data *invoice* dari semua transaksi yang dilakukan karyawan dalam bentuk grafik maupun riwayat *invoice* secara detail. Kemudian untuk hak karyawan digambarkan dengan *use case* pada gambar berikut:

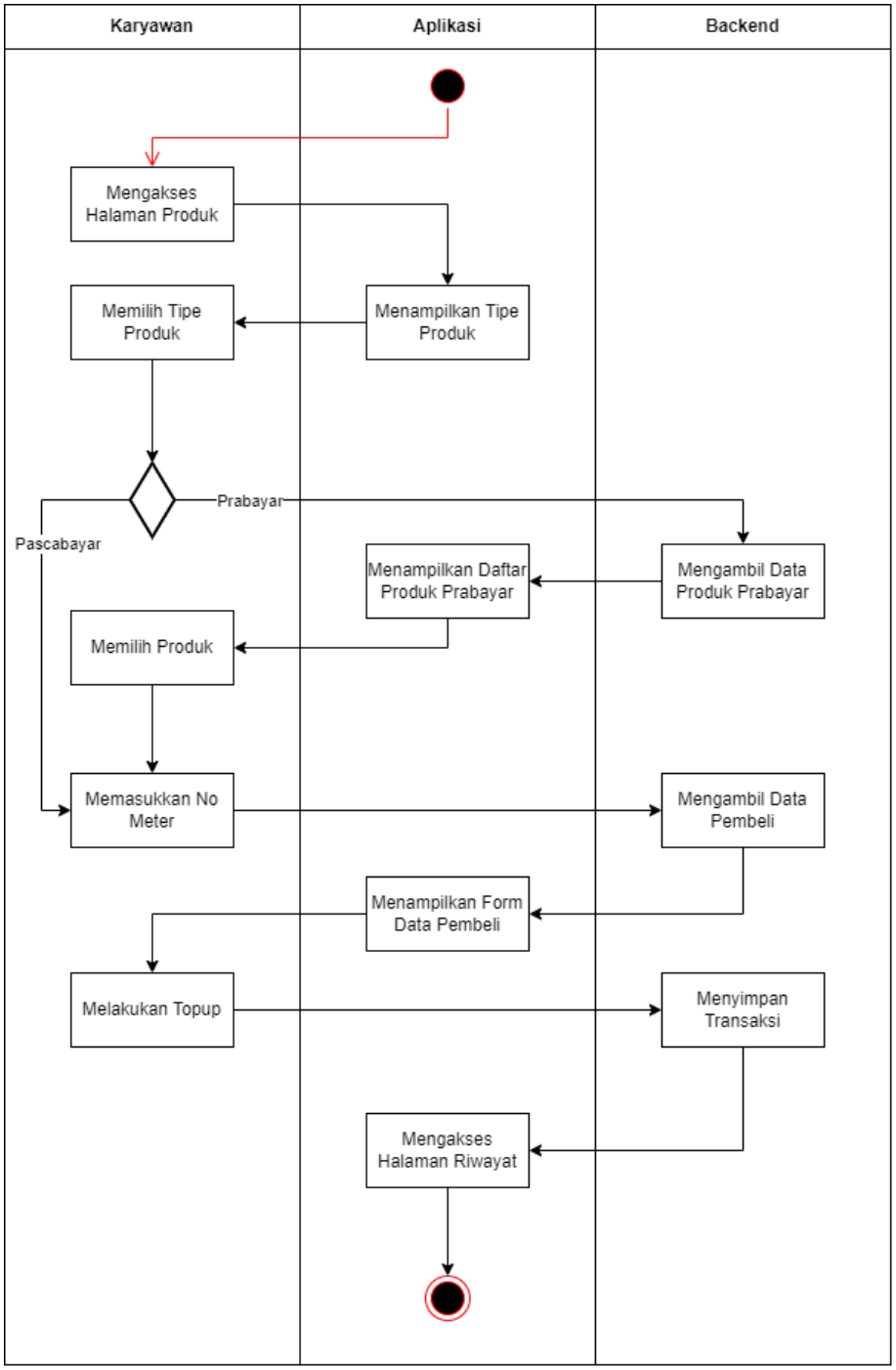


Gambar 3. 3 *Use Case Diagram* Karyawan

Dalam sistem ini karyawan memiliki hak untuk membuat transaksi pembelian produk. Produk dibagi menjadi 3 tipe, yaitu token listrik, tagihan listrik, dan PLN non taglis. Selanjutnya karyawan dapat melakukan proses pembelian produk dengan memilih produk jika tipenya token listrik. Setelah itu mengisi no meter dari pelanggan, sedangkan untuk tipe lainnya karyawan bisa langsung mengisi no meter kemudian mencari data pelanggan. Selanjutnya melakukan pembayaran dan mendapatkan *output* berupa *invoice*.

b. Activity Diagram

Activity diagram menunjukkan alur aktivitas sistem, bagaimana setiap aktivitas berinteraksi satu sama lain, dan bagaimana setiap aktivitas berakhir. Proses transaksi digambarkan dalam *activity diagram* berikut:

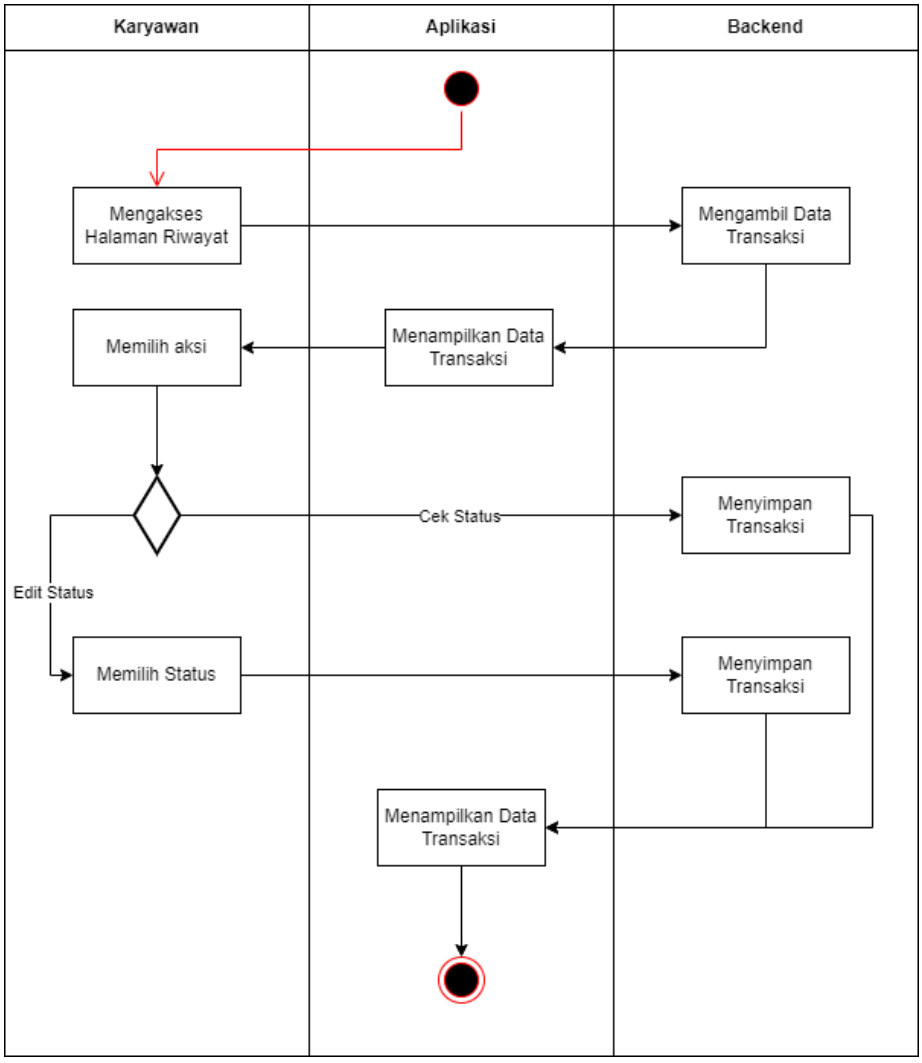


Gambar 3. 4 *Activity Diagram* Proses Transaksi

Setelah karyawan login, pada aplikasi langsung ditampilkan halaman produk yang kemudian di halaman tersebut melakukan *request* data

produk lalu karyawan bisa memilih tipe produk yang mau diproses. Ada dua kondisi ketika memproses pembelian listrik. Jika prabayar maka harus memilih produk terlebih dahulu kemudian mengisi no meter pembeli. *Backend* memproses data kemudian menyimpan data transaksi, setelah transaksi berhasil diarahkan ke halaman riwayat. Sedangkan pascabayar langsung mengisi no meter pembeli dan langsung di proses oleh *backend* untuk disimpan.

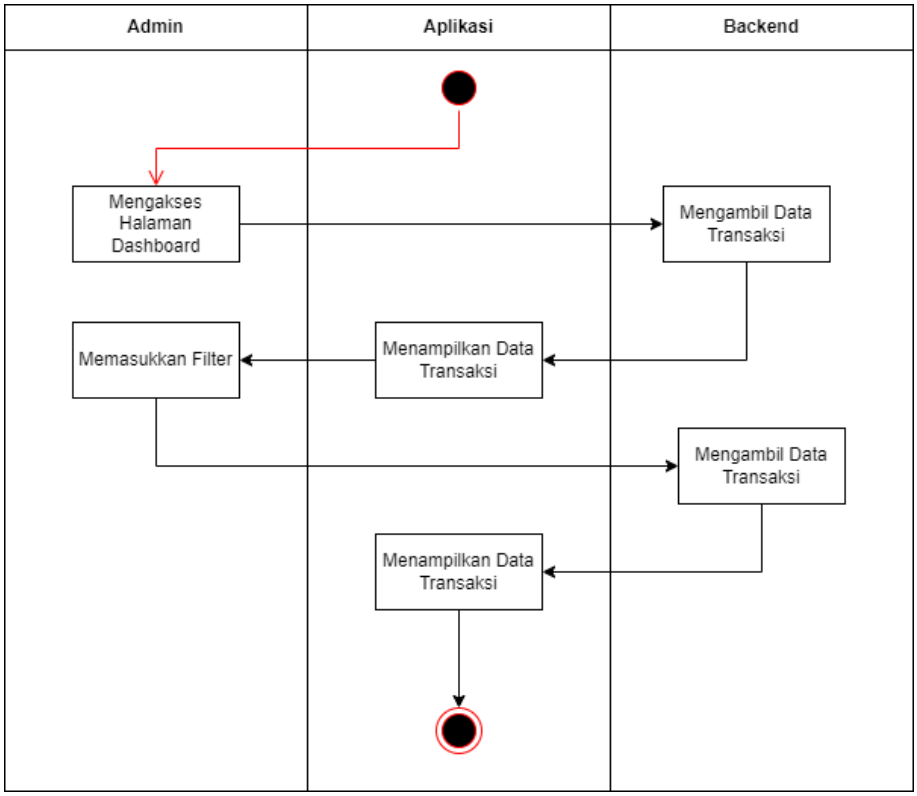
Berikut ini menampilkan *activity diagram* dari proses aksi pada riwayat:



Gambar 3. 5 *Activity Diagram* Proses *Update*

Setelah memproses transaksi kemudian karyawan akan masuk ke dalam halaman riwayat, yang dimana ada beberapa aksi yang bisa dilakukan diantaranya cek status dan edit status manual. Ketika cek status aplikasi akan mengecek perubahan status yang ada kemudian akan memperbaharui data ketika ada perubahan. Sedangkan untuk *edit status manual*, karyawan akan memilih status yang akan diubah baru kemudian menyimpan ke dalam *database*.

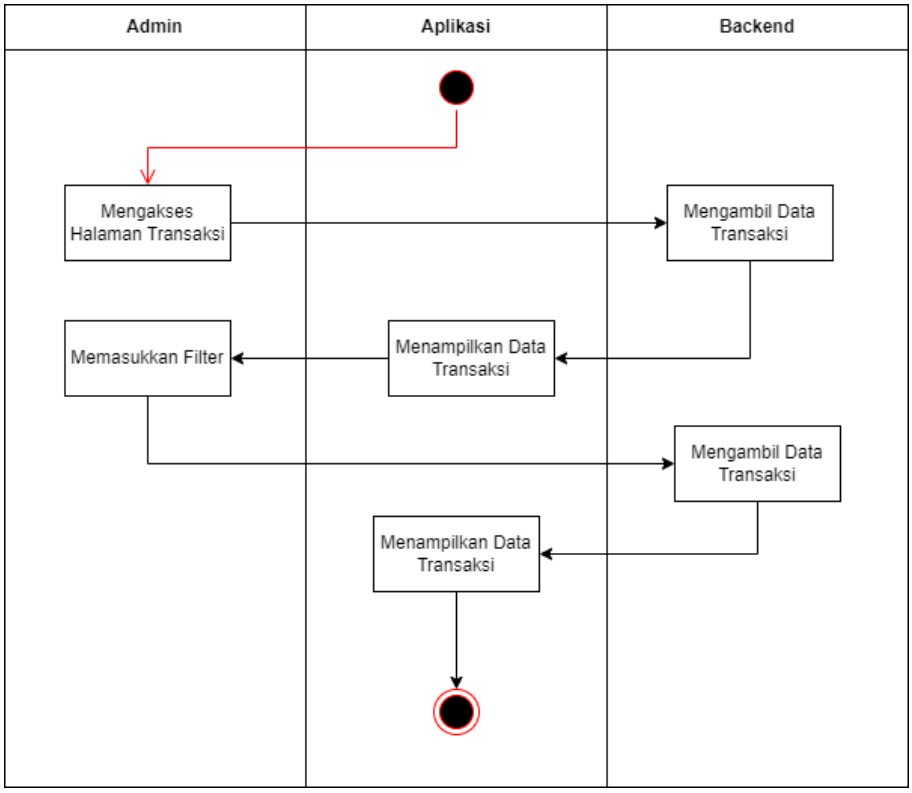
Berikut ini menampilkan *activity diagram* dari proses *dashboard* yang ada pada akses admin:



Gambar 3. 6 *Activity Diagram* Proses *Summary Dashboard*

Pada halaman *dashboard*, admin ditampilkan data transaksi dalam bentuk grafik dan ringkasan dari semua transaksi yang ada. Admin juga bisa mengubah data dengan mengganti *filter* yang sudah disediakan oleh sistem.

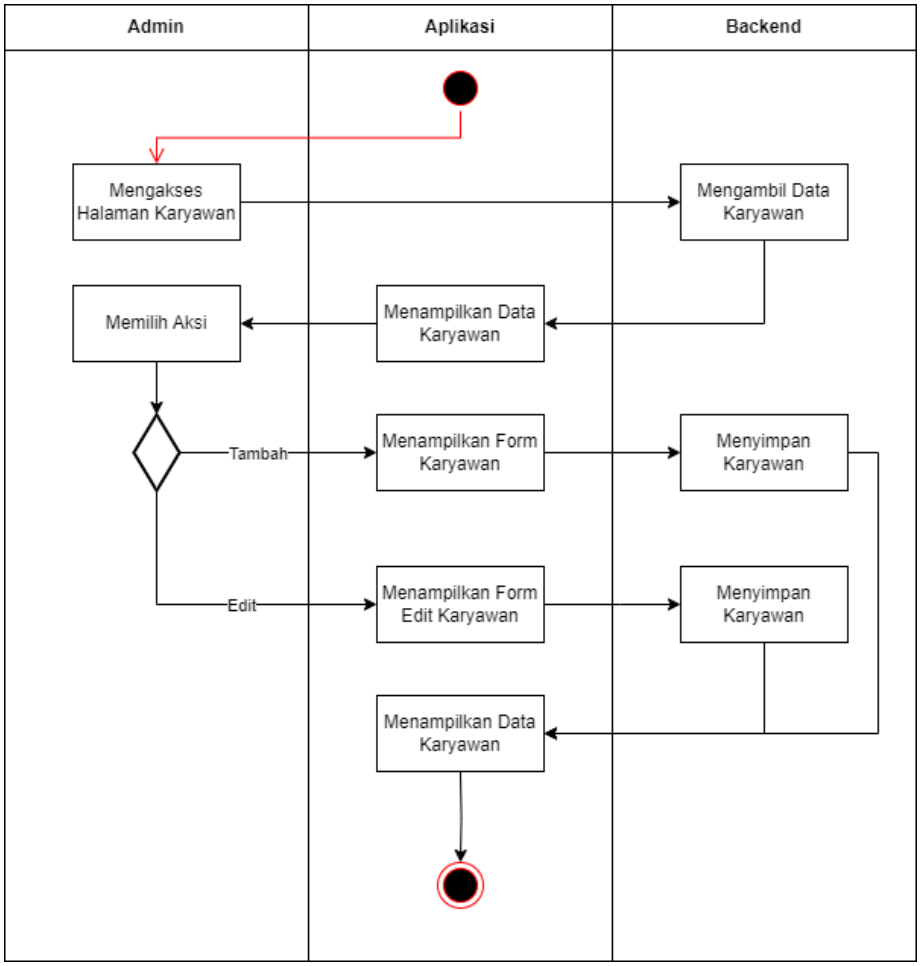
Berikut ini menampilkan *activity diagram* dari *transaksi* yang ada pada akses admin:



Gambar 3. 7 *Activity Diagram* Proses Transaksi

Pada halaman transaksi aktifitas yang bisa dilakukan admin hanya untuk mengubah data menggunakan *filter* yang disediakan oleh sistem. Data yang ditampilkan beruba data yang lebih mendetail dibandingkan dengan halaman *dashboard*.

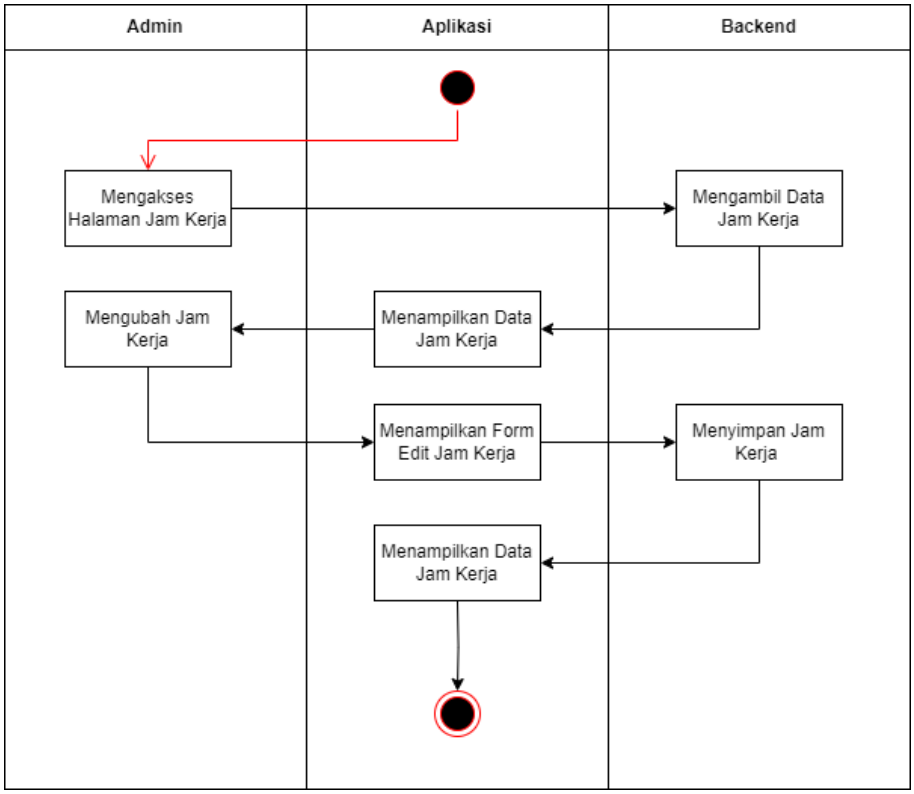
Berikut ini menampilkan *activity diagram* dari proses karyawan yang ada pada akses admin:



Gambar 3. 8 *Activity Diagram* Proses Karyawan

Setelah masuk halaman karyawan, pada aplikasi langsung ditampilkan halaman karyawan yang kemudian di halaman tersebut melakukan *request* data karyawan. Admin dapat menambah karyawan dengan mengisi data yang diperlukan, kemudian menyimpan data karyawan dalam aplikasi. Selain itu, admin juga memiliki kemampuan untuk mengubah data. karyawan yang hanya sebatas status dan jam kerja yang ada. Setelah itu, menyimpan data karyawan ke dalam aplikasi.

Berikut ini menampilkan *activity diagram* dari proses jam kerja yang ada pada akses admin:



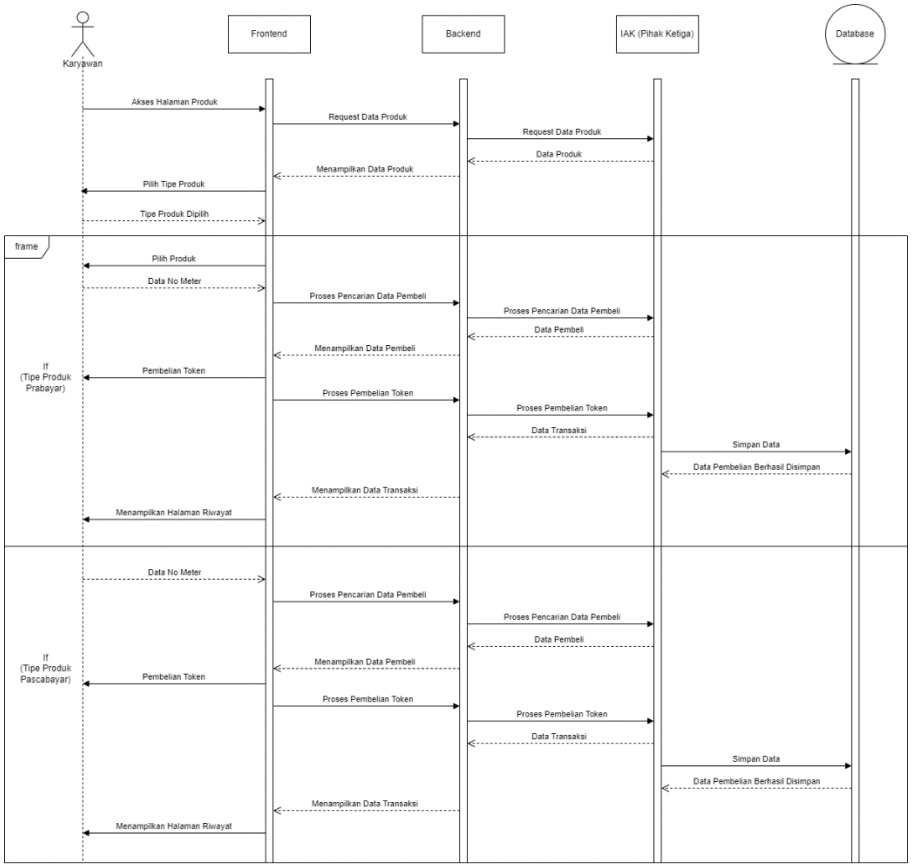
Gambar 3. 9 *Activity Diagram* Proses Jam Kerja

Pada aktifitas jam kerja, admin mendapatkan data jam kerja yang sudah ada. Admin hanya sebatas bisa mengubah status yang ada pada jam kerja. Kemudian menyimpan data tersebut kedalam aplikasi.

c. *Sequence Diagram*

Alur interaksi dinamis antara objek dalam aplikasi digambarkan dalam *sequence diagram.* Sequence diagram akan memperlihatkan lebih jelas bagaimana respon sistem aplikasi akan bertindak sesuai dengan

scenario use case yang sudah dijabarkan. Berikut *sequence diagram* untuk *scenario* transaksi pada karyawan:

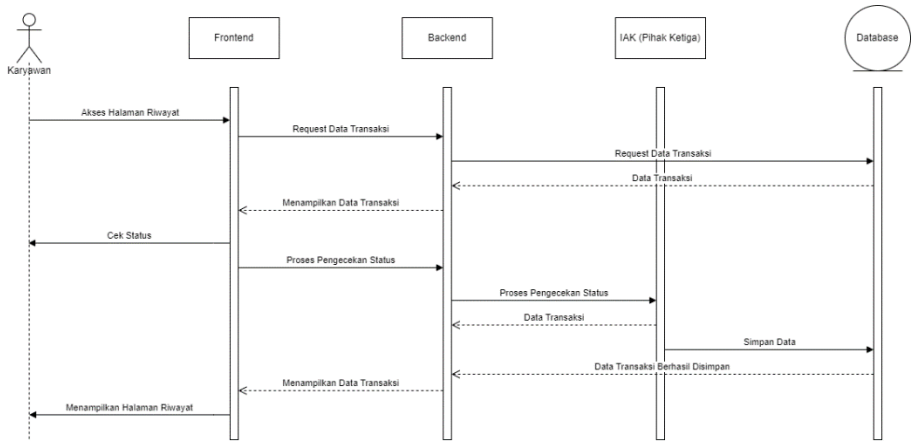


Gambar 3. 10 *Sequence* Transaksi Karyawan

Pertama kali karyawan ditampilkan halaman produk / utama. Karyawan diminta untuk memilih tipe produk yang ingin dipesan. Setelah memilih tipe produk yang dipesan, ketika tipe produk prabayar memilih produk kemudian baru mengisi no meter. Sedangkan tipe produk pascabayar langsung mengisi no meter. Kemudian *backend request* ke pihak ketiga untuk mendapatkan detail data pembeli. Setelah menampilkan data pembeli karyawan konfirmasi pembelian

transaksi lalu bagian *backend request* ke pihak ketiga dan menyimpan data produk ke dalam *database*. Setelah data berhasil disimpan, karyawan diarahkan pada halaman riwayat dan menampilkan riwayat transaksi yang sudah dilakukan.

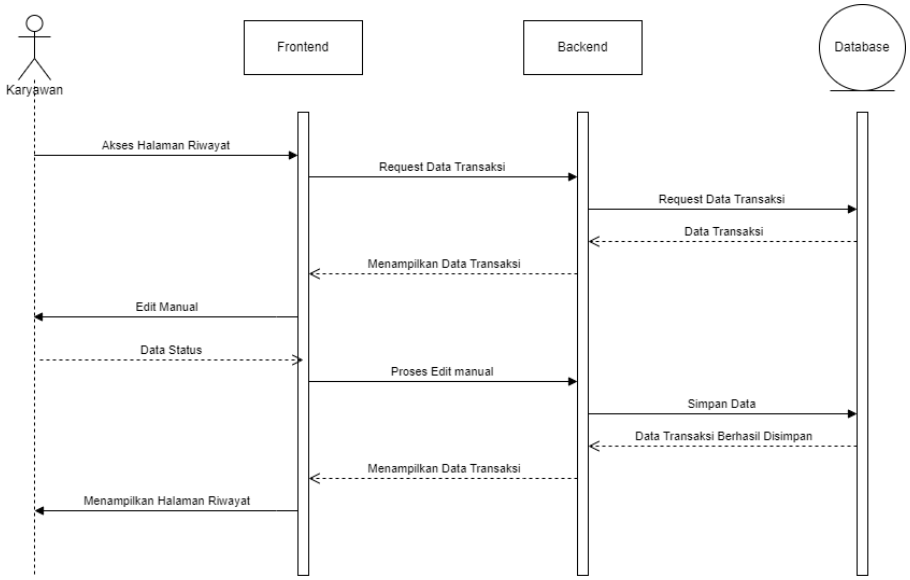
Berikut *sequence diagram* untuk *scenario* cek status pada karyawan:



Gambar 3. 11 *Sequence* Cek Status

Karyawan masuk ke dalam halaman riwayat, diberikan aksi untuk cek status. Pada saat cek status, dari *backend* mengecek data ke pihak ketiga sesuai kode unik yang dimiliki bersama kemudian memperbarui data ketika ada perubahan status. Setelah data berhasil disimpan, karyawan diarahkan pada halaman riwayat.

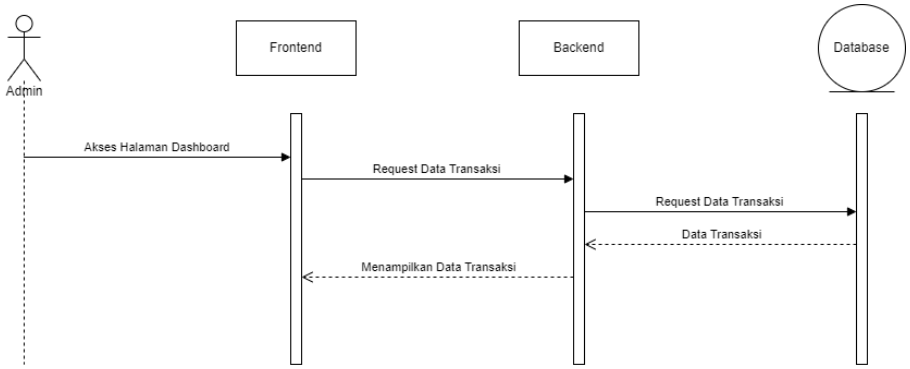
Berikut *sequence diagram* untuk *scenario edit manual* pada karyawan:



Gambar 3. 12 *Sequence Edit Manual*

Karyawan masuk ke dalam halaman riwayat, diberikan aksi untuk *edit manual*. Pada *edit manual*, karyawan akan memilih status terbaru yang akan disimpan pada *database*. Setelah data berhasil disimpan, karyawan diarahkan pada halaman riwayat.

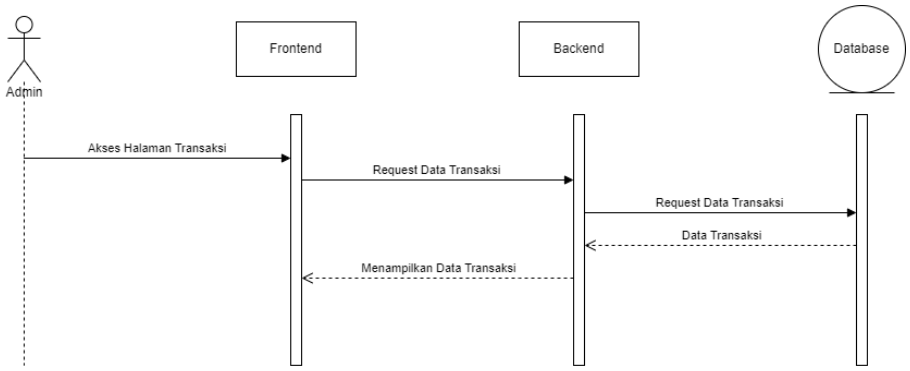
Berikut *sequence diagram* untuk *scenario dashboard* pada admin:



Gambar 3. 13 *Sequence Dashboard*

Admin masuk ke dalam halaman *dashboard*, kemudian *backend* akan request data transaksi yang kemudian diberikan data transaksi. Data yang ditampilkan berupa ringkasan dari transaksi yang ada dan ditampilkan dalam bentuk grafik.

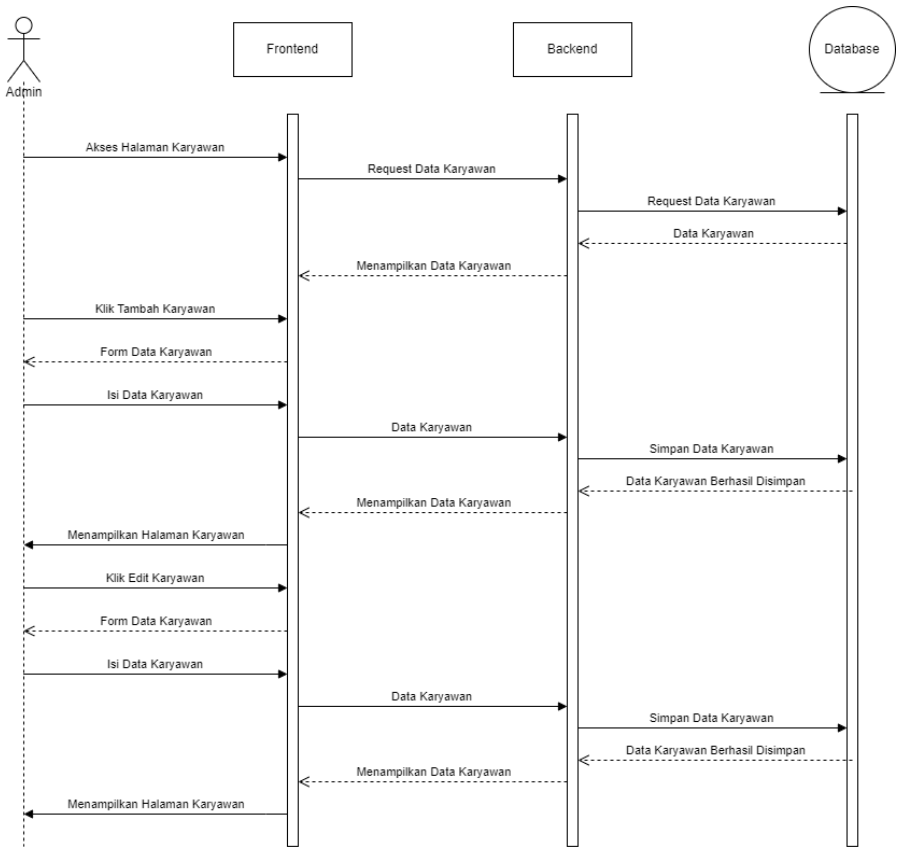
Berikut *sequence diagram* untuk *scenario* transaksi pada admin:



Gambar 3. 14 *Sequence* Transaksi Admin

Admin masuk ke dalam halaman transaksi, kemudian *backend* akan request data transaksi yang kemudian diberikan data transaksi. Data yang ditampilkan berupa daftar dari transaksi yang ada dan ditampilkan dalam bentuk secara mendetail.

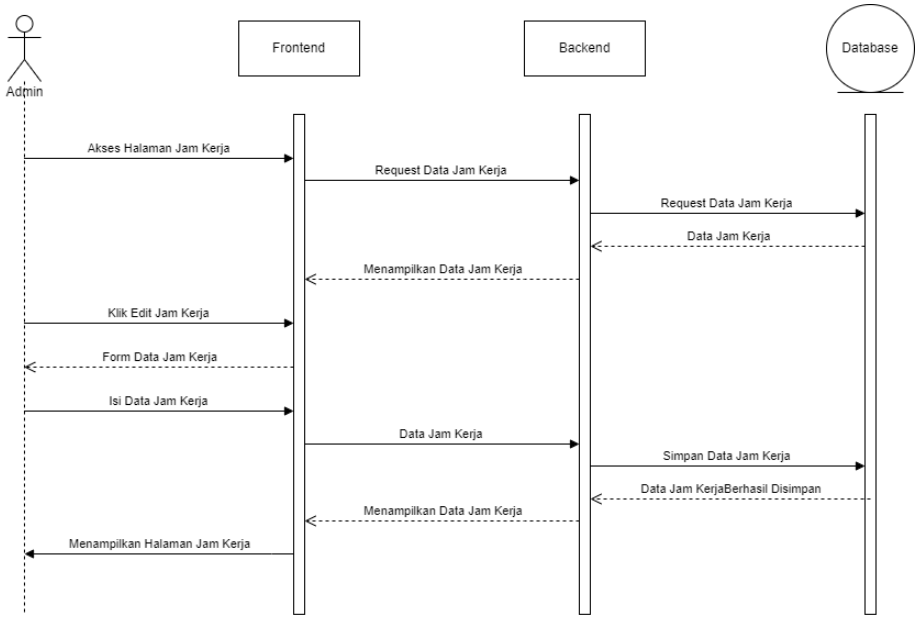
Berikut *sequence diagram* untuk *scenario* karyawan pada admin:



Gambar 3. 15 *Sequence* Karyawan

Admin masuk ke dalam halaman karyawan, kemudian *backend* akan *request* data karyawan yang kemudian diberikan data karyawan. Dari data yang ditampilkan, admin bisa menambahkan karyawan dengan mengisi data karyawan kemudian mengirimkan ke *backend* yang selanjutnya aka disimpan. Setelah berhasil disimpan admin dapat melihat halaman karyawan dengan data terbaru. Admin juga bisa mengubah data karyawan yang ada dengan mengisi data yang diperbolehkan untuk diperbaharui. Setelah berhasil disimpan admin dapat melihat halaman karyawan dengan data terbaru

Berikut *sequence diagram* untuk *scenario* jam kerja pada admin:



Gambar 3. 16 *Sequence* Jam Kerja

Admin masuk ke dalam halaman jam kerja, kemudian *backend* akan *request* data jam kerja yang kemudian diberikan data jam kerja. Dari data yang ditampilkan, admin bisa mengubah data yang ada dengan mengisi data yang diperbolehkan untuk diperbaharui. Setelah berhasil disimpan admin dapat melihat halaman jam kerja dengan data terbaru.

1.2.2 Perancangan Data

Database adalah sekumpulan data yang disimpan secara sistematis di dalam komputer dan dapat diakses dan dikelola dengan program komputer yang digunakan untuk mengekstrak dan mengakses data tersebut. Pada sistem ini *database* yang digunakan adalah mongoDB dan setiap *service* mengakses *database* yang berbeda, berikut adalah beberapa *database* dan *schema collection* dari *database* setiap *service* yang digunakan di sistem ini.

1. *Database Service User*

    a. *Schema User*

    Schema user digunakan untuk menyimpan data-data *user* yang terdaftar di sistem, seperti admin dan karyawan.

    Tabel 3. 1 *Schema User*

Nama *Field*	Keterangan
id (ObjectId)	Id user
name (String)	Nama user
phone (String)	No handphone user
email (String)	Email user
password (String)	Password user
address (String)	Alamat user
numberId (String)	No. Karyawan
active (Boolean)	Status user
userType (String)	Tipe user
officeHoursId (ObjectId)	Id jam kerja
createdAt (Date)	Tanggal data dibuat
updatedAt (Date)	Tanggal data diperbaharui

2. *Database Service Master*

    a. *Schema* jam kerja

    Schema jam kerja digunakan untuk menyimpan jam kerja yang digunakan oleh karyawan.

    Tabel 3. 2 *Schema* Jam Kerja

Nama *Field*	Keterangan
id (ObjectId)	Id jam kerja
name (String)	Nama jam kerja
startAt (String)	Jam kerja mulai
endAt (String)	Jam kerja selesai
active (Boolean)	Status jam kerja
createdAt (Date)	Tanggal data dibuat
updatedAt (Date)	Tanggal data diperbaharui

3. *Database Service Invoice*

a. *Schema Invoice*

Schema invoice digunakan untuk menyimpan data-data transaksi.

Tabel 3. 3 *Schema Invoice*

Nama *Field*	Keterangan
id (ObjectId)	Id invoice
invoice (String)	Kode unik invoice
customer_id (String)	No meter pelanggan
product_code (String)	Kode produk yang di beli
status (Number)	Status invoice
admin (Number)	Biaya admin
nominal (Number)	Harga yang dibayarkan
price (Number)	Total harga yang dibayarkan
rc (String)	Kode respon dari pihak ketiga
tr_id (Number)	Id transaksi dari pihak ketiga
info (String)	Data dari pihak ketiga
user_id (ObjectId)	Id karyawan
createdAt (Date)	Tanggal data dibuat
updatedAt (Date)	Tanggal data diperbaharui

1.2.3 Perancangan User Interface / *Mock-up* aplikasi

Gambar *wireframe* menunjukkan perancangan antarmuka pengguna (UI), termasuk tampilan halaman utama, beranda, transaksi, karyawan, dan jam kerja.

a. Tampilan Login

Gambar berikut menunjukkan halaman login yang muncul ketika program dimulai. Di sana, pengguna harus mengisi alamat email dan password mereka untuk masuk ke sistem.

Gambar 3. 17 *Mock-up* Login

b. Tampilan Halaman Utama

Di tampilan halaman utama menampilkan data secara grafik dimana ada perhitungan transaksi yang ada, jumlah transaksi berdasarkan status. Di tampilan ini juga menampilkan tren transaksi dalam beberapa waktu.

Gambar 3. 18 *Mock-up* Halaman Utama

c. Tampilan Transaksi

Halaman transakasi ini berisi tentang semua transaksi yang di proses oleh karyawan. Di halaman ini juga bisa di *filter* berdasarkan karyawan dan jam kerja yang ada.

Gambar 3. 19 *Mock-up* Transaksi

d. Tampilan Karyawan

Halaman karyawan berisi data karyawan yang sudah terdaftar pada sistem ini. Mengatur untuk jam kerja dan menambah karyawan baru.

Gambar 3. 20 *Mock-up* Karyawan

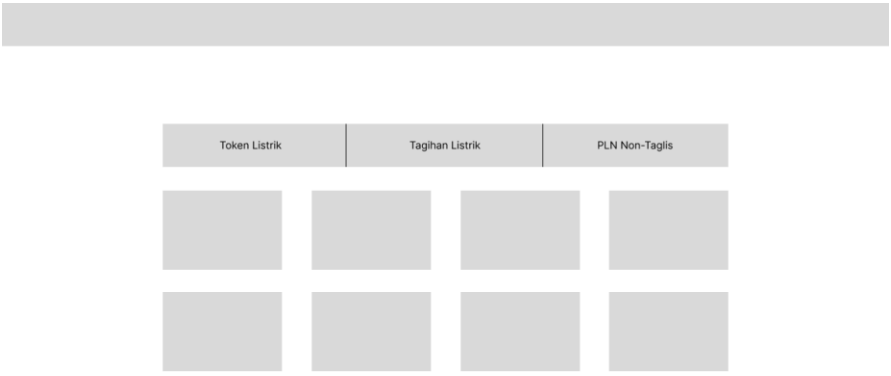
e. Tampilan Jam Kerja

Tampilan jam kerja berisi status jam kerja yang aktif dan bisa membantu perbarui jam kerja yang aktif.

Gambar 3. 21 *Mock-up* Jam Kerja

f. Tampilan Produk

Tampilan produk ini berada pada akses karyawan, disini dibedakan pada 3 jenis produk.

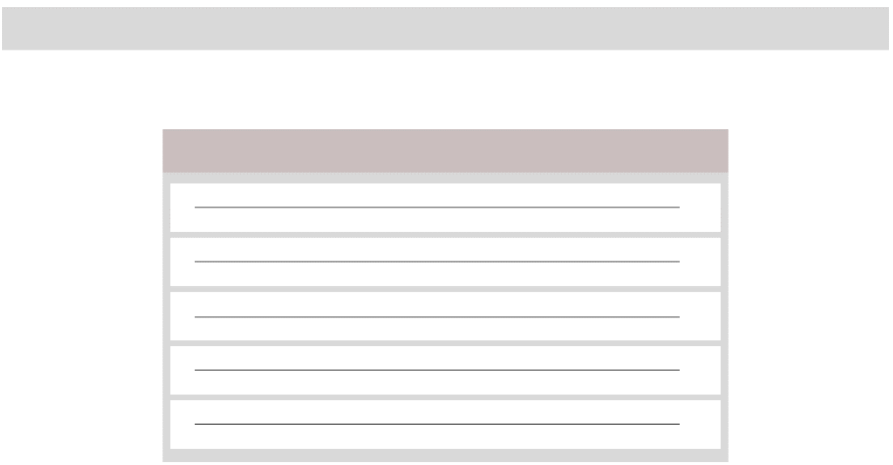


Gambar 3. 22 *Mock-up* Produk Prabayar

Gambar 3. 23 *Mock-up* Produk Pascabayar

g. Tampilan Transaksi Karyawan

Tampilan transaksi memiliki sedikit tampilan berbeda dengan yang ada pada admin, namun hanya muncul berdasarkan karyawan yang sudah bertugas.



Gambar 3. 24 *Mock-up* Karyawan

1.3 Rancangan Pengujian

Mengidentifikasi kebutuhan untuk pengujian perangkat lunak *load test* adalah bagian dari proses perencanaan pengujian perangkat lunak.

Perangkat lunak pengujian *load test* dirancang untuk mengukur kapasitas sebuah aplikasi untuk menangani beban. Banyaknya pengguna yang mengakses sebuah aplikasi dalam waktu tertentu menunjukkan kapasitas sebuah aplikasi untuk menangani beban tersebut. Dalam penelitian ini, pengujian menggunakan tiga parameter yaitu *connections*, *pipelines*, dan lama pengujian.

Tabel 3. 4 Rancangan Pengujian

Nama Arsitektur	*Connections*	*Pipelines*	Lama Pengujian
Microservice	100	10	1 menit
	500	100	1 menit
	1000	100	1 menit

Pengujian kedua dilakukan untuk mengetahui salah satu keunggulan arsitektur *microservices* yaitu toleransi kesalahan yang tinggi. Dimana skenario yang digunakan dengan cara mematikan salah satu *service* atau layanan dan memastikan tidak mempengaruhi *service* atau layanan yang lain.