menambahkan karyawan serta memberi akses jam kerja kepada karyawan.

b. Karyawan

Karyawan bertugas untuk melakukan transaksi pembelian listrik yang dilakukan oleh pelanggan. Selain itu, karyawan juga berhak untuk mengetahui total transaksi yang diproses pada saat dia bekerja. Karyawan hanya bisa menggunakan aplikasi ketika jam kerjanya sudah aktif.

4.2 Implementasi

4.2.1 Spesifikasi Produk

Hasil aplikasi disajikan dalam bentuk *playground* dan website yang mudah untuk digunakan dan diakses. Aplikasi dijalankan dalam sebuah node JS *server*. Minimum spesifikasi *hardware* untuk dapat menggunakan aplikasi ini adalah minimal RAM 2GB dan ruang kosong penyimpanan minimal 3GB, dan juga *hardware* harus mendukung koneksi internet dan sudah terinstal browser pada perangkat, pengguna dapat mengakses dan menggunakan aplikasi dengan menjalankan *command* yang sudah di tentukan.

4.2.2 Implementasi *Database*

MongoDB digunakan dalam proses pembuatan database untuk website pembelian token listrik. Penerapan MongoDB berbeda jauh dengan MySQL, yang dimana MongoDB memilik *syntax*-nya sendiri. Dalam pembuatan tabel, pada

MongoDB menggunakan “new mongoose.Schema”. Pengaturan untuk tiap kolom didefinisikan di dalam *object* nama kolomnya. Ada banyak konfigurasi untuk pembuatan kolom namun peneliti tidak membutuhkan semua, yang digunakan hanya yang sesuai kebutuhan aplikasi, diantaranya *type, required, index, default.*

Kegunaan *type* untuk mendefinisikan kolom tersebut menggunakan tipe data, yang umum digunakan adalan (*String*, *Number*, *Boolean*, *Date*, *ObjectId*). Kegunaan *required* untuk mengatur kolom tersebut harus diisi atau tidak. *Index* untuk membantu pencarian data yang efisien dalam basis data. Sedangkan *default* digunakan untuk memberi nilai jika kolom tidak diisi.

Implementasi beberapa *database* dan *table* yang ada pada penelitian ini, seperti yang ditunjukkan pada gambar berikut:

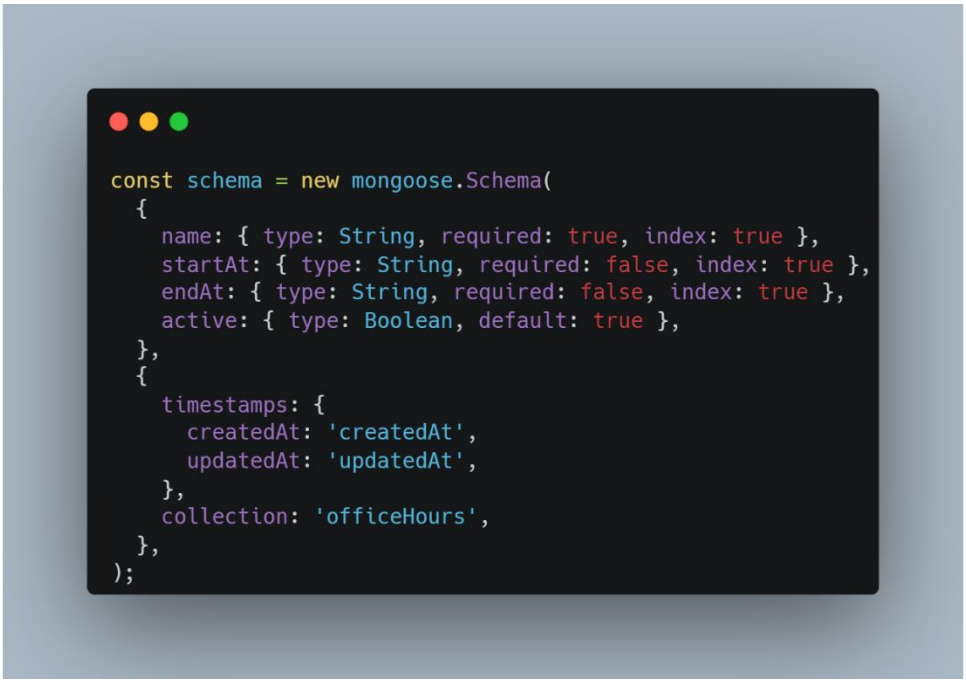
Database Service User

```
const schema = new mongoose.Schema(
  {
    name: { type: String, required: true, index: true },
    phone: { type: String, required: true },
    email: { type: String, required: true, index: true },
    numberId: { type: String, required: true, index: true },
    address: { type: String, required: false },
    password: { type: String, select: false },
    active: { type: Boolean, default: true },
    userType: { type: String, enum: ['user', 'admin'], default: 'user' },
    officeHoursId: { type: mongoose.Schema.Types.ObjectId, index: true },
  },
  {
    timestamps: {
      createdAt: 'createdAt',
      updatedAt: 'updatedAt',
    },
    collection: 'user',
  },
);
```

Gambar 4. 1 Tabel *User*

Pada *database service user* ini dibuatkan sebuah *table* bernama *user*, dengan memiliki kolom yaitu *name, phone, email, numberId, address, password, active, userType, officeHoursId, createdAt* dan *updatedAt*. Tabel ini berfungsi untuk menyimpan semua data karyawan dan admin menjadi satu.

Database Service Master



```
const schema = new mongoose.Schema(
  {
    name: { type: String, required: true, index: true },
    startAt: { type: String, required: false, index: true },
    endAt: { type: String, required: false, index: true },
    active: { type: Boolean, default: true },
  },
  {
    timestamps: {
      createdAt: 'createdAt',
      updatedAt: 'updatedAt',
    },
    collection: 'officeHours',
  },
);
```

Gambar 4. 2 Tabel Jam Kerja

Pada *database master* ini dibuatkan sebuah *table* bernama *officeHours*, dengan memiliki kolom yaitu *name, startAt, endAt, active, createdAt* dan *updatedAt*. Tabel ini berfungsi untuk menyimpan data jam kerja untuk diimplementasikan ke karyawan.

Database Service Invoice

```
const schema = new mongoose.Schema(
  {
    invoice: { type: String, required: true},
    customer_id: { type: String, required: true, index: true },
    product_code: { type: String, required: true },
    status: { type: Number, required: true },
    admin: { type: Number, required: false, default: 0 },
    nominal: { type: Number, required: false, default: 0 },
    price: { type: Number, required: true },
    rc: { type: String, required: true },
    tr_id: { type: Number, required: true },
    info: { type: String, required: false },
    user_id: { type: String, required: true },
  },
  {
    timestamps: {
      createdAt: 'createdAt',
      updatedAt: 'updatedAt',
    },
    collection: 'transaction',
  },
);
```

Gambar 4. 3 Tabel *Invoice*

Pada *database invoice* ini dibuatkan sebuah *table* bernama *transaction*, dengan memiliki kolom yaitu *invoice*, *customer_id*, *product_code*, *status*, *admin*, *nominal*, *price*, *rc*, *tr_id*, *info*, *user_id*, *createdAt* dan *updatedAt*. Data transaksi yang diproses oleh karyawan disimpan dalam tabel ini.

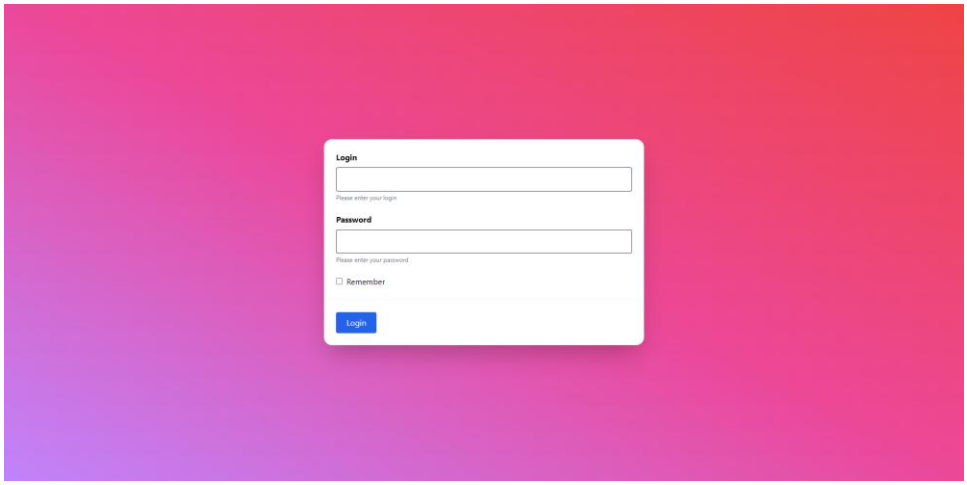
4.2.3 Implementasi Program

Setelah melakukan pembuatan *database*, selanjutnya adalah proses implementasi pembuatan program menggunakan Visual Studio Code. Aplikasi ini menggunakan *framework* Fastify untuk *backend*, sedangkan untuk *frontend* menggunakan nextJs.

Pada implementasi program ini, diputuskan untuk menggunakan pihak ketiga untuk proses pencarian produk maupun pelanggan melalui no meter. Alasan penggunaan pihak ketiga dikarenakan data selalu terbaru dan sudah terintegrasi langsung dengan PLN secara langsung. Proses pengimplementasian ini dibagi menjadi tiga *service*, yaitu *service user*, *service master*, *service* transaksi. *Service user* dibuat hanya untuk menangani semua tentang pengguna aplikasi tersebut. *Service master* digunakan untuk menangani semua yang berhubungan dengan master data, disini ada jam kerja dan juga produk yang disambungkan dengan pihak ketiga IAK. Semua produk yang ditampilkan di halaman karyawan ditempatkan pada *service master* sehingga tidak mengganggu di *service* yang lain. *Service* ketiga digunakan untuk khusus transaksi, semua transaksi dari *create*, *update* status, dan *dashboard* pada admin/pemilik ada pada bagian ini.

Tampilan *user interface* (UI) aplikasi yang dibuat ditunjukkan di sini, beserta segmentasi program (*source code*) dari masing-masing *interface* tersebut.

Pertama, halaman login untuk user maupun admin:



Gambar 4. 4 Tampilan Halaman Login

Pada halaman login aplikasi, pengguna harus memasukkan email dan password yang telah mereka masukkan sebelumnya. Ini adalah tampilan awal sebelum mereka dapat masuk ke aplikasi. Jika pengguna salah memasukkan email atau password, mereka tidak akan dapat masuk ke aplikasi. Disisi lain untuk karyawan ada beberapa aturan ketika hendak masuk kedalam aplikasi. Selain harus memiliki akun, aturan kedua adalah karyawan harus masuk sesuai jam kerja yang telah ditentukan admin. Jika tidak memenuhi persyaratan tersebut maka notifikasi yang dikeluarkan untuk karyawan belum waktu jam kerja. Penerapannya dalam aplikasi ada pada segmentasi program yang ada pada dibawah ini.

Berikut adalah *source code* untuk login, pada bagian backend:

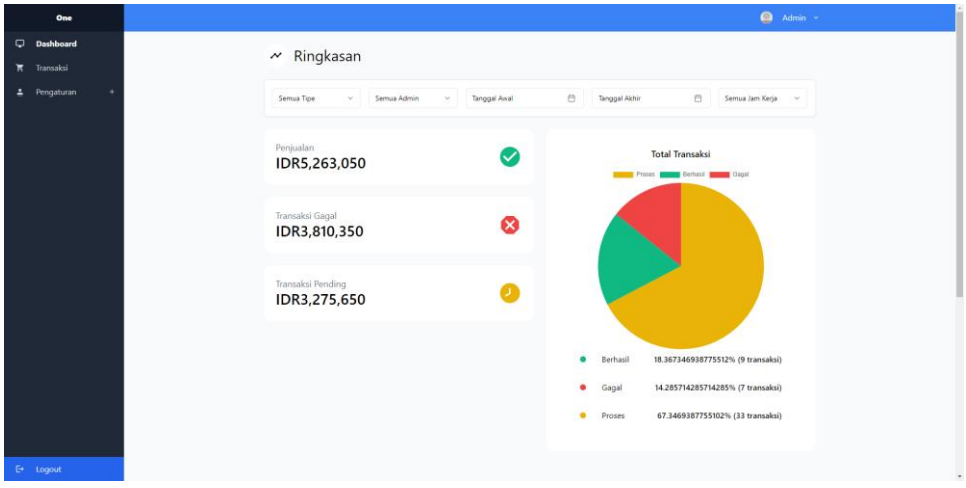
```
async login(data: Partial<ILogin>): Promise<object> {
    const { email, password } = data;
    const response = await User.findOne({ email }).select('+password');
    if (!response) {
      throw new Error('Email atau password salah');
    }

    const isPasswordCorrect = authenticate(password, response.password);
    if (!isPasswordCorrect) {
      throw new Error('Email atau password salah');
    }
    if (response.userType === 'user') {
      const master: any = await fetch(`${process.env.SERVICE_MASTER}/api/office-
hours/${response.officeHoursId}`);
      const dataMaster = await master.json();
      if (
        !(
          new Date().getHours() >= new Date(dataMaster.data.startAt).getHours() &&
          new Date().getHours() <= new Date(dataMaster.data.endAt).getHours() &&
          dataMaster.data.active
        )
      ) {
        throw new Error('Belum waktunya jam kerja');
      }
    }

    const token = jwtSign({
      id: response.id,
      name: response.name,
      phone: response.phone,
      email: response.email,
      numberId: response.numberId,
      address: response.address,
      active: response.active,
```

Segmen Program 4. 1 *Source Code* Login

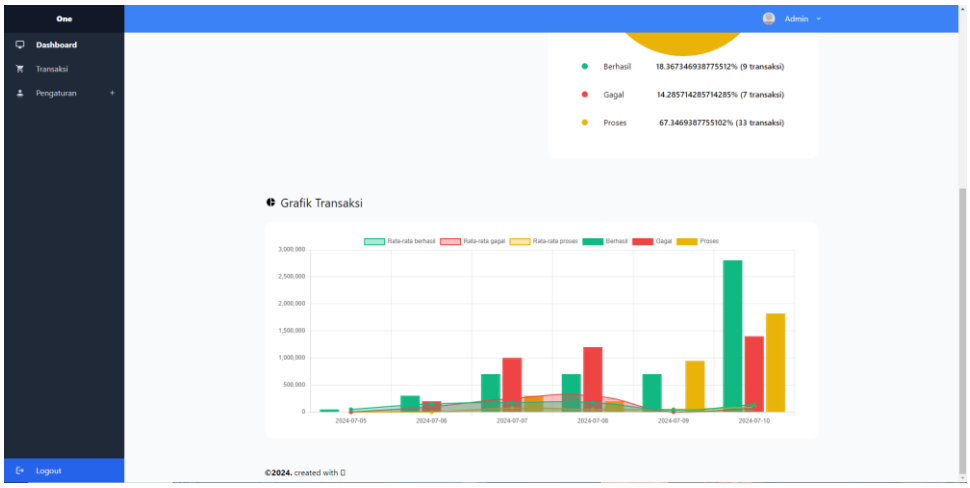
```
    userType: response.userType,
    officeHoursId: response.officeHoursId,
  });
  return {
    token,
    _id: response.id,
    name: response.name,
    phone: response.phone,
    email: response.email,
    numberId: response.numberId,
    address: response.address,
    active: response.active,
    userType: response.userType,
    officeHoursId: response.officeHoursId,
    createdAt: response.createdAt,
    updatedAt: response.updatedAt,
  };
}
```

Segmen Program 4. 2 *Source Code* Login II

Selanjutnya adalah tampilan dashboard untuk admin/pemilik:



Gambar 4. 5 Tampilan *Dashboard* Admin



Gambar 4. 6 Tampilan *Dashboard* Admin II

Tampilan *dashboard* yang ada pada admin, memiliki tiga komponen yang ditampilkan secara data. Pada komponen pertama menampilkan jumlah penjualan berdasarkan status yang ada, jumlah penjualan ini dalam bentuk total rupiah. Untuk komponen kedua menampilkan grafik pie, yang berisikan total transaksi berdasarkan status yang ada. Dalam tampilan ini, admin bisa mengetahui total transaksi yang telah diproses maupun belum oleh karyawan. Grafik ini juga memudahkan mengidentifikasi dalam bentuk presentase dari semua transaksi yang ada. Komponen ketiga merupakan data transaksi yang dilakukan oleh karyawan dalam bentuk grafik yang ditampilkan dalam waktu per hari. Tujuannya agar admin mengetahui tren penjualan perhari yang dilakukan oleh karyawan.

Pada tampilan *dashboard* juga, admin bisa mengubah data yang mau dilihat berdasarkan beberapa pengaturan yang disediakan. *Filter* yang ada pada halaman dashboard ini terdiri dari tipe produk, karyawan, tanggal mulai, tanggal selesai dan jam kerja. Admin bisa mengatur data sesuai yang diharapkan, untuk tipe produk

akan menyaring transaksi berdasarkan tipe produk yang ada. *Filter* karyawan berfungsi untuk menyaring data berdasarkan karyawan yang memproses transaksi. Disisi lain bisa di saring berdasarkan *range* tanggal yang diinginkan dan transaksi yang diproses berdasarkan jam kerja.

Berikut adalah *source code* untuk dashboard admin/pemilik pada bagian backend:

```
async getTransactionSummary(query: any, token: string): Promise<object> {
  let filter: any = {};
  switch (query.type) {
    case 'all':
      filter.product_code = { $ne: null };
      break;
    case 'PREPAID':
      filter.product_code = { $nin: ['PLNPOSTPAID', 'PLNNONTAG'] };
      break;

    default:
      filter.product_code = query.type;
      break;
  }
  if (query.start_date && query.end_date) {
    filter.createdAt = {
      $gte: new Date(new Date(query.start_date).setHours(0, 0, 0, 0)),
      $lt: new Date(new Date(query.end_date).setHours(23, 59, 59, 999)),
    };
  }
  if (query.office_hour !== 'all') {
    const master: any = await fetch(`${process.env.MASTER_SERVICE}/api/office-
hours/${query.office_hour}`);
    const dataMaster = await master.json();
```

Segmen Program 4. 3 *Source Code Dashboard* Transaksi

```
    filter['time.hour'] = {
      $gte: Number(+dataMaster.data.startAt.substr(11, 2)),
      $lt: Number(+dataMaster.data.endAt.substr(11, 2)) === 0 ? 24 :
Number(+dataMaster.data.endAt.substr(11, 2)),
    };
  }
  if (query.user_id !== 'all') {
    filter.user_id = query.user_id;
  }

  const data = await Transaction.aggregate([
   { $set: { time: { $dateToParts: { date: '$createdAt' } } } },
   { $match: filter },
   {
    $group: {
     _id: null,
     price: {
      $sum: { $cond: [{ $eq: ['$status', 1] }, '$price', 0] },
     },
     loss: {
      $sum: { $cond: [{ $eq: ['$status', 2] }, '$price', 0] },
     },
     waiting: {
      $sum: { $cond: [{ $or: [{ $eq: ['$status', 0] }, { $eq: ['$status', 3] }] }, '$price', 0] },
     },
     process: {
      $sum: { $cond: [{ $or: [{ $eq: ['$status', 0] }, { $eq: ['$status', 3] }] }, 1, 0] },
     },
     failed: {
      $sum: { $cond: [{ $eq: ['$status', 2] }, 1, 0] },
     },
     success: {
      $sum: { $cond: [{ $eq: ['$status', 1] }, 1, 0] },
     },
    },
```

Segmen Program 4. 4 *Source Code Dashboard* Transaksi II

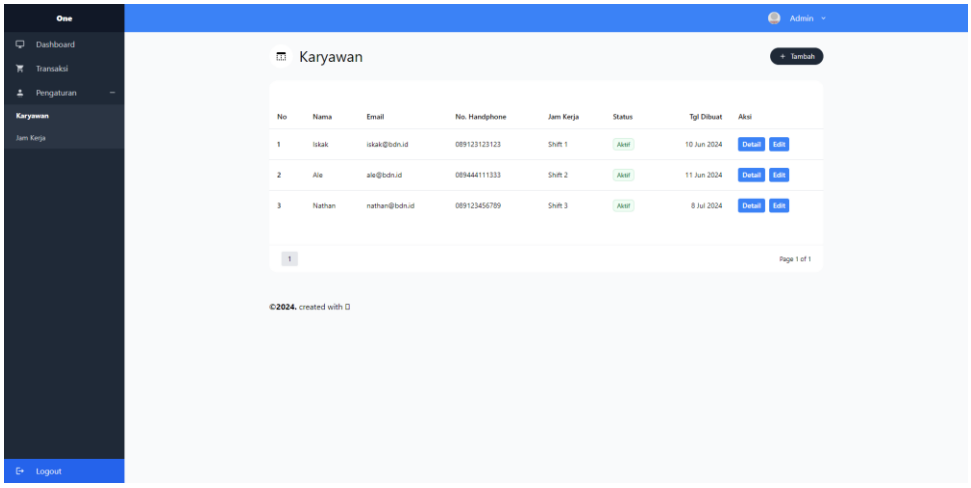
```
  },
  { $sort: { createdAt: -1 } },
]);
const history = await Transaction.aggregate([
  { $set: { time: { $dateToParts: { date: '$createdAt' } } } },
  { $match: filter },
  {
    $group: {
      _id: { $dateToString: { format: '%Y-%m-%d', date: '$createdAt', timezone: 'Asia/Jakarta' } },
      success: {
        $sum: { $cond: [{ $eq: ['$status', 1] }, '$price', 0] },
      },
      process: {
        $sum: { $cond: [{ $or: [{ $eq: ['$status', 0] }, { $eq: ['$status', 3] }] }, '$price', 0] },
      },
      failed: {
        $sum: { $cond: [{ $eq: ['$status', 2] }, '$price', 0] },
      },
      avgSuccess: {
        $avg: { $cond: [{ $eq: ['$status', 1] }, '$price', 0] },
      },
      avgProcess: {
        $avg: { $cond: [{ $or: [{ $eq: ['$status', 0] }, { $eq: ['$status', 3] }] }, '$price', 0] },
      },
      avgFailed: {
        $avg: { $cond: [{ $eq: ['$status', 2] }, '$price', 0] },
      },
    },
  },
  { $sort: { _id: 1 } },
]);

const tmpData = {
```

Segmen Program 4. 5 *Source Code Dashboard* Transaksi III

```
    price: data[0]?.price || 0,
    waiting: data[0]?.waiting || 0,
    loss: data[0]?.loss || 0,
    history: history.map((x) => {
      return {
        label: x._id,
        success: x.success,
        process: x.process,
        failed: x.failed,
        avgSuccess: x.avgSuccess,
        avgProcess: x.avgProcess,
        avgFailed: x.avgFailed,
      };
    }),
    status: {
      process: data[0]?.process || 0,
      success: data[0]?.success || 0,
      failed: data[0]?.failed || 0,
    },
  };

  return tmpData;
}
```

Segmen Program 4. 6 *Source Code Dashboard* Transaksi IV

Tampilan selanjutnya merupakan tampilan dari halaman karyawan:



Gambar 4. 7 Tampilan Halaman Karyawan

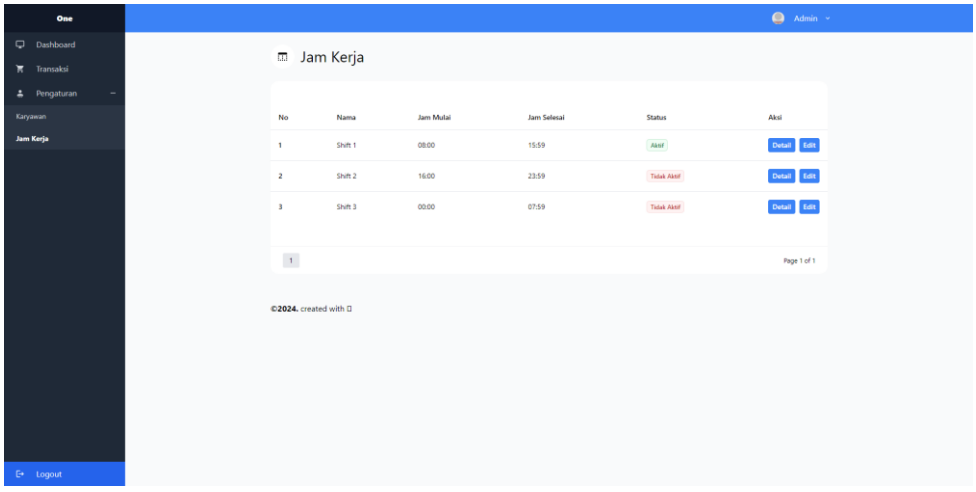
Pada tampilan halaman karyawan, admin bisa membuat, mengubah dan melihat data yang saat ini ada. Untuk menambah karyawan, admin harus mengisi data karyawan. Data karyawan yang harus diisi adalah ID Karyawan, Nama, Email, Password, No. Handphone, Jam Kerja, Alamat, dan Status. Beda halnya dengan mengubah karyawan, data yang diperlu diisi hanya sebatas Jam Kerja dan Status Karyawan. Setelah membuat atau memperbaharui data, admin bisa melihat informasi karyawan dengan menekan tombol detail.

Berikut adalah *source code* untuk halaman karyawan:

```
async getUsers(query: any, token: string): Promise<object> {
    const data: any = jwtDecode(token.replace('Bearer ', ''));
    let officeHours = [];
    const newResponse: any = [];
    const master: any = await fetch(`${process.env.SERVICE_MASTER}/api/office-hours`);
    if (master.ok) {
      officeHours = await master.json();
    }
    const response = await User.find({ _id: { $ne: data.id } });
    response.map(async (x: any) => {
      const data = officeHours.data.find((y: any) => JSON.stringify(y._id) ===
JSON.stringify(x._doc.officeHoursId));
      newResponse.push({ ...x._doc, officeHour: data });
      return { ...x, officeHour: data };
    });

    return newResponse;
  }
```

Segmen Program 4. 7 *Source Code* Karyawan

Selanjutnya adalah tampilan dari halaman jam kerja yang ada pada admin/pemilik:



Gambar 4. 8 Tampilan Halaman Jam Kerja

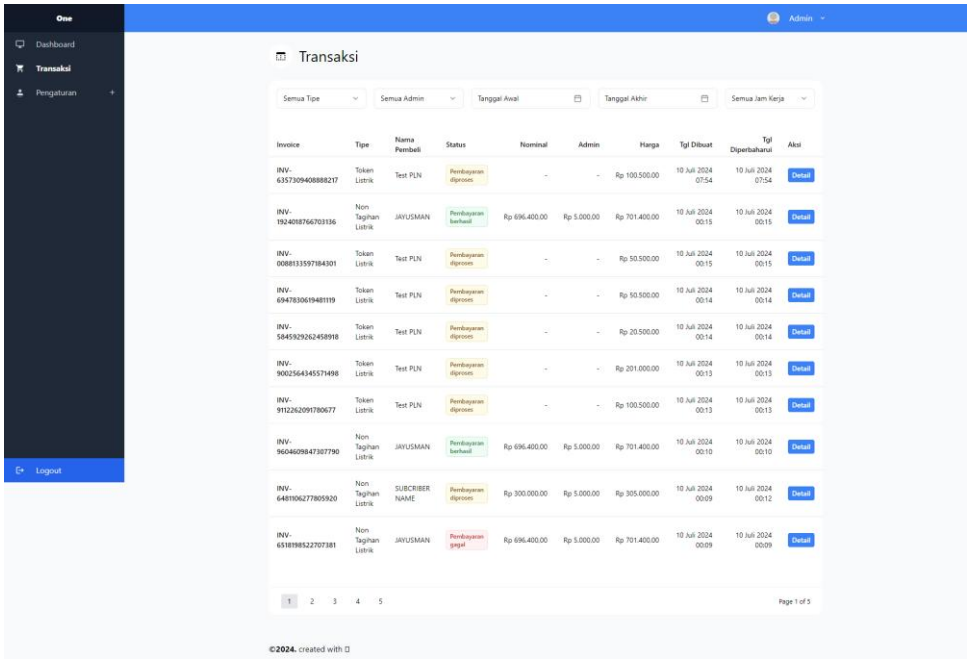
Tampilan selanjutnya adalah tampilan jam kerja. Pada tampilan ini hanya memiliki dua akses yaitu melihat dan memperbaharui data. Admin bisa melihat data dengan menekan tombol detail, untuk mengubah data hanya membutuhkan mengubah status Jam Kerja saja.

Berikut adalah *source code* untuk halaman jam kerja:

```
async getOfficeHours(query: any): Promise<object> {
   let filter: any = {};
   if (query.status) {
     filter.active = query.status;
   }
   const response = await OfficeHours.find({
     ...filter,
   });

   return response;
 }
```

Segmen Program 4. 8 *Source Code* Jam Kerja

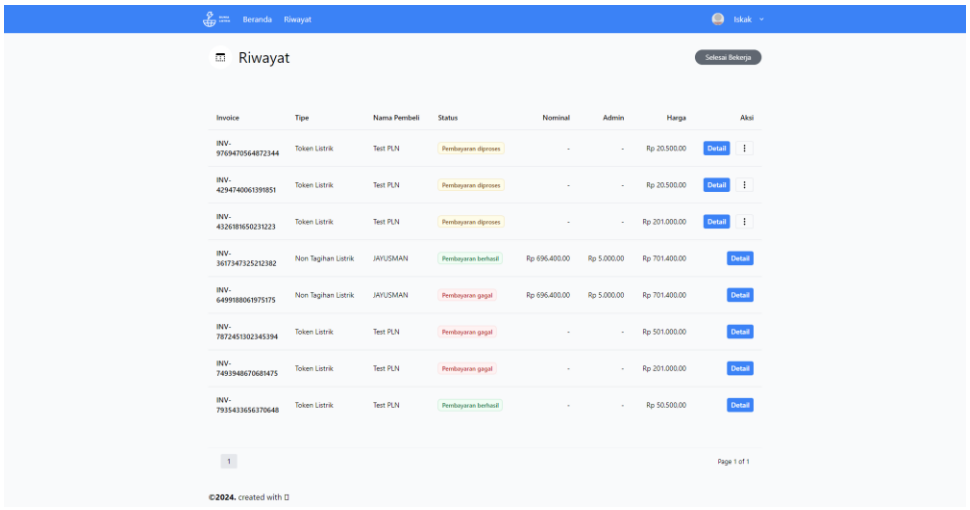
Selanjutnya adalah tampilan dari halaman transaksi yang ada pada admin/pemilik:



Gambar 4. 9 Tampilan Halaman Transaksi pada Admin

Tampilan halaman transaksi pada admin, menampilkan data transaksi yang telah di proses oleh karyawan secara detail disbanding dengan yang ada pada di dashboard. *Dashboard* merupakan kesimpulan dari halaman transaksi yang disederhanakan. Tampilan transaksi juga memiliki *filter* yang dimiliki oleh tampilan *dashboard*. *Filter* yang ada pada halaman dashboard ini terdiri dari tipe produk, karyawan, tanggal mulai, tanggal selesai dan jam kerja. Admin bisa mengatur data sesuai yang diharapkan, untuk tipe produk akan menyaring transaksi berdasarkan tipe produk yang ada. *Filter* karyawan berfungsi untuk menyaring data berdasarkan karyawan yang memproses transaksi. Disisi lain bisa di saring berdasarkan *range* tanggal yang diinginkan dan transaksi yang diproses berdasarkan jam kerja.

Sedangkan disisi karyawan tampilan halaman transaksi yang ada pada karyawan:



Gambar 4. 10 Tampilan Halaman Riwayat pada Karyawan

Tampilan transaksi yang ada pada karyawan dinamakan halaman riwayat, tampilan ini tidak memiliki *filter* namun sudah tersaring dengan karyawan yang melakukan transaksi. Data yang muncul pada halaman ini dipastikan mengerucut pada karyawan yang sudah masuk kedalam aplikasi.

Berikut adalah *source code* yang digunakan pada transaksi disisi karyawan maupun pemilik:

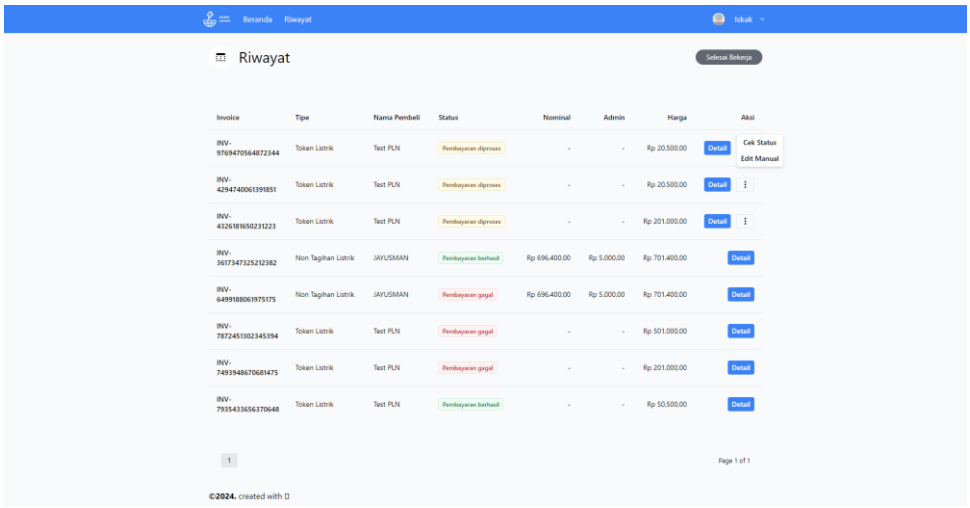
```
  async getTransactions(query: any, token: string): Promise<object> {
    let data;
    let filter: any = {};
    const dataToken: any = jwtDecode(token.replace('Bearer ', ''));
    switch (query.type) {
      case 'all':
        filter.product_code = { $ne: null };
        break;
      case 'PREPAID':
        filter.product_code = { $nin: ['PLNPOSTPAID', 'PLNNONTAG'] };
        break;
      default:
        filter.product_code = query.type;
        break;
    }
    if (query.start_date && query.end_date) {
      filter.createdAt = {
        $gte: new Date(new Date(query.start_date).setHours(0, 0, 0, 0)),
        $lt: new Date(new Date(query.end_date).setHours(23, 59, 59, 999)),
      };
    }
    if (query.office_hour !== 'all' && dataToken.userType === 'admin') {
      const master: any = await fetch(`${process.env.MASTER_SERVICE}/api/office-
hours/${query.office_hour}`);
      const dataMaster = await master.json();


      filter['time.hour'] = {
        $gte: Number(+dataMaster.data.startAt.substr(11, 2)),
        $lt: Number(+dataMaster.data.endAt.substr(11, 2)) === 0 ? 24 :
Number(+dataMaster.data.endAt.substr(11, 2)),
      };
    }
    if (query.user_id !== 'all' && dataToken.userType !== 'user') {
      filter.user_id = query.user_id;
    }
```

Segmen Program 4. 9 *Source Code* Transaksi

```
  if (dataToken.userType === 'user') {
    data = Transaction.find({ user_id: dataToken.id, ...filter }).sort({ createdAt: -1 });
  } else {
    data = Transaction.aggregate([
      { $set: { time: { $dateToParts: { date: '$createdAt' } } } },
      { $match: filter },
      { $sort: { createdAt: -1 } },
    ]);
  }
  return data;
}
```

Segmen Program 4. 10 *Source Code* Transaksi II

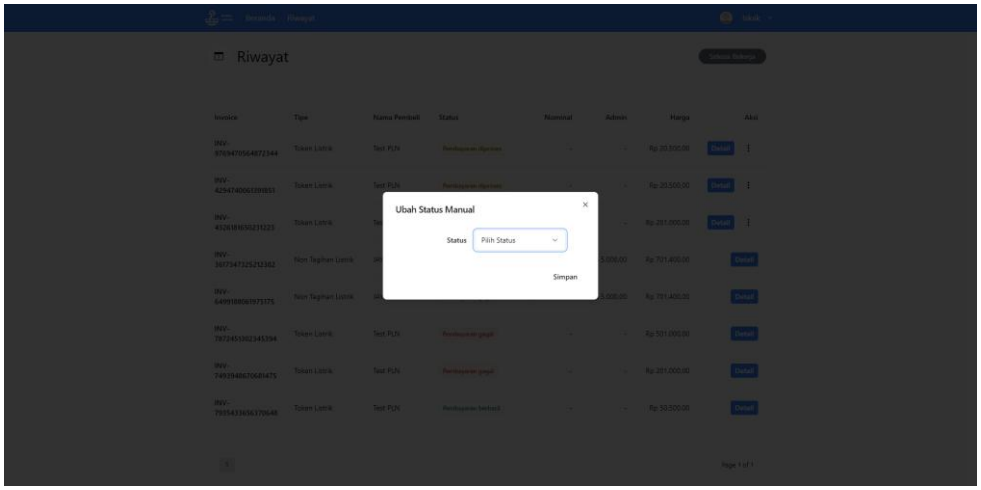
Untuk transaksi disisi karyawan terdapak aksi untuk *update* manual, maupun cek status ke pihak ketiga, berikut tampilannya:



Gambar 4. 11 Tampilan Aksi Cek Status dan Edit Status Manual

Tampilan cek status digunakan oleh karyawan untuk mengetahui status terbaru dari pihak ketiga. Aksi ini hanya muncul ketika status dalam proses, untuk melihat status dari pihak ketiga memerlukan data yang harus dikirimkan ke pihak

ketiga tersebut. Data yang dikirimkan tidak ditampilkan di form melainkan dikirim melalu *headers* pada saat melakukan *request* ke pihak ketiga. Untuk tipe produk prabayar data yang dikirimkan *username*, ref_id, dan sign. *Username* didapat ketika mendaftarkan diri ke pihak ketiga, untuk ref_id merupakan kode unik yang ada pada transaksi aplikasi, sedangkan *sign* merupakan hasil enkripsi dari kombinasi *username*, *api key* dan ref_id. Kombinasi ini di enkripsi menggunakan md5 sesuai dokumentasi dari mereka. Lain halnya dengan tipe produk pascabayar, yang dikirimkan *commands*, *username*, ref_id dan *sign*. *Commands* yang dikirim adalah "checkstatus", untuk perintah lainnya bisa dilihat pada dokumentasi pihak IAK. Untuk kombinasi *sign* diperlukan data *username*, *api key* dan kata "cs", seperti pada prabayar kombinasi ini di enkripsi menggunakan md5.



Gambar 4. 12 Tampilan Edit Status Manual

Pada tampilan edit status manual, fitur ini digunakan untuk mencegah adanya kendala pada pihak ketiga. Dimana kendala yang sering terjadi adalah di pihak ketiga sudah berubah statusnya, namun ketika admin cek status dia

mendapatkan *response error*. Admin bisa mengubah statusnya menyamakan dengan *dashboard* pihak ketiga yang dimana status bisa diubah menjadi sukses maupun gagal.

Berikut adalah source code untuk cek status maupun *update* manual:

```
async updateManualTransaction({ type, status }: any, { id }: any): Promise<object> {
  if (type === 'prepaid') {
    await Transaction.updateOne(
      { invoice: id },
      {
        $set: {
          status,
          rc: status === 1 ? '00' : '07',
        },
      },
    );
```

Segmen Program 4. 11 *Source Code Update* Manual Transaksi

```
async updateTransaction({ type }: any, { id }: any): Promise<object> {
  if (type === 'prepaid') {
    const response: any = await fetch(`${process.env.PREPAID_SERVICE}/api/check-status`, {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({
        username: process.env.API_USERNAME,
        ref_id: id,
        sign: md5(process.env.API_USERNAME + process.env.API_KEY + id),
      }),
    }).then((res) => res.json());
    const responseDetail: any = await
fetch(`${process.env.MASTER_SERVICE}/api/product/inquiry-pln`, {
```

Segmen Program 4. 12 *Source Code* Cek Status

```
    }).then((res) => res.json());
    const responseDetail: any = await
fetch(`${process.env.MASTER_SERVICE}/api/product/inquiry-pln`, {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({
        customer_id: response.data.customer_id,
      }),
    }).then((res) => res.json());
    await Transaction.updateOne(
     { invoice: id },
     {
       $set: {
         customer_id: response.data.customer_id,
         product_code: response.data.product_code,
         status: response.data.status,
         price: response.data.price,
         rc: response.data.rc,
         tr_id: response.data.tr_id,
         info: JSON.stringify({ customer: responseDetail.data.data, data: response.data }),
       },
     },
    );
    const data = await Transaction.findOne({ invoice: id });
    return data;
  }
  const response: any = await fetch(`${process.env.POSTPAID_SERVICE}/api/v1/bill/check`, {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({
      commands: 'checkstatus',
      username: process.env.API_USERNAME,
      ref_id: id,
      sign: md5(process.env.API_USERNAME + process.env.API_KEY + 'cs'),
```

Segmen Program 4. 13 *Source Code* Cek Status II

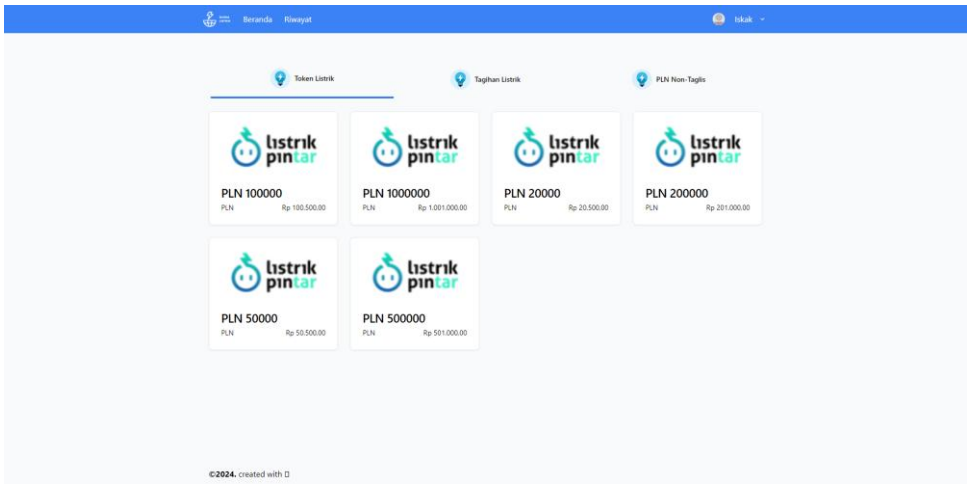
```
  }),
}).then((res) => res.json());
if (response.data.response_code === '00') {
  await Transaction.updateOne(
    { invoice: id },
    {
      $set: {
        customer_id: response.data.hp,
        product_code: response.data.code,
        status: response.data.status || 3,
        price: response.data.price,
        rc: response.data.response_code,
        tr_id: response.data.tr_id,
        info: JSON.stringify(response.data),
        admin: response.data.admin,
        nominal: response.data.nominal,
      },
    },
  );
} else {
  await Transaction.updateOne(
    { invoice: id },
    {
      $set: {
        customer_id: response.data.hp,
        product_code: response.data.code,
        status: response.data.status || 3,
        price: response.data.price,
        rc: response.data.response_code,
        tr_id: response.data.tr_id,
        admin: response.data.admin,
        nominal: response.data.nominal,
      },
    },
```

Segmen Program 4. 14 *Source Code* Cek Status III

```
    );
  }
  const data = await Transaction.findOne({ invoice: id });
  return data;
}
```

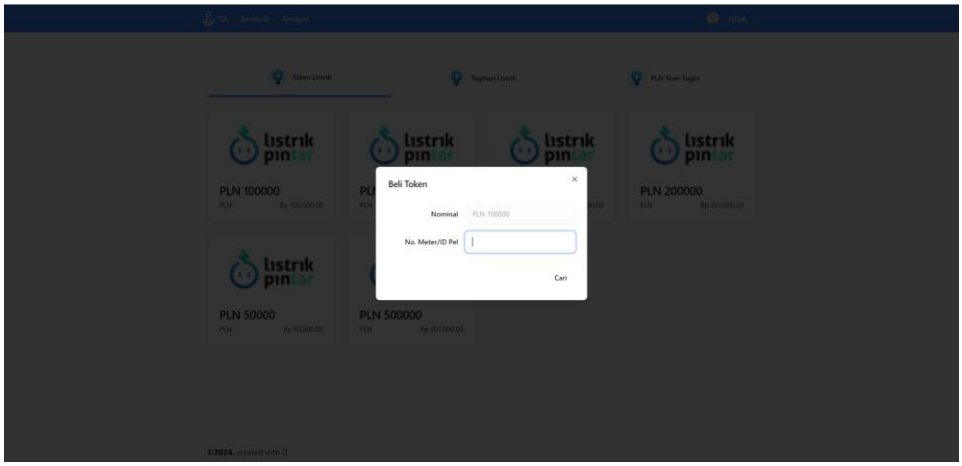
Segmen Program 4. 15 *Source Code* Cek Status IV

Di karyawan ada halaman produk untuk melakukan transaksi pembelian listrik. Dibagi menjadi 3 tipe yaitu, token listrik, tagihan listrik, dan PLN Non-Taglis:



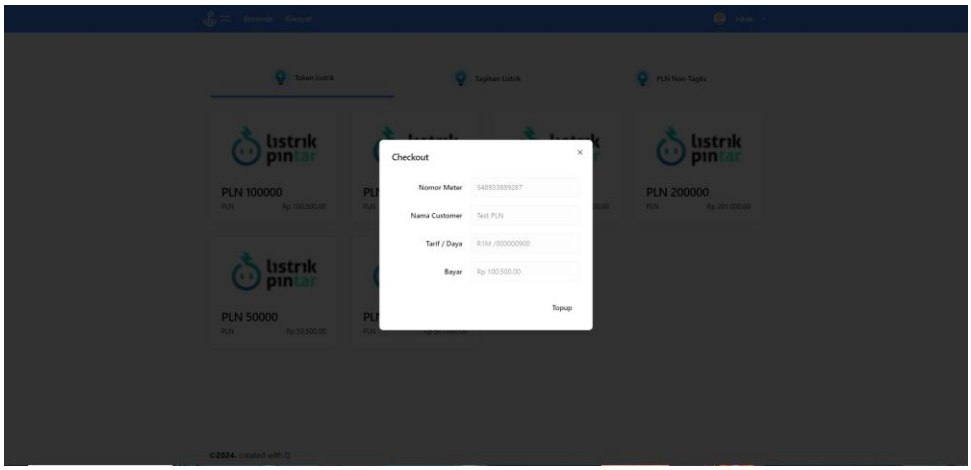
Gambar 4. 13 Tampilan Produk Token Listrik

Tampilan produk token listrik ini didapatkan melalui *API request* pihak ketiga. Data yang dikirim untuk mendapatkan daftar produknya adalah *username*, status, dan *sign*. Status yang dikirim merupakan status produk dari pihak ketiga, isi dari status diantaranya “all”, “active”, “non active”. Untuk kombinasi *sign* berisi data *username*, *api key* dan “pl” yang dienkripsi menjadi md5.



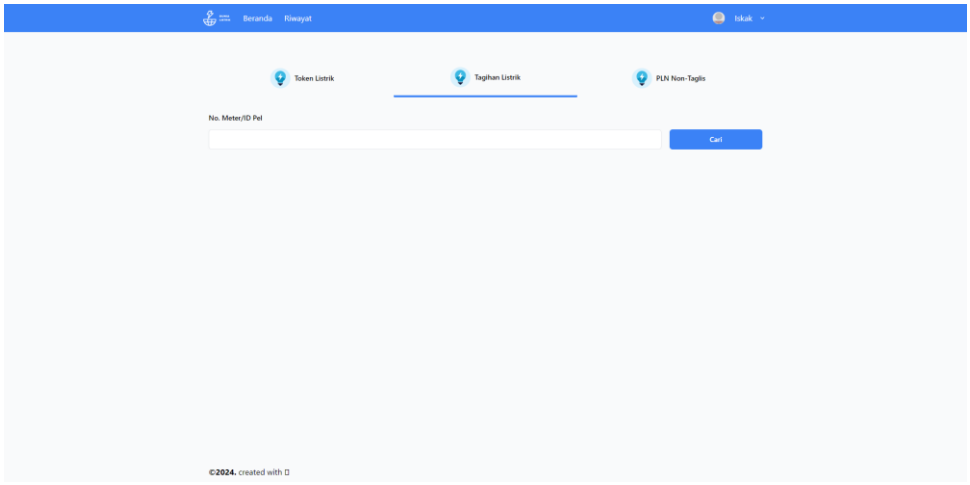
Gambar 4. 14 Tampilan Pencarian Data Pembeli

Kemudian karyawan memilih dari daftar produk yang ada akan menampilkan form untuk mengisi no meter dari pembeli. Dari no meter kemudian menekan tombol cari akan mendapatkan info dari data pembeli tersebut. Ketika mencari data pembeli yang harus dikirimkan ke pihak ketiga merupakan *username*, customer_id, dan *sign*.

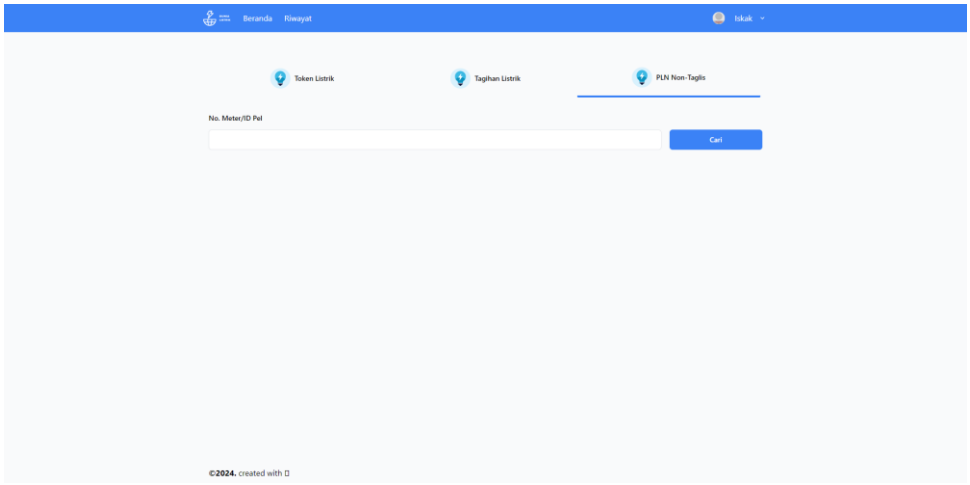


Gambar 4. 15 Tampilan Data Pembeli Prabayar

Data customer_id merupakan hasil dari input karyawan pada kolom no meter. *Sign* yang dikirim kan merupakan kombinasi dari *username*, *api key*, dan customer_id yang dienkripsi oleh md5. Diakhir proses tersebut bisa melakukan proses topup yang dimana data yang dikirim *username*, ref_id, customer_id, product_code, dan sign. Ref_id disini dibuat oleh sistem aplikasi yang digunakan untuk transaksi data antara pihak ketiga dan aplikasi, untuk product_code didapatkan dari produk yang dipilih pada tahap awal. Kombinasi *sign* adalah *username*, *api key* dan ref_id yang dienkripsi oleh md5.

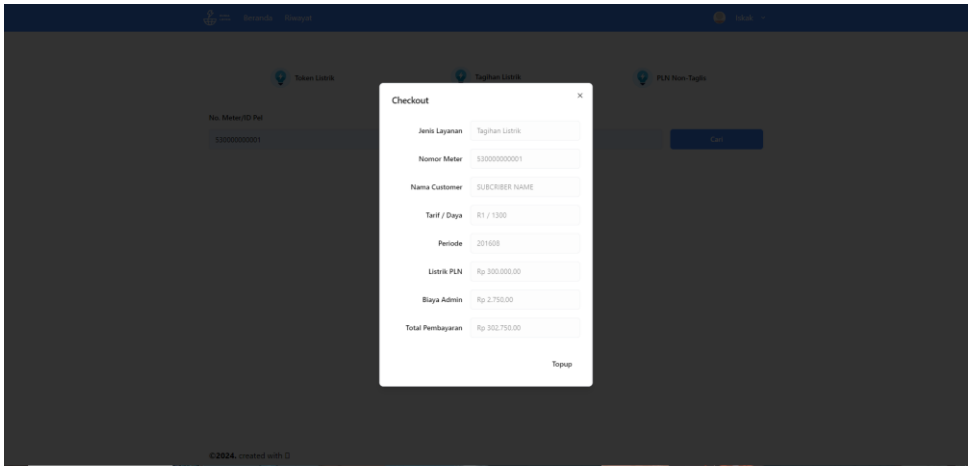


Gambar 4. 16 Tampilan Produk Tagihan Listrik

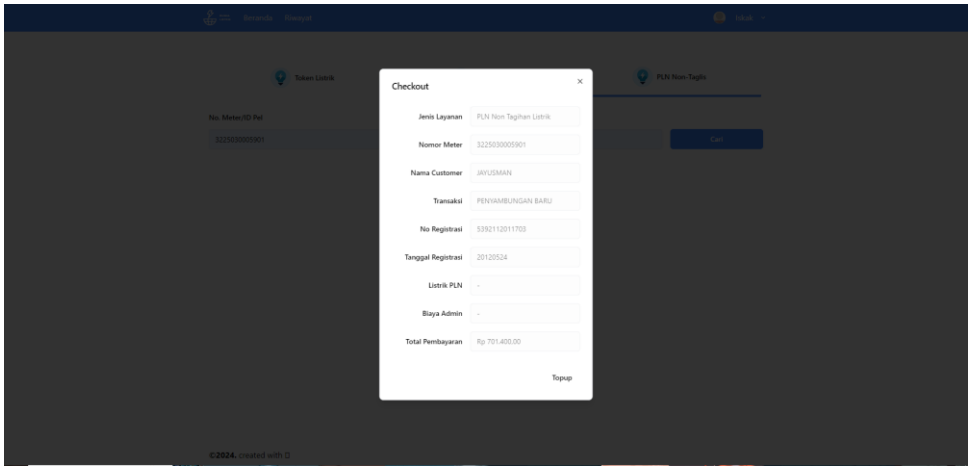


Gambar 4. 17 Tampilan Produk PLN Non Taglis

Pada tampilan produk PLN non taglis dan tagihan listrik menggunakan API *request* yang sama, perbedaannya ada pada *code* yang dikirm. Karyawan melakukan pembelian transaksi pascabayar dengan mengisi no meter pelanggan atau pembeli. Data yang dikirim untuk mencari data pelanggan merupakan *commands*, *username*, *code*, hp, ref_id, dan *sign*. *Commands* dikirim menggunakan "inq-pasca", *code* diisi berdasarkan *code* tagihan listrik ataupun non tagihan listrik, hp berisi no meter yang dimasukkan oleh karyawan, ref_id dibuat oleh sistem aplikasi dan *sign* merupakan hasil enkripsi md5 dari *username*, *api key* dan ref_id.



Gambar 4. 18 Tampilan Data Pembeli Produk Tagihan Listrik



Gambar 4. 19 Tampilan Data Pembeli Produk PLN Non Taglis

Setelah itu karyawan bisa memproses pembelian listrik. Data yang dikirim untuk memprosesnya adalah *commands*, *username*, tr_id, dan *sign*. *Commands* diisi dengan "pay-pasca", tr_id merupakan hasil dari pencarian data yang dilakukan sebelumnya, dan *sign* merupakan hasil enkripsi md5 dari *username*, *api key*, dan tr_id.

Berikut adalah *source code* yang digunakan untuk mendapatkan ketiga tipe produk ini:

```
async getPostpaids(data: Partial<IPostpaid>): Promise<object> {
   const response: any = await
fetch(`${process.env.POSTPAID_SERVICE}/api/v1/bill/check/${data.type}`, {
     method: 'POST',
     headers: {
       'Content-Type': 'application/json',
     },
     body: JSON.stringify({
       commands: 'pricelist-pasca',
       username: process.env.API_USERNAME,
       sign: md5(process.env.API_USERNAME + process.env.API_KEY + 'pl'),
       status: data.status,
     }),
   }).then((res) => res.json());

   return response;
 }
```

Segmen Program 4. 16 *Source Code* Produk Prabayar

Pada segmentasi diatas digunakan untuk mendapatkan data listrik pascabayar terhadap pihak ketiga dengan mengirimkan data yang diperlukan oleh pihak ketiga, diantara lain *commands*, *username*, *sign*, dan status. Disini juga digunakan untuk mendapatkan tipe pascabayar yang berupa tagihan listrik dan juga PLN Non-taglis. Data yang ada dikirim sesuai format yang diberikan, *commands* diisi menggunakan *pricelist-pasca*, *username* diisi dengan *username* yang kita miliki pada pihak ketiga, *sign* merupakan kombinasi *username* + *api_key* + pl yang di enkripsi menggunakan md5, dan status berisi status yang ada (*active, non active*).

```
async getPrepaids(data: Partial<IPrepaid>): Promise<object> {
    const response: any = await
fetch(`${process.env.PREPAID_SERVICE}/api/pricelist/${data.type}/${data.operator}`, {
      method: 'POST',
      headers: {'Content-Type': 'application/json'},
      body: JSON.stringify({
        username: process.env.API_USERNAME,
        sign: md5(process.env.API_USERNAME + process.env.API_KEY + 'pl'),
        status: data.status,
      }),
    }).then((res) => res.json());

    return response;
  }
```

Segmen Program 4. 17 *Source Code* Produk Pascabayar

Segmentasi program diatas merupakan fungsi yang digunakan untuk mendapatkan data token yang dijual saat ini. Data yang ditampilkan biasanya dalam bentuk token yang sudah memiliki nominal tertentu. Untuk mendapatkan data tersebut aplikasi memerlukan *username*, *sign*, dan status ketika melakukan request terhadap pihak IAK. *Username* merupakan *username* yang kita miliki di pihak ketiak, *sign* merupakan kombinasi dari *username* + *api_key* + pl yang di enkripsi menggunakan md5, dan status yang dimiliki oleh pihak ketiga (*active, non active*).

```
async inquiryPln(data: Partial<IInquiryPln>): Promise<object> {
  const response: any = await fetch(`${process.env.PREPAID_SERVICE}/api/inquiry-pln`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({
      username: process.env.API_USERNAME,
      sign: md5(process.env.API_USERNAME + process.env.API_KEY + data.customer_id),
      customer_id: data.customer_id,
    }),
  }).then((res) => res.json());

  return response;
}
```

Segmen Program 4. 18 *Source Code* Mencari Data Pelanggan

Segmentasi program diatas digunakan untuk mencari data pembeli berdasarkan no meter yang dimasukkan oleh karyawan untuk memastikan data pembeli itu benar. Pada bagian ini memerlukan data *username, sign* dan *customer_id*. Seperti pada sebelumnya *username* merupakan *username* yang ada pada pihak ketiga, untuk *sign* merupakan kombinasi dari *username* + *api_key* + *customer_id* yang dienkripsi md5, dan untuk *customer_id* merupakan data yang dimasukkan oleh karyawan pada saat proses pengecekan no meter.

Kemudian step terakhir ada *source code* untuk membuat transaksi prabayar atau token listrik.

```
async transaction(data: Partial<ITransaction>, token: string): Promise<object> {
    const user_id: any = jwtDecode(token.replace('Bearer ', ''));
    var voucher_codes = require('voucher-code-generator');
    const ref_id =
      'INV-' +
      voucher_codes.generate({
        length: 16,
        count: 1,
        charset: '0123456789',
      })[0];

    const response: any = await fetch(`${process.env.PREPAID_SERVICE}/api/top-up`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        username: process.env.API_USERNAME,
        ref_id: ref_id,
        customer_id: data.customer_id,
        product_code: data.product_code,
        sign: md5(process.env.API_USERNAME + process.env.API_KEY + ref_id),
      }),
    }).then((res) => res.json());
    const responseDetail: any = await
fetch(`${process.env.MASTER_SERVICE}/api/product/inquiry-pln`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
```

Segmen Program 4. 19 *Source Code* Membuat Transaksi Prabayar

```
      customer_id: data.customer_id,
    }),
  }).then((res) => res.json());
  const transaction = new Transaction({
    invoice: response.data.ref_id,
    customer_id: response.data.customer_id,
    product_code: response.data.product_code,
    status: response.data.status,
    price: response.data.price,
    rc: response.data.rc,
    tr_id: response.data.tr_id,
    info: JSON.stringify({ customer: responseDetail.data.data, data: response.data }),
    url: '',
    user_id: user_id?.id,
  });

  await transaction.save();
  return transaction;
}
```

Segmen Program 4. 20 *Source Code* Membuat Transaksi Prabayar II

Pada segmentasi program 4.19 dan 4.20 merupakan proses pembuatan transaksi pada tipe prabayar atau token listrik. Pada proses ini membuat kode *invoice* yang sama digunakan sebagai jembatan dua pihak yaitu pihak IAK dan juga pihak aplikasi. Data yang dikirim ke pihak ketiga untuk membuat transaksi diantara lain *username, ref_id, customer_id, product_code* dan *sign*. *Username* merupakan *username* yang dimiliki pada IAK, *ref_id* merupakan kode *invoice* yang dibuat sistem dan sebagai acuan terhadap dua pihak, *customer_id* merupakan no meter pembeli yang dimasukkan oleh karyawan, *product_code* merupakan kode produk yang ditampilkan pada list produk di awal, dan *sign* merupakan kombinasi dari

username + *api_key* + *ref_id* yang dienkripsi oleh md5. Kemudian setelah mendapatkan *response* dari pihak IAK, data yang didapat diproses dan dimasukkan kedalam database yang digunakan.

Berikut merupakan *source code* untuk membuat transaksi pascabayar atau tagihan listrik maupun PLN Non-taglis.

```
async transactionPostpaid(data: Partial<ITransactionPostpaid>, token: string): Promise<object> {
    const user_id: any = jwtDecode(token.replace('Bearer ', ''));
    var voucher_codes = require('voucher-code-generator');
    let tmpData = {};
    let transaction;

    if (data.commands === 'inq-pasca') {
      const ref_id =
        'INV-' +
        voucher_codes.generate({
          length: 16,
          count: 1,
          charset: '0123456789',
        })[0];
      tmpData = {
        commands: data.commands,
        username: process.env.API_USERNAME,
        code: data.code,
        hp: data.hp,
        ref_id: ref_id,
        sign: md5(process.env.API_USERNAME + process.env.API_KEY + ref_id),
      };

      const response: any = await fetch(`${process.env.POSTPAID_SERVICE}/api/v1/bill/check`, {
        method: 'POST',
```

Segmen Program 4. 21 *Source Code* Membuat Transaksi Pascabayar

```
    headers: { 'Content-Type': 'application/json', },
    body: JSON.stringify(tmpData),
  }).then((res) => res.json());
  transaction = new Transaction({
    invoice: response.data.ref_id,
    customer_id: response.data.hp,
    product_code: response.data.code,
    status: response.data.status || '3',
    price: response.data.price,
    rc: response.data.response_code,
    tr_id: response.data.tr_id,
    info: JSON.stringify(response.data),
    url: '',
    admin: response.data.admin,
    nominal: response.data.nominal,
    user_id: user_id?.id,
  });

  await transaction.save();
```

Segmen Program 4. 22 *Source Code* Membuat Transaksi Pascabayar II

Pada segmentasi program 4.21 sampai 4.22 merupakan proses pembuatan taransaksi pascabayar atau tipe tagihan listrik dan PLN Non tag-lis. Proses transaksi ini membuat kode *invoice* yang sama digunakan sebagai jembatan dua pihak yaitu pihak IAK dan juga pihak aplikasi. Proses pembuatannya melalui dua tahap yaitu *inqury* dan *pay* (pembayaran). Pada tahap *inquiry,* akan mendapatkan data dari pembeli dan total jumlah tagihan yang dimiliki beserta biaya admin. Selanjutnya data yang didapat dari IAK akan disimpan kedalam *database* aplikasi menggunakan status *pending* atau sedang dalam proses. Untuk mendapatkan data tersebut, perlu mengirimkan beberapa data diantara lain *commands, username, code, hp, ref_id*

dan *sign*. *Commands* dikirim menggunakan *inq-pasca*, *username* dikirim menggunakan *username* yang digunakan pada IAK, *code* dikirim menggunakan kode pascabayar tagihan listrik atau PLN Non taglis, *hp* merupakan no meter dari pembeli yang dimasukkan oleh karyawan, *ref_id* merupakan kode *invoice* yang dibuat sistem sebagai acuan kedua belah pihak, dan *sign* merupakan kombinasi *username* + *api_key* + *ref_id* yang dienkripsi oleh md5.

```
} else if (data.commands === 'pay-pasca') {
  tmpData = {
    commands: data.commands,
    username: process.env.API_USERNAME,
    tr_id: data.tr_id,
    sign: md5(process.env.API_USERNAME + process.env.API_KEY + data.tr_id),
  };

  const response: any = await fetch(`${process.env.POSTPAID_SERVICE}/api/v1/bill/check`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(tmpData),
  }).then((res) => res.json());

  if (response.data.response_code === '00') {
    transaction = Transaction.updateOne(
      { tr_id: data.tr_id },
      {
        $set: {
          invoice: response.data.ref_id,
          customer_id: response.data.hp,
          product_code: response.data.code,
```

Segmen Program 4. 23 *Source Code* Membuat Transaksi Pascabayar III

```
            status: response.data.status || 3,
            price: response.data.price,
            rc: response.data.response_code,
            tr_id: response.data.tr_id,
            url: '',
            admin: response.data.admin,
            nominal: response.data.nominal,
            user_id: user_id?.id,
          },
        },
      );
    }
  }

  return transaction;
}
```

Segmen Program 4. 24 *Source Code* Membuat Transaksi Pascabayar IV

Pada segmentasi program 4.23 sampai 4.24 merupakan proses tahan kedua yaitu *pay* (pembayaran). Pada proses *pay* hanya memperbaharui status pada transaksi yang sudah dibuat pada proses *inquiry*. Untuk mendapatkan data yang terbaru, perlu mengirimkan *commands*, *username*, *tr_id* dan *sign* ke pihak IAK. *Commands* dikirim menggunakan *pay-pasca*, *username* dikirim menggunakan *username* yang digunakan pada IAK, *tr_id* merupakan kode *invoice* yang dikirim sistem sebagai acuan kedua belah pihak, dan *sign* merupakan kombinasi *username* + *api_key* + *rtr_id* yang dienkripsi oleh md5.

4.3 Uji Coba

Pada titik ini, sistem diuji untuk mengidentifikasi kesalahan atau kekurangan dalam penerapan arsitektur *microservices* dalam pembelian token listrik. Pengujian dilakukan untuk memverifikasi bahwa sistem yang dikembangkan telah memenuhi spesifikasi dan kriteria yang ditetapkan dalam tahap perencanaan awalnya.

Pengujian sistem *load test* saat ini menggunakan metode pengujian *benchmarking* dan *error http service*. Tujuan pengujian *benchmarking* adalah memperoleh penilaian secara objektif karena diuji menggunakan *benchmarking tools*, sedangkan tujuan untuk pengujian *error http service* adalah untuk mengetahui jika ada *error* pada arsitektur *microservices*, apakah sistem tetap berjalan apakah akan menghentikan semua *service*.

Pengujian *benchmarking* pada arsitektur microservices digunakan untuk mengetahui apakah dengan arsitektur *microservices* dapat menampung request yang besar. Pengujian dilakukan dengan menggunakan Fastify Benchmarking Tools, yaitu *Modern HTTP benchmarking tool* untuk http *request* pada Fastify. Dengan menjalankan *command* ***"benchmarking"*** pada *terminal / command prompt, command* tersebut akan menjalankan *server* dengan spesifikasi yang telah dijabarkan pada rancangan pengujian. Hasil pengujian *benchmarking* pada arsitektur *microservices* dengan skenario yang telah ditentukan memperoleh hasil seperti dibawah ini.

```
{
  "url": "http://127.0.0.1:3000",
  "connections": 100,
  "sampleInt": 1000,
  "pipelining": 10,
  "duration": 60.01,
  "samples": 60,
  "start": "2024-08-15T20:01:33.040Z",
  "finish": "2024-08-15T20:02:33.054Z",
  "errors": 0,
  "timeouts": 0,
  "mismatches": 0,
  "non2xx": 0,
  "resets": 0,
  "1xx": 0,
  "2xx": 3986665,
  "3xx": 0,
  "4xx": 0,
  "5xx": 0,
  "statusCodeStats": {
    "200": {
      "count": 3986665
    }
  },
  "latency": {
    "average": 14.55,
    "mean": 14.55,
    "stddev": 13.67,
    "min": 1,
    "max": 164,
    "totalCount": 3986665
  },
  "requests": {
    "average": 66445.07,
    "mean": 66445.07,
    "stddev": 4584.84,
    "min": 46177,
    "max": 70824,
    "total": 3986665,
    "sent": 3987665
  },
  "throughput": {
    "average": 12491298.14,
    "mean": 12491298.14,
    "stddev": 862558.99,
    "min": 8681276,
    "max": 13314912,
    "total": 749493020,
  },
  "server": "fastify"
}
```


Gambar 4. 20 Hasil *Load Test* Aplikasi *Microservice* Skenario 1

Pada gambar diatas, dijalankan dengan spesifikasi 10 *pipelines* dan 100 *connections* dan akan di jalankan selama 60 detik. Framework fastify yang dipakai di arsitektur *microservices* pembelian token listrik mendapatkan *request* 3.000.000 *requests per second* yang mana dengan *requests* tersebut sangat besar untuk menampung data keluar masuk.

```
{
  "url": "http://127.0.0.1:3000",
  "connections": 500,
  "sampleInt": 1000,
  "pipelining": 100,
  "duration": 60.43,
  "samples": 60,
  "start": "2024-08-15T19:48:00.580Z",
  "finish": "2024-08-15T19:49:01.005Z",
  "errors": 5434,
  "timeouts": 0,
  "mismatches": 0,
  "non2xx": 0,
  "resets": 0,
  "1xx": 0,
  "2xx": 3961547,
  "3xx": 0,
  "4xx": 0,
  "5xx": 0,
  "statusCodeStats": {
    "200": {
      "count": 3961547
    }
  },
  "latency": {
    "average": 7726.27,
    "mean": 7726.27,
    "stddev": 7407.51,
    "min": 1,
    "max": 26175,
    "totalCount": 3961547
  },
  "requests": {
    "average": 66029.34,
    "mean": 66029.34,
    "stddev": 10444.05,
    "min": 35700,
    "max": 76800,
    "total": 3961547,
    "sent": 4554947
  },
  "throughput": {
    "average": 12412825.6,
    "mean": 12412825.6,
    "stddev": 1963354.9,
    "min": 6711600,
    "max": 14438400,
    "total": 744770836,
  },
  "server": "fastify"
}
```


Gambar 4. 21 Hasil *Load Test* Aplikasi *Microservice* Skenario 2

Pada gambar diatas, dijalankan dengan spesifikasi 100 *pipelines* dan 500 *connections* dan akan di jalankan selama 60 detik. Framework fastify yang dipakai di arsitektur *microservices* pembelian token listrik mendapatkan *request* 3.000.000 *requests per second* yang mana dengan *requests* tersebut sangat besar untuk menampung data keluar masuk.

```
{
  "url": "http://127.0.0.1:3000",
  "connections": 1000,
  "sampleInt": 1000,
  "pipelining": 100,
  "duration": 60.86,
  "samples": 53,
  "start": "2024-08-15T19:53:24.969Z",
  "finish": "2024-08-15T19:54:25.832Z",
  "errors": 12555,
  "timeouts": 0,
  "mismatches": 0,
  "non2xx": 0,
  "resets": 0,
  "1xx": 0,
  "2xx": 3019492,
  "3xx": 0,
  "4xx": 0,
  "5xx": 0,
  "statusCodeStats": {
    "200": {
      "count": 3019492
    }
  },
  "latency": {
    "average": 18524.63,
    "mean": 18524.63,
    "stddev": 13610.93,
    "min": 94,
    "max": 52110,
    "totalCount": 3019492
  },
  "requests": {
    "average": 56968.34,
    "mean": 56968.34,
    "stddev": 18650.57,
    "min": 6200,
    "max": 79100,
    "total": 3019492,
    "sent": 4374992
  },
  "throughput": {
    "average": 10710431.4,
    "mean": 10710431.4,
    "stddev": 3506758.84,
    "min": 1165600,
    "max": 14870800,
    "total": 567664496,
  },
  "server": "fastify"
}
```


Gambar 4. 22 Hasil Load Test Aplikasi Microservice Skenario 3

Pada gambar diatas, dijalankan dengan spesifikasi 100 *pipelines* dan 1.000 *connections* dan akan di jalankan selama 60 detik. Framework fastify yang dipakai di arsitektur *microservices* pembelian token listrik mendapatkan *request* 3.000.000 *requests per second* yang mana dengan *requests* tersebut sangat besar untuk menampung data keluar masuk.

Pada pengujian tahap kedua, digunakan untuk mencoba keuntungan dari *microservice* yang berupa toleransi kesalahan. Di pengujian ini dilakukan dengan cara mematikan salah satu *service* atau layanan yang ada. Berikut adalah uji coba mematikan salah satu service pada arsitektur *microservices*.

Tabel 4. 1 Hasil Uji Coba Toleransi Arsitektur *Microservices*

Skenario	Hasil uji coba
Mematikan *service user*	Karyawan masih bisa melakukan pembelian listrik, mengakses riwayat pembelian Admin masih bisa mengakses transaksi, mengakses *master* jam kerja
Mematikan *service master*	Karyawan masih bisa melakukan pembelian listrik, mengakses riwayat pembelian Admin masih bisa mengakses transaksi, mengakses karyawan (*user*)
Mematikan *service* transaksi	Karyawan masih bisa *login*, dan *logout* Admin masih bisa mengakses karyawan (*user*), mengakses *master* jam kerja