4. IMPLEMENTASI SISTEM

Bab ini akan membahas tentang implementasi sistem Administrasi berbasis web yang sudah dibuat. Program dibuat pada aplikasi Visual Studio Code menggunakan bahasa pemrograman HTML, Javascript, dan CSS untuk *frontend* dan Express Js untuk *backend*.

4.1 Implementasi Program

Tabel 4.1

Tabel Segmen Program

Proses	Segmen Program	Referensi *Activity Diagram*
Penjualan Barang	Segmen Program 4.1 - 4.2	Tabel 3.5
Perhitungan HPP	Segmen Program 4.3	Tabel 3.7
Pembelian Barang	Segmen Program 4.4	Tabel 3.8
Perhitungan Safety Stock	Segmen Program 4.5	Tabel 3.12
Perhitungan ROP	Segmen Program 4.6	Tabel 3.10
Perhitungan EOQ	Segmen Program 4.7	Tabel 3.14

4.1.1 Proses Penjualan Barang

Segmen Program 4.1 Penjualan Barang

```
async function updateStockAndCreateKartuStok(barangId, kuantitas, salesId, hargaJual,
tanggalPenjualan, hppValue) {
  try {
    // Mendapatkan data produk dari backend untuk mendapatkan nilai stok yang sekarang
    const response = await fetch(productsAPIURL + `/${barangId}`);
    const data = await response.json();
```

```
    const currentStock = parseInt(data.stok);

    // Mengurangi stok berdasarkan input kuantitas
    const newStock = currentStock - kuantitas;

    const calculatedSafetyStock = await safetyStock(barangId); // Ubah nama variabel ini
    const safetyStockValue = calculatedSafetyStock;

    // Melakukan PATCH request untuk mengupdate stok barang
    const patchResponse = await fetch(productsAPIURL + `/${barangId}`, {
      method: 'PATCH',
      headers: {
        'Content-Type': 'application/json'
      },
      body: JSON.stringify({
        stok: newStock,
        safety_stock: safetyStockValue
      })
    });

    if (!patchResponse.ok) {
      throw new Error('Gagal mengupdate stok barang.');
    }

    // Pembuatan kartu stok hanya jika stok berhasil diubah
    const kartuStokResponse = await fetch('http://localhost:3000/kartuStok', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json'
      },
```

```
            body: JSON.stringify({
                refId: salesId,
                barangId: barangId,
                jumlah: kuantitas,
                jumlah_produk: newStock, // Jumlah produk setelah penjualan
                jenis_transaksi: 0, // Jenis transaksi untuk penjualan
                harga: hargaJual,
                hpp: hppValue,
                tanggalTransaksi: tanggalPenjualan
            })
        });

        if (kartuStokResponse.ok) {
            console.log(`Kartu stok berhasil dibuat untuk barang ${barangId}.`);
            Swal.fire({
                title: "Penjualan Disimpan!",
                customClass: {
                    title: 'text-green', // Menentukan kelas CSS khusus untuk judul
                }
            });
            // closeCompleteOrderPopup();
        } else {
            console.error(`Gagal membuat kartu stok untuk barang ${barangId}.`);
            console.error(kartuStokResponse);
        }
    } catch (error) {
        console.error('Error:', error);
        alert('Terjadi kesalahan saat mengupdate stok barang atau membuat kartu stok.');
    }
}
```

Segmen Program 4.2 Penjualan Barang

```
let confirmNoteBtn = document.querySelector("#sales-note-confirm-btn");

confirmNoteBtn.addEventListener("click", async function () {
    const tableRows = document.querySelectorAll("#sales-note-products-table-body tr");
    let pickedDateInput = document.querySelector("#create-sales-date");

    // Pengecekan apakah sudah memilih tanggal nota
    if (pickedDateInput.value === "") {
        alert('Pilih tanggal terlebih dahulu.');
        return;
    }
    // Pengecekan apakah ada barang di dalam tabel
    if (tableRows.length === 0) {
        alert('Tidak ada data dalam nota.');
        return;
    } else {
        document.querySelector("#sales-note-total").innerHTML = totalSales;

        let nomorNota2 = nomorNota;

        console.log('nomorNota2 setelah di tekan:', nomorNota2);
        console.log('totalSales setelah di tekan:', totalSales);
        console.log('addedPickedDateTime setelah di tekan:', addedPickedDateTime);
        try {
            console.log(`addedPickedDateTime saat buat sales: ${addedPickedDateTime}`);
            // POST request untuk penjualan (sales)
            const salesResponse = await fetch(salesAPI, {
```

```
            method: 'POST',
            headers: {
               'Content-Type': 'application/json'
            },
            body: JSON.stringify({ salesId: nomorNota2, total_harga: totalSales,
tanggalTransaksi: addedPickedDateTime }) // sesuaikan dengan data yang diperlukan untuk
penjualan
         });

         if (!salesResponse.ok) {
            throw new Error('Gagal membuat penjualan.');
         }

         // // Mendapatkan nomor nota dari respons penjualan
         // let { nomorNota2 } = await salesResponse.json();

         // Menggunakan nomor nota sebagai salesId dalam setiap item penjualan
         tableRows.forEach(async (row, index) => {
            // Dapatkan data produk dari setiap baris tabel
            const barangId = row.dataset.barangId;
            const hargaJualCell = row.querySelector('td[data-harga-jual-id]');
            const hargaJual = parseFloat(hargaJualCell.dataset.hargaJualId);
            const kuantitasCell = row.querySelector('td[data-kuantitas-id]');
            const kuantitas = parseInt(kuantitasCell.dataset.kuantitasId);

            console.log(`addedPickedDateTime saat buat sales items:
${addedPickedDateTime}`);
            const productHPPResponse = await
fetch(`http://localhost:3000/products/${barangId}`);
            const productHPPData = await productHPPResponse.json();
```

```
            const hppValue = productHPPData.hpp;
            // Persiapkan data untuk dikirim ke backend
            const salesData = {
                barangId: barangId,
                harga_jual: hargaJual,
                kuantitas: kuantitas,
                salesId: nomorNota2, // Menggunakan nomor nota sebagai salesId
            };
            console.log('salesData:', salesData)

            // POST request untuk item penjualan (salesItems)
            const salesItemsResponse = await fetch(salesItemsAPI, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(salesData)
            });

            if (!salesItemsResponse.ok) {
                throw new Error('Gagal menyimpan data penjualan.');
            }

            // Update stok dan hapus baris dari tabel setelah data dikirim ke backend
            await updateStockAndCreateKartuStok(barangId, kuantitas, nomorNota2, hargaJual,
addedPickedDateTime, hppValue);
            row.parentNode.removeChild(row);

            // Jika ini adalah baris terakhir, tampilkan pesan sukses
```

```
            if (index === tableRows.length - 1) {
                console.log('nomor nota disimpan: ', nomorNota);
                nomorNota = ''; // reset nomorNota setelah penggunaan
                totalSales = 0;

                setDefaultDate(pickedDateInput);
                document.querySelector("#sales-note-total").innerHTML = totalSales;
                // Lakukan penyesuaian lain yang diperlukan setelah penjualan berhasil disimpan
            }
        });

        pickedDateInput.placeholder = 'dd/mm/yyyy';
    } catch (error) {
        console.error('Error:', error);
        alert('Terjadi kesalahan saat menyimpan data penjualan.');
    }
  }
});
```

4.1.2 Proses Perhitungan HPP

Segmen Program 4.3 Perhitungan HPP

```
async function hitungHPP(barangId, newStock, hargaBeli, kuantitasDiterima) {
  const productHPPResponse = await fetch(`http://localhost:3000/products/${barangId}`);
  const productHPPData = await productHPPResponse.json();
```

```
    const hppAwal = productHPPData.hpp;
    const stokSekarang = productHPPData.stok;
    const hargaBeliProduk = hargaBeli;
    const stokDiterima = kuantitasDiterima;

    console.log('hppAwal:', hppAwal);
    console.table('stokSekarang', stokSekarang);
    console.log('hargaBeliProduk:', hargaBeliProduk);
    console.log('stokDiterima:', stokDiterima);


    // Menghitung HPP dengan metode rata-rata
    const hpp = Math.round((((hppAwal * stokSekarang) + (hargaBeliProduk * stokDiterima)) / newStock));

    return hpp;
}
```

4.1.3 Proses Pembelian Barang

Segmen Program 4.4 Pembelian Barang

```
confirmNoteBtn.addEventListener("click", async function () {
    const tableRows = document.querySelectorAll("#purchases-note-products-table-body tr");
    let pickedDate = document.querySelector("#create-purchases-date");

    // Pengecekan apakah ada barang di dalam tabel
    if (tableRows.length === 0) {
        Swal.fire({
            icon: 'warning',
            title: 'Peringatan',
            text: 'Tidak ada data dalam nota',
```

```
        position: 'top-end',
        showConfirmButton: false,
        timer: 1500,
        timerProgressBar: true,
        toast: true,
        showClass: {
          popup: 'animate__animated animate__fadeInDown custom-animate'
        },
        hideClass: {
          popup: 'animate__animated animate__fadeOutUp custom-animate'
        }
      });
      return;
    } else {
      document.querySelector("#purchases-note-total").innerHTML = totalPurchases;

      let nomorNota2 = nomorNota;
      try {
        // POST request untuk pembelian (purchases)
        const purchasesResponse = await fetch(purchasesAPI, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({ purchasesId: nomorNota2, nota_supplier: addedNotaSupplier,
total_harga: totalPurchases, supplierId: addedSupplier, tanggalTransaksi: pickedDate.value })
// sesuaikan dengan data yang diperlukan untuk pembelian
        });

        if (!purchasesResponse.ok) {
```

```
            Swal.fire({
                icon: 'error',
                title: 'Error',
                text: error.message,
                position: 'center', // Posisikan di tengah halaman
                showConfirmButton: false,
                timer: 3000,
                timerProgressBar: true,
                showClass: {
                    popup: 'animate__animated animate__fadeInDown custom-animate'
                },
                hideClass: {
                    popup: 'animate__animated animate__fadeOutUp custom-animate'
                }
            });
            throw new Error('Gagal membuat pembelian.');
        }

        // Menggunakan nomor nota sebagai purchasesDetailsId dalam setiap item pembelian
        tableRows.forEach(async (row, index) => {

            console.log('ini index: ', index);
            // Dapatkan data produk dari setiap baris tabel
            const barangId = row.dataset.barangId;
            const kodeBarangCell = row.querySelector('td[data-kode-barang-id]');
            const kodeBarang = kodeBarangCell.textContent;
            const namaBarangCell = row.querySelector('td[data-nama-barang-id]');
            const namaBarang = namaBarangCell.textContent;
            const hargaBeliCell = row.querySelector('td[data-harga-beli-id]');
            const hargaBeli = parseFloat(hargaBeliCell.dataset.hargaBeliId);
```

```
            const kuantitasCell = row.querySelector('td[data-kuantitas-id]');
            const kuantitas = parseInt(kuantitasCell.dataset.kuantitasId);

            // Persiapkan data untuk dikirim ke backend
            const purchasesData = {
                barangId: barangId,
                purchasesId: nomorNota2, // Menggunakan nomor nota sebagai
purchasesDetailsId
                harga_beli: hargaBeli,
                kuantitas_dipesan: kuantitas,
                kuantitas_belum_diterima: kuantitas,
                tanggalTransaksi: pickedDate.value
            };

            console.log(purchasesData);

            try {
                // Cek apakah produk sudah ada di database
                const existingProductResponse = await fetch(`${productsAPIURL}/${barangId}`);
                if (!existingProductResponse.ok) {
                    throw new Error('Gagal memeriksa produk yang ada.');
                }

                const existingProductData = await existingProductResponse.json();

                if (!existingProductData || Object.keys(existingProductData).length === 0) {
                    // Tambahkan kode berikut di bawah komentar 'Produk belum ada, tambahkan
produk baru ke database'
                    console.log('produk belum ada');
                    const idBarangInput = barangId;
```

```
            const kodeBarangInput = kodeBarang;
            const namaBarangInput = namaBarang
            const hargaBeliInput = parseFloat(hargaBeli);

            // Produk belum ada, tambahkan produk baru ke database
            const newProductResponse = await fetch(productsAPIURL, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify({
                    barangId: idBarangInput,
                    kode_barang: kodeBarangInput,
                    nama_barang: namaBarangInput,
                    modal: hargaBeliInput,
                    stok: 0 // Atur stok awal untuk produk baru
                    // tambahkan data lain yang diperlukan untuk produk baru
                })
            });

            console.log('idBarangInput:', idBarangInput);
            console.log('kodeBarangInput:', kodeBarangInput);
            console.log('namaBarangInput:', namaBarangInput);
            console.log('hargaBeliInput:', hargaBeliInput);

            if (!newProductResponse.ok) {
                Swal.fire({
                    icon: 'error',
                    title: 'Error',
                    text: error.message,
```

```
                    position: 'center', // Posisikan di tengah halaman
                    showConfirmButton: false,
                    timer: 3000,
                    timerProgressBar: true,
                    showClass: {
                        popup: 'animate__animated animate__fadeInDown custom-animate'
                    },
                    hideClass: {
                        popup: 'animate__animated animate__fadeOutUp custom-animate'
                    }
                });
                throw new Error('Gagal menambahkan produk baru.');
            }

        } else {
            console.log('produk sudah ada');
        }

        // Lanjutkan dengan proses pembelian seperti biasa
        // Pengecekan apakah kode_barang adalah undefined
        if (typeof purchasesData.kode_barang === 'undefined') {
            // Lakukan pembaruan nilai kode_barang
            purchasesData.kode_barang = kodeBarang; // Gantikan 'newValue' dengan nilai
yang sesuai
        }

        const purchasesItemsResponse = await fetch(purchasesItemsAPI, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
```

```
        },
        body: JSON.stringify(purchasesData)
    });

    console.log('purchasesData.kode_barang:', purchasesData.kode_barang)
    console.log('kode_barang:', kodeBarang)

    if (!purchasesItemsResponse.ok) {
        Swal.fire({
            icon: 'error',
            title: 'Error',
            text: error.message,
            position: 'center', // Posisikan di tengah halaman
            showConfirmButton: false,
            timer: 3000,
            timerProgressBar: true,
            showClass: {
                popup: 'animate__animated animate__fadeInDown custom-animate'
            },
            hideClass: {
                popup: 'animate__animated animate__fadeOutUp custom-animate'
            }
        });
        throw new Error('Gagal menyimpan data pembelian.');
    }

    // Hapus baris dari tabel setelah data dikirim ke backend
    row.parentNode.removeChild(row);

    // Jika ini adalah baris terakhir, tampilkan pesan sukses
```

```
            if (index === tableRows.length - 1) {
                nomorNota = ''; // reset nomorNota setelah penggunaan
                totalPurchases = 0;

                setDefaultDate(pickedDate);
                document.querySelector("#purchases-note-total").innerHTML = `Rp.
${formatNumber(totalPurchases)}`;
                document.querySelector("#purchases-supplier-note-number").innerHTML = ''

                document.querySelector("#no-nota-supplier").value = '';
                addedNotaSupplier = null;

                // Mengosongkan input field supplier
                const supplierSelect = document.getElementById("search-supplier");
                supplierSelect.value = '';
                addedSupplier = null;

                document.querySelector("#pesan-rekomendasi").style.display = "none";
                document.querySelector("#pesan-rekomendasi").textContent = "";

                Swal.fire({
                    icon: 'success',
                    title: 'Sukses!',
                    text: 'Data pembelian telah disimpan',
                    position: 'center', // Posisikan di tengah halaman
                    showConfirmButton: false,
                    timer: 1500,
                    timerProgressBar: true,
                    showClass: {
                        popup: 'animate__animated animate__fadeInDown custom-animate'
```

```
                },
                hideClass: {
                    popup: 'animate__animated animate__fadeOutUp custom-animate'
                }
            });
        }
    } catch (error) {
        console.error('Error:', error);
        Swal.fire({
            icon: 'error',
            title: 'Error',
            text: 'Terjadi kesalahan saat menyimpan data pembelian',
            position: 'center', // Posisikan di tengah halaman
            showConfirmButton: false,
            timer: 1500,
            timerProgressBar: true,
            showClass: {
                popup: 'animate__animated animate__fadeInDown custom-animate'
            },
            hideClass: {
                popup: 'animate__animated animate__fadeOutUp custom-animate'
            }
        });
    }
});

pickedDate.placeholder = 'dd/mm/yyyy';
} catch (error) {
    console.error('Error:', error);
    alert('Terjadi kesalahan saat menyimpan data pembelian.');
```

```
    }
  }
});
```

4.1.4 Proses Perhitungan Safety Stock

Segmen Program 4.5 Perhitungan Safety Stock

```
async function safetyStock(barangId) {
  try{
  const salesResponse = await fetch("http://localhost:3000/salesItems/byBarangId/" + barangId);
  const salesData = await salesResponse.json();
  const purchasesResponse = await fetch("http://localhost:3000/purchasesItems/getLeadTime/" +
barangId);
  const purchasesData = await purchasesResponse.json();

  // Hitung penjualan harian maksimum dan rata-rata
  const maxPenjualan = Math.max(...salesData.map(sale => sale.kuantitas));
  const averagePenjualan = salesData.reduce((acc, sale) => acc + sale.kuantitas, 0) / salesData.length;
  console.log(`maxPenjualan & averagePenjualan for ${barangId}: ${maxPenjualan}, ${averagePenjualan}
`);

  // Hitung lead time maksimum dan rata-rata
  const purchasesLeadTimes = purchasesData.map(purchase =>
purchase.purchasesLeadTime).filter(leadTime => leadTime !== undefined);

  const maxLeadTime = purchasesLeadTimes.length > 0 ? Math.max(...purchasesLeadTimes) : 0;

  const averageLeadTime = purchasesLeadTimes.length > 0 ? purchasesLeadTimes.reduce((acc, leadTime)
=> acc + leadTime, 0) / purchasesLeadTimes.length : 0;

  console.log(`maxLeadTime & averageLeadTime for ${barangId}: ${maxLeadTime}, ${averageLeadTime} `);
```

```
    // Hitung safety stock berdasarkan rumus yang diberikan
    let calculatedSafetyStock = Math.round((maxPenjualan * maxLeadTime) - (averagePenjualan *
averageLeadTime));


    if (isNaN(calculatedSafetyStock) || calculatedSafetyStock < 0) {
        calculatedSafetyStock = 0;
    }
    const safetyStock = Math.round(calculatedSafetyStock);
    console.log(`safetyStock for ${barangId}: ${safetyStock} `);
    return safetyStock;
    } catch(error) {
        console.error("Error calculating safety stock:", error);
        return 0;
    }

}
```

4.1.5 Proses Perhitungan ROP

Segmen Program 4.6 Perhitungan ROP

```
const productsDataResponse = await fetch("http://localhost:3000/products");
    const productData = await productsDataResponse.json();

const tableData = productData.map(async (product) => {
        const { barangId, kode_barang, nama_barang, kategori_barang, harga_jual, stok, safety_stock } =
product;
        const demandResponse = await fetch("http://localhost:3000/salesItems/byBarangId/" + barangId);
        const demandData = await demandResponse.json();
        const salesQuantity = demandData.reduce((total, item) => total + item.kuantitas, 0);

        const leadTimeResponse = await fetch("http://localhost:3000/purchasesItems/getLeadTime/" +
barangId);
```

```
    const leadTimeData = await leadTimeResponse.json();
    const leadTimesForProduct = leadTimeData.filter(item => item.barangId === barangId);
    let leadTimeSum = 0;
    for (const item of leadTimesForProduct) {
      leadTimeSum += item.purchasesLeadTime;
    }
    const averageLeadTime = leadTimeSum / leadTimesForProduct.length;
    const ROP = Math.round((salesQuantity * averageLeadTime) + safety_stock);
    product.ROP = ROP;

    const kategoriText = kategoriMapping[kategori_barang] || '-';
    product.kategoriText = kategoriText;
    return product;
  });
```

4.1.6 Proses Perhitungan EOQ

Segmen Program 4.7 Perhitungan EOQ

```
 const productsDataResponse = await fetch("http://localhost:3000/products");
   const productData = await productsDataResponse.json();



const tableData = productData.map(async (product) => {
    const { barangId, kode_barang, nama_barang, kategori_barang, harga_jual, stok, safety_stock } =
product;
    const demandResponse = await fetch("http://localhost:3000/salesItems/byBarangId/" + barangId);
    const demandData = await demandResponse.json();
    const salesQuantity = demandData.reduce((total, item) => total + item.kuantitas, 0);

    const leadTimeResponse = await fetch("http://localhost:3000/purchasesItems/getLeadTime/" +
```

```
barangId);
      const leadTimeData = await leadTimeResponse.json();
      const leadTimesForProduct = leadTimeData.filter(item => item.barangId === barangId);
      let leadTimeSum = 0;
      for (const item of leadTimesForProduct) {
        leadTimeSum += item.purchasesLeadTime;
      }
      const averageLeadTime = leadTimeSum / leadTimesForProduct.length;
      const EOQ = Math.round(Math.sqrt((2 * salesQuantity * safety_stock) / (harga_jual * 0.1))); //
Assuming holding cost is 10% of item cost
      product.EOQ = EOQ;

      const kategoriText = kategoriMapping[kategori_barang] || '-';
      product.kategoriText = kategoriText;
      return product;
    });
```