# LAPORAN
# TUGAS BESAR PEMROGRAMAN BERBASIS OBJEK
# APLIKASI PEMANTAUAN BAHAN BAKU MAKAN BERGIZI BAIK

Dibuat untuk memenuhi Tugas Besar mata kuliah Pemrograman Berbasis Objek


Oleh:

| | |
|---|---|
| Fahraj Ananta Aulia Arkan | 241511042 |
| Raihana Aisha Az-Zahra | 241511056 |
| Seruni Libertina Islami | 241511064 |

**PROGRAM STUDI D3 TEKNIK INFORMATIKA**

**JURUSAN TEKNIK INFORMATIKA**

**POLITEKNIK NEGERI BANDUNG**

**KOTA BANDUNG**


**2025**

# DAFTAR ISI

# BAB I
# PENDAHULUAN

## A. Latar Belakang

Program Makan Siang Gratis (MBG) merupakan salah satu upaya pemerintah untuk menyediakan makanan bergizi bagi anak-anak mulai dari rentang sekolah Tingkat dasar hingga sekolah menengah atas (SMA). Tujuan utamanya adalah memastikan anak-anak mendapatkan asupan gizi yang cukup untuk mendukung kesehatan dan produktivitas. Namun, dalam praktiknya, pengelolaan bahan baku menjadi tantangan penting. Beberapa kasus di lapangan menunjukkan adanya risiko keracunan akibat penggunaan bahan yang sudah mendekati atau melewati tanggal kedaluwarsa. Kejadian ini biasanya terjadi karena kurangnya pemantauan stok secara rutin, pencatatan bahan yang kurang rapi, serta proses distribusi bahan dari gudang ke dapur yang tidak terkontrol dengan baik.

Situasi ini menimbulkan dampak serius, tidak hanya bagi kesehatan penerima makanan, tetapi juga bagi reputasi program MBG dan kepercayaan masyarakat terhadap penyelenggara. Untuk mencegah hal tersebut, dibutuhkan sistem yang mampu memantau stok dan status kedaluwarsa secara otomatis, mempermudah pengajuan dan persetujuan permintaan bahan, serta mencatat setiap aktivitas pengelolaan bahan secara terstruktur. Dengan adanya sistem berbasis desktop ini, petugas gudang dan dapur dapat bekerja lebih efisien, memastikan bahan yang digunakan masih layak konsumsi, dan mengurangi risiko terjadinya keracunan makanan, sehingga program MBG dapat berjalan dengan aman, efektif, dan terpercaya.

## B. Tujuan

Tujuan dari pembuatan aplikasi pemantauan bahan baku ini adalah:

1. Memastikan semua bahan baku yang digunakan dalam program MBG masih layak konsumsi.
2. Memudahkan petugas gudang dalam memantau stok dan status kedaluwarsa bahan baku.
3. Membantu petugas dapur dalam mengajukan permintaan bahan secara terkontrol dan aman.

4. Meningkatkan efisiensi koordinasi antara gudang dan dapur.

## C. Manfaat

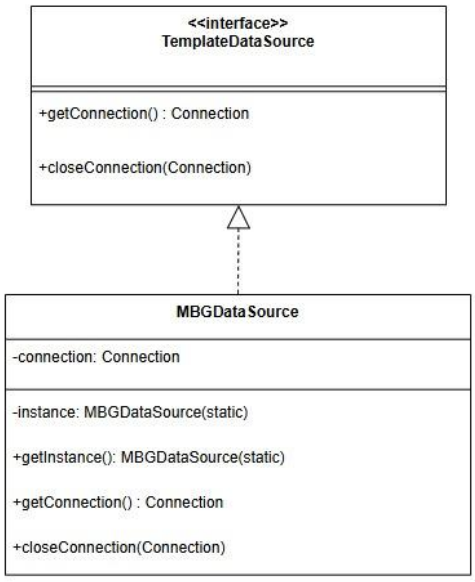
Manfaat dari penerapan sistem ini antara lain:

1. **Keamanan Konsumsi:** Mengurangi risiko keracunan akibat penggunaan bahan kadaluarsa.
2. **Pengelolaan Stok Lebih Baik:** Stok bahan baku dapat dipantau secara otomatis dengan status yang jelas.
3. **Efisiensi Operasional:** Permintaan bahan dari dapur dapat diproses lebih cepat dan terstruktur.
4. **Transparansi dan Akuntabilitas:** Semua aktivitas pengelolaan bahan tercatat sehingga memudahkan pelaporan dan evaluasi.

# BAB II
# ANALISIS DESIGN PATTERN

## A. Singleton Design Pattern (Creational)

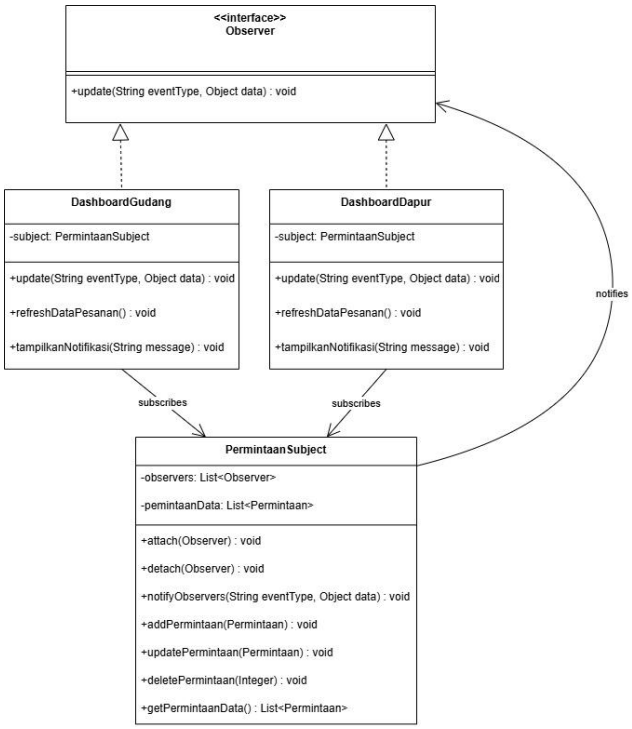
Pada program ini, Design Pattern Singleton diterapkan pada kelas *DatabaseConnection* untuk memastikan hanya ada satu instance koneksi database selama aplikasi berjalan. Penggunaan Singleton sangat penting karena setiap modul dalam program, baik dari sisi gudang maupun dapur, membutuhkan akses yang sama ke database. Dengan adanya satu koneksi tunggal, sumber daya sistem digunakan secara efisien dan konsistensi data tetap terjaga, sehingga mengurangi risiko konflik atau duplikasi data. Selain itu, Singleton memudahkan manajemen transaksi dan mempermudah pemeliharaan koneksi karena semua modul menggunakan instance yang sama tanpa perlu membuat koneksi baru.




## B. Observer Design Pattern (Behavioral)

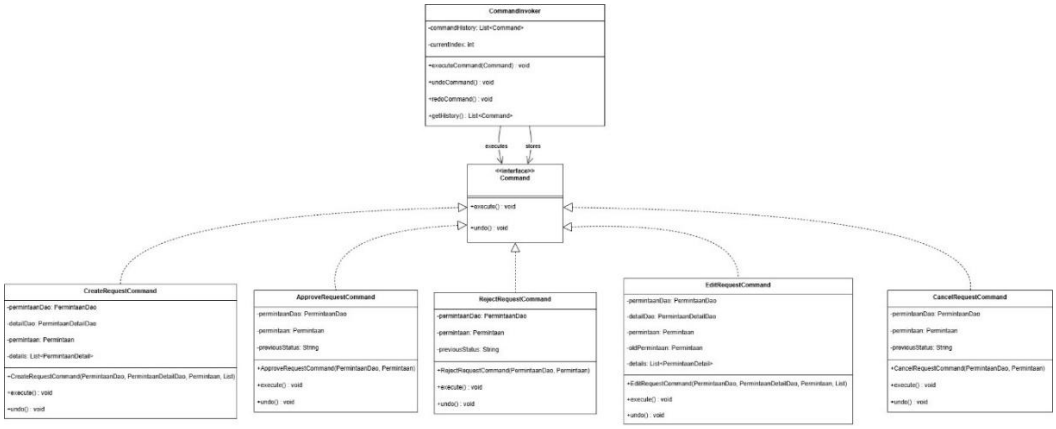
Design Pattern Observer digunakan agar Dashboard Gudang dapat secara otomatis ter-update ketika ada perubahan data stok atau permintaan bahan baku. Dashboard bertindak sebagai observer yang menerima notifikasi dari subject (data stok/permintaan) setiap kali terjadi perubahan. Dengan begitu, pengguna tidak

perlu melakukan refresh manual untuk melihat data terbaru. Pola ini meningkatkan interaktivitas GUI, memastikan informasi yang ditampilkan selalu akurat, dan membuat sistem lebih responsif terhadap perubahan data secara real-time, sehingga pengambilan keputusan menjadi lebih cepat dan tepat.



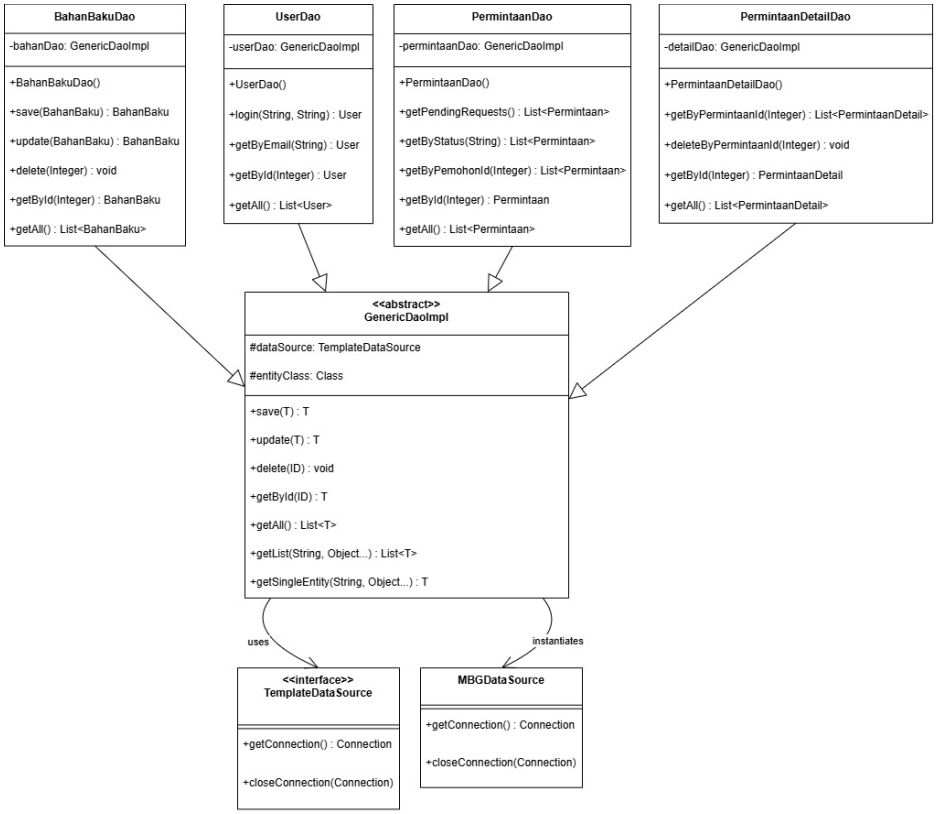

## C. Command Design Pattern (Behavioral)

Design Pattern Command diterapkan untuk membungkus setiap aksi pengguna, seperti approve atau reject permintaan bahan baku dan membuat permintaan bahan baru, menjadi objek terpisah. Pendekatan ini memisahkan eksekusi aksi dari pemanggilnya, sehingga alur program menjadi lebih fleksibel dan modular. Command memudahkan implementasi fitur tambahan seperti logging aksi, undo/redo, atau antrian eksekusi, sehingga pengujian dan pengembangan fitur baru bisa dilakukan tanpa mengubah kode inti. Dengan demikian, Command meningkatkan fleksibilitas aplikasi, mendukung perluasan fungsionalitas, dan mempermudah pengelolaan aksi pengguna secara terstruktur.




## D. Template Data Access Object (Structural)

Pola Data Acess Object (DAO) diterapkan pada program untuk memisahkan logika akses data dari implementasi konkret database. Antarmuka BahanBakuDAO berperan sebagai template yang mendefinisikan metode-metode standar untuk operasi CRUD bahan baku, sementara MySQLBahanBakuDAO menyediakan implementasi konkret menggunakan query SQL. Pemisahan ini membuat kode bisnis di modul lain tidak tergantung pada jenis database tertentu, sehingga memudahkan penggantian database di masa depan tanpa mengubah logika utama program. Selain itu, pola DAO meningkatkan modularitas dan maintainability, karena setiap perubahan pada akses data hanya perlu dilakukan pada implementasi DAO, serta memudahkan pengujian unit dengan menggantikan database asli menggunakan mock atau stub. Walaupun DAO tidak termasuk dalam 23 pola desain GoF, kami menggunakan pola ini untuk menjaga modularitas dan memisahkan logika bisnis dari akses data. Pemisahan ini membuat kode bisnis tidak tergantung pada jenis database tertentu, memudahkan penggantian database, meningkatkan maintainability, dan mendukung pengujian unit dengan mock atau stub. Dengan demikian, DAO membantu menjaga struktur aplikasi tetap teratur dan pengelolaan akses data lebih terkontrol.


BahanBakuDao
-bahanDao: GenericDaoImpl
+BahanBakuDao()
+save(BahanBaku) : BahanBaku
+update(BahanBaku) : BahanBaku
+delete(Integer) : void
+getById(Integer) : BahanBaku
+getAll() : List<BahanBaku>
UserDao
-userDao: GenericDaoImpl
+UserDao()
+login(String, String) : User
+getByEmail(String) : User
+getById(Integer) : User
+getAll() : List<User>
PermintaanDao
-permintaanDao: GenericDaoImpl
+PermintaanDao()
+getPendingRequests() : List<Permintaan>
+getByStatus(String) : List<Permintaan>
+getByPemohonId(Integer) : List<Permintaan>
+getById(Integer) : Permintaan
+getAll() : List<Permintaan>
PermintaanDetailDao
-detailDao: GenericDaoImpl
+PermintaanDetailDao()
+getByPermintaanId(Integer) : List<PermintaanDetail>
+deleteByPermintaanId(Integer) : void
+getById(Integer) : PermintaanDetail
+getAll() : List<PermintaanDetail>
<<abstract>>
GenericDaoImpl
#dataSource: TemplateDataSource
#entityClass: Class
+save(T) : T
+update(T) : T
+delete(ID) : void
+getById(ID) : T
+getAll() : List<T>
+getList(String, Object...) : List<T>
+getSingleEntity(String, Object...) : T
uses
instantiates
<<interface>>
TemplateDataSource
+getConnection() : Connection
+closeConnection(Connection)
MBGDataSource
+getConnection() : Connection
+closeConnection(Connection)

# BAB III
# IMPLEMENTASI

## A. Struktur Program

Program “Aplikasi Makanan Bergizi Baik” dibangun dengan bahasa Java dan menerapkan arsitektur modular agar setiap komponen memiliki tanggung jawab yang jelas. Struktur folder dibagi menjadi beberapa paket utama yaitu:

### 1. Config

Berisi konfigurasi aplikasi, terutama pengaturan koneksi database. MBGDataSource.java menggunakan Singleton Pattern untuk memastikan hanya ada satu instance koneksi database yang digunakan sepanjang aplikasi berjalan. Ini meningkatkan efisiensi dan mencegah multiple connection error.

```java
public class MBGDataSource implements TemplateDataSource {

    private static MBGDataSource instance;
    private Connection connection;
    private boolean isConnected = false;


    // Private constructor untuk mencegah instansiasi dari luar
    private MBGDataSource() {
    }

    // Method untuk mendapatkan instance Singleton
    public static synchronized MBGDataSource getInstance() {
        if (instance == null) {
            instance = new MBGDataSource();
        }
        return instance;
    }

    @Override
    public Connection getConnection() throws Exception {
        try {
            // Jika koneksi sudah ada dan masih aktif, return koneksi yang sama
            if (connection != null && !connection.isClosed() && isConnected) {
                System.out.println("Menggunakan koneksi yang sudah ada");
                return connection;
            }

            // Jika koneksi belum ada atau sudah ditutup, buat koneksi baru
            System.out.println("Membuat koneksi database baru...");
            Class.forName(TemplateConfiguration.getString("driverClassName"));

            connection = DriverManager.getConnection(
                TemplateConfiguration.getString("db.connection.url"),
                TemplateConfiguration.getString("db.connection.username"),
                TemplateConfiguration.getString("db.connection.checksum")
            );

            isConnected = true;
            System.out.println("Koneksi database berhasil dibuat");
            return connection;

        } catch (ClassNotFoundException e) {
            isConnected = false;
            throw new Exception("Database driver tidak ditemukan: " + e.getMessage(), e);
        } catch (SQLException e) {
            isConnected = false;
            throw new Exception("Gagal membuat koneksi database: " + e.getMessage(), e);
        }
    }

    // Menutup koneksi database
    @Override
    public void closeConnection(Connection con) {
        if (con != null) {
            try {
                con.close();
                isConnected = false;
                System.out.println("Koneksi database ditutup");
            } catch (SQLException e) {
                System.err.println("Error saat menutup koneksi: " + e.getMessage());
                e.printStackTrace();
            }
        }
    }

    // Method untuk reset instance
    public static synchronized void resetInstance() {
        if (instance != null && instance.connection != null) {
            try {
                instance.connection.close();
                instance.isConnected = false;
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        instance = null;
    }

    // Method untuk check status koneksi
    public boolean isConnected() {
        return isConnected;
    }
}
```

### a) Data Access Object (DAO)

Folder ini mengimplementasikan Data Access Object (DAO) untuk mengelola operasi database. UserDao, BahanBakuDao, PermintaanDao, dan PermintaanDetailDao mewarisi fungsionalitas CRUD dari GenericDaoImpl. DAO memisahkan logika akses data dari kode bisnis, membuat aplikasi lebih modular dan mudah diuji.

**BahanBakuDao.java**

```java
package com.mbg.dao;

import com.codeway.daoTemplate.dao.GenericDaoImpl;
import com.mbg.config.MBGDataSource;
import com.mbg.model.BahanBaku;

public class BahanBakuDao extends GenericDaoImpl<Integer, BahanBaku> {
    public BahanBakuDao() {
        super(BahanBaku.class, MBGDataSource.getInstance());
    }

    public BahanBaku getById(Integer id) throws Exception {
        return getSingleEntity("id = ?", id);
    }
}
```

**PermintaanDao.java**

```java
package com.mbg.dao;

import java.util.List;

import com.codeway.daoTemplate.dao.GenericDaoImpl;
import com.mbg.config.MBGDataSource;
import com.mbg.model.Permintaan;

public class PermintaanDao extends GenericDaoImpl<Integer, Permintaan> {
    public PermintaanDao() {
        super(Permintaan.class, MBGDataSource.getInstance());
    }

    public List<Permintaan> getPendingRequests() throws Exception {
        // Mengambil semua permintaan yang statusnya 'menunggu'
        return getList("status = ?", "menunggu");
    }

    public List<Permintaan> getByStatus(String status) throws Exception {
        return getList("status = ?", status);
    }

    public List<Permintaan> getByPemohonId(Integer pemohonId) throws Exception {
        return getList("pemohon_id = ?", pemohonId);
    }

    public Permintaan getById(Integer id) throws Exception {
        return getSingleEntity("id = ?", id);
    }

}
```

**PermintaanDetailDao.java**

```java
package com.mbg.dao;

import java.util.List;

import com.codeway.daoTemplate.dao.GenericDaoImpl;
import com.mbg.config.MBGDataSource;
import com.mbg.model.PermintaanDetail;

public class PermintaanDetailDao extends GenericDaoImpl<Integer, PermintaanDetail> {
    public PermintaanDetailDao() {
        super(PermintaanDetail.class, MBGDataSource.getInstance());
    }

    public List<PermintaanDetail> getByPermintaanId(Integer permintaanId) throws Exception {
        return getList("permintaan_id = ?", permintaanId);
    }

    public void deleteByPermintaanId(Integer permintaanId) throws Exception {
        try {
            List<PermintaanDetail> details = getByPermintaanId(permintaanId);
            for (PermintaanDetail detail : details) {
                remove(detail.getId());
            }
        } catch (Exception e) {
            System.err.println("Error deleting detail permintaan: " + e.getMessage());
            throw e;
        }
    }
}
```

**UserDao.java**

```java
package com.mbg.dao;

import com.codeway.daoTemplate.dao.GenericDaoImpl;
import com.mbg.config.MBGDataSource;
import com.mbg.model.User;

public class UserDao extends GenericDaoImpl<Integer, User> {
    public UserDao() {
        super(User.class, MBGDataSource.getInstance());
    }

    public User login(String email, String password) throws Exception {
        // Mengambil data user berdasarkan email
        return getSingleEntity("email = ?", email);
    }

    public User getByEmail(String email) throws Exception {
        return getSingleEntity("email = ?", email);
    }
}
```

**b) Graphical User Interface (GUI)**

Berisi seluruh tampilan antarmuka pengguna untuk Petugas Gudang dan Petugas Dapur, seperti form tambah bahan, form permintaan, tabel stok, dan panel persetujuan permintaan.

**c) Helper**

Berisi class utilitas yang membantu proses aplikasi. Contohnya DateLabelFormatter.java yang memformat input tanggal dari GUI agar sesuai format yyyy-MM-dd.

```java
package com.mbg.helper;

import javax.swing.JFormattedTextField.AbstractFormatter;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class DateLabelFormatter extends AbstractFormatter {
    private final String datePattern = "yyyy-MM-dd";
    private final SimpleDateFormat dateFormatter = new SimpleDateFormat(datePattern);

    @Override
    public Object stringToValue(String text) throws ParseException {
        return dateFormatter.parse(text);
    }

    @Override
    public String valueToString(Object value) {
        if (value != null) {
            Calendar cal = (Calendar) value;
            return dateFormatter.format(cal.getTime());
        }
        return "";
    }
}

```

**d) Model**

Menyimpan class entity yang merepresentasikan tabel di database. User, BahanBaku, Permintaan, dan PermintaanDetail. Masing-masing class hanya menyimpan atribut dan getter/setter, tanpa logika database, sehingga memisahkan data representation dari akses data.

**BahanBaku.java**

```java
package com.mbg.model;

import javax.persistence.Column;
import javax.persistence.Id;
import javax.persistence.Table;
import java.sql.Date;
import java.sql.Timestamp;

@Table(name = "bahan_baku")
public class BahanBaku {

    @Id
    private Integer id;

    private String nama;
    private String kategori;
    private Integer jumlah;
    private String satuan;

    @Column(name = "tanggal_masuk")
    private Date tanggalMasuk;

    @Column(name = "tanggal_kadaluarsa")
    private Date tanggalKadaluarsa;

    private String status; // tersedia, segera_kadaluarsa, kadaluarsa, habis

    @Column(name = "created_at")
    private Timestamp createdAt;

    // Getters and Setters
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public String getNama() { return nama; }
    public void setNama(String nama) { this.nama = nama; }
    public String getKategori() { return kategori; }
    public void setKategori(String kategori) { this.kategori = kategori; }
    public Integer getJumlah() { return jumlah; }
    public void setJumlah(Integer jumlah) { this.jumlah = jumlah; }
    public String getSatuan() { return satuan; }
    public void setSatuan(String satuan) { this.satuan = satuan; }
    public Date getTanggalMasuk() { return tanggalMasuk; }
    public void setTanggalMasuk(Date tanggalMasuk) { this.tanggalMasuk = tanggalMasuk; }
    public Date getTanggalKadaluarsa() { return tanggalKadaluarsa; }
    public void setTanggalKadaluarsa(Date tanggalKadaluarsa) { this.tanggalKadaluarsa = tanggalKadaluarsa; }
    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }
    public Timestamp getCreatedAt() { return createdAt; }
    public void setCreatedAt(Timestamp createdAt) { this.createdAt = createdAt; }
}
```

**Permintaan.java**

```java
package com.mbg.model;

import javax.persistence.Column;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;
import java.sql.Date;
import java.sql.Timestamp;
import java.util.List;

@Table(name = "permintaan")
public class Permintaan {

    @Id
    private Integer id;

    @Column(name = "pemohon_id")
    private Integer pemohonId;

    @Column(name = "tgl_masak")
    private Date tglMasak;

    @Column(name = "menu_makan")
    private String menuMakan;

    @Column(name = "jumlah_porsi")
    private Integer jumlahPorsi;

    private String status; // menunggu, disetujui, ditolak

    @Column(name = "created_at")
    private Timestamp createdAt;

    // Properti Transient (Tidak disimpan langsung ke tabel permintaan, tapi untuk relasi)
    @Transient
    private User pemohon;

    @Transient
    private List<PermintaanDetail> detail;

    // Getters and Setters
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public Integer getPemohonId() { return pemohonId; }
    public void setPemohonId(Integer pemohonId) { this.pemohonId = pemohonId; }
    public Date getTglMasak() { return tglMasak; }
    public void setTglMasak(Date tglMasak) { this.tglMasak = tglMasak; }
    public String getMenuMakan() { return menuMakan; }
    public void setMenuMakan(String menuMakan) { this.menuMakan = menuMakan; }
    public Integer getJumlahPorsi() { return jumlahPorsi; }
    public void setJumlahPorsi(Integer jumlahPorsi) { this.jumlahPorsi = jumlahPorsi; }
    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }
    public Timestamp getCreatedAt() { return createdAt; }
    public void setCreatedAt(Timestamp createdAt) { this.createdAt = createdAt; }

    public User getPemohon() { return pemohon; }
    public void setPemohon(User pemohon) { this.pemohon = pemohon; }
    public List<PermintaanDetail> getDetail() { return detail; }
    public void setDetail(List<PermintaanDetail> detail) { this.detail = detail; }
}
```

**PermintaanDetail.java**

```java
package com.mbg.model;

import javax.persistence.Column;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;

@Table(name = "permintaan_detail")
public class PermintaanDetail {

    @Id
    private Integer id;

    @Column(name = "permintaan_id")
    private Integer permintaanId;

    @Column(name = "bahan_id")
    private Integer bahanId;

    @Column(name = "jumlah_diminta")
    private Integer jumlahDiminta;

    @Transient
    private BahanBaku bahanBaku;

    // Getters and Setters
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public Integer getPermintaanId() { return permintaanId; }
    public void setPermintaanId(Integer permintaanId) { this.permintaanId = permintaanId; }
    public Integer getBahanId() { return bahanId; }
    public void setBahanId(Integer bahanId) { this.bahanId = bahanId; }
    public Integer getJumlahDiminta() { return jumlahDiminta; }
    public void setJumlahDiminta(Integer jumlahDiminta) { this.jumlahDiminta = jumlahDiminta; }

    public BahanBaku getBahanBaku() { return bahanBaku; }
    public void setBahanBaku(BahanBaku bahanBaku) { this.bahanBaku = bahanBaku; }
}
```

**User.java**

```java
package com.mbg.model;

import javax.persistence.Column;
import javax.persistence.Id;
import javax.persistence.Table;
import java.sql.Timestamp;

@Table(name = "user")
public class User {

    @Id
    private Integer id;

    private String name;
    private String email;
    private String password;
    private String role; // 'gudang' atau 'dapur'

    @Column(name = "created_at")
    private Timestamp createdAt;

    // Getters and Setters
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }
    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }
    public Timestamp getCreatedAt() { return createdAt; }
    public void setCreatedAt(Timestamp createdAt) { this.createdAt = createdAt; }
}
```

**e) Pattern**

Mengimplementasikan desain pattern behavioral yaitu Command dan Observer.

**(1) Command**

**(2) Observer**

Menghubungkan subject (PermintaanSubject) dengan observer (Dashboard) sehingga setiap perubahan data permintaan otomatis memicu notifikasi untuk update tampilan.

**Observer.java**

```java
package com.mbg.pattern.observer;

/**
  Observer Interface - Behavioral Pattern
  Mendefinisikan kontrak untuk semua observer yang ingin menerima notifikasi
  dari subject ketika ada perubahan data.
**/
public interface Observer {

    /**
     * Method yang dipanggil ketika subject melakukan notifikasi
     *
     * @param eventType Tipe event yang terjadi (PERMINTAAN_BARU, PERMINTAAN_DIUPDATE, dll)
     * @param data Data yang berkaitan dengan event
     */
    void update(String eventType, Object data);
}
```

**PermintaanSubject.java**

```java
package com.mbg.pattern.observer;

import java.util.ArrayList;
import java.util.List;

import com.mbg.dao.PermintaanDao;
import com.mbg.model.Permintaan;

/**
  PermintaanSubject - Subject dalam Observer Pattern
  Berfungsi untuk:
  - Menyimpan daftar observers (dashboard yang berlangganan)
  - Memberitahu observers ketika ada perubahan data permintaan
  - Mengelola data permintaan dari database
**/
public class PermintaanSubject {

    private List<Observer> observers;
    private PermintaanDao permintaanDao;
    private List<Permintaan> permintaanData;

    // Constructor
    public PermintaanSubject(PermintaanDao permintaanDao) {
        this.observers = new ArrayList<>();
        this.permintaanDao = permintaanDao;
        this.permintaanData = new ArrayList<>();
    }

    // Menambahkan observer ke daftar
    public void attach(Observer observer) {
        if (!observers.contains(observer)) {
            observers.add(observer);
            System.out.println("Observer ditambahkan: " + observer.getClass().getSimpleName());
        }
    }

    // Menghapus observer dari daftar
    public void detach(Observer observer) {
        if (observers.remove(observer)) {
            System.out.println("Observer dihapus: " + observer.getClass().getSimpleName());
        }
    }

    // Memberitahu semua observers tentang perubahan data
    public void notifyObservers(String eventType, Object data) {
        System.out.println("\n[PermintaanSubject] Notifying " + observers.size() + " observers...");
        System.out.println("[PermintaanSubject] Event: " + eventType);

        for (Observer observer : observers) {
            observer.update(eventType, data);
        }
    }

    // Menambahkan permintaan baru dan notify observers
    public void addPermintaan(Permintaan permintaan) throws Exception {
        Permintaan saved = permintaanDao.save(permintaan);
        permintaanData.add(saved);
        notifyObservers("PERMINTAAN_BARU", saved);
    }

    // Mengupdate permintaan dan notify observers
    public void updatePermintaan(Permintaan permintaan) throws Exception {
        permintaanDao.update(permintaan);
        notifyObservers("PERMINTAAN_DIUPDATE", permintaan);
    }

    // Menghapus permintaan dan notify observers
    public void deletePermintaan(Integer permintaanId) throws Exception {
        Permintaan permintaan = permintaanDao.getById(permintaanId);
        if (permintaan != null) {
            permintaanDao.remove(permintaanId);
            permintaanData.remove(permintaan);
            notifyObservers("PERMINTAAN_DIHAPUS", permintaanId);
        }
    }

    // Mengambil data permintaan dari database
    public List<Permintaan> getPermintaanData() throws Exception {
        permintaanData.clear();
        permintaanData.addAll(permintaanDao.getAll());
        return permintaanData;
    }

    // Mendapatkan jumlah observers yang aktif
    public int getObserverCount() {
        return observers.size();
    }
}
```

## B. Penjelasan Modul Utama

Aplikasi ini dibagi menjadi tiga modul interaksi utama yang saling terhubung melalui *logic* database dan *design pattern* yang telah dirancang:

1. **Modul Autentikasi (Login)**

   Program dimulai melalui kelas MainApp. Modul ini berfungsi sebagai gerbang masuk bagi pengguna.

   - Sistem memvalidasi kredensial pengguna (email) menggunakan UserDao.
   - Berdasarkan peran (*role*) yang didapatkan dari database, aplikasi akan mengarahkan pengguna ke antarmuka yang sesuai: gudang untuk peran "gudang" atau dapur untuk peran "dapur".

2. **Modul Dapur (dapur.java)**

   Modul ini dirancang untuk petugas dapur guna mengajukan permintaan bahan baku.

   - **Manajemen Pesanan:** Menggunakan PermintaanDao untuk menampilkan riwayat pesanan.
   - **Pembuatan Permintaan:** Menggunakan Command Pattern (CreateRequestCommand) untuk membungkus proses pembuatan permintaan baru beserta detail bahan bakunya agar transaksi bersifat atomik.
   - **Fleksibilitas Aksi:** Terdapat fitur *Undo* yang dikelola oleh CommandInvoker untuk membatalkan aksi terakhir, seperti pengeditan atau pembatalan pesanan.

3. **Modul Gudang (gudang.java)**

   Modul ini berfungsi sebagai pusat kontrol stok dan persetujuan permintaan.

   - **Manajemen Stok:** Petugas gudang dapat melakukan operasi CRUD (Create, Read, Update, Delete) pada data bahan baku menggunakan BahanBakuDao.
   - **Persetujuan Permintaan:** Mengimplementasikan Observer Pattern. Kelas gudang bertindak sebagai *Observer* yang memantau PermintaanSubject. Jika ada perubahan status permintaan atau

permintaan baru masuk, antarmuka gudang akan otomatis diperbarui atau memberikan notifikasi tanpa perlu *refresh* manual.

- **Validasi Stok:** Saat menyetujui permintaan, sistem secara otomatis mengecek ketersediaan stok di BahanBakuDao. Jika stok mencukupi, status permintaan diubah menjadi "disetujui" dan jumlah stok berkurang secara otomatis.

## C. Tampilan GUI

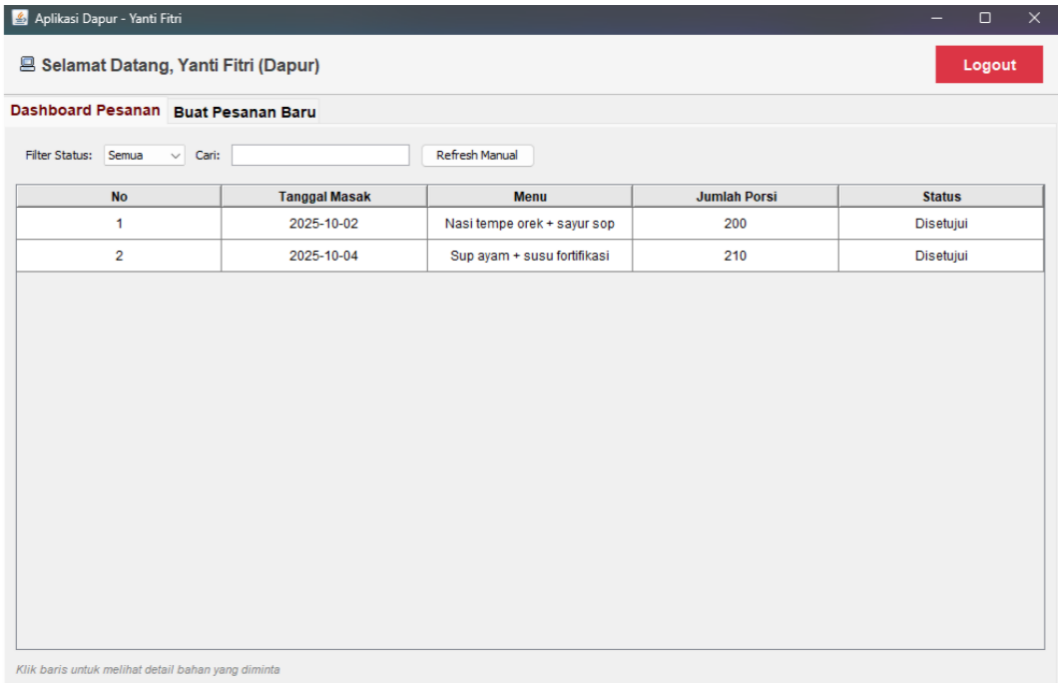
Antarmuka pengguna dibangun menggunakan Java Swing dengan pendekatan *tabbed pane* untuk memisahkan fungsionalitas. Berikut adalah deskripsi alur antarmuka:

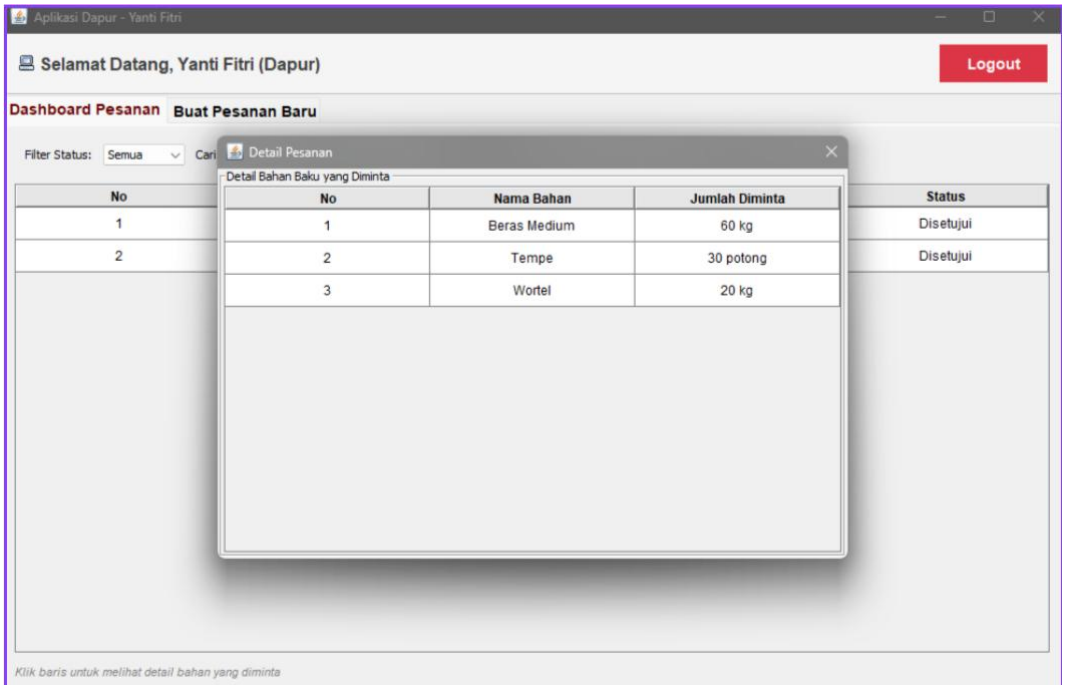
### 1. Antarmuka Login

Tampilan awal berupa *form* sederhana yang meminta input Email dan Password. Terdapat validasi jika input kosong atau email tidak ditemukan di database.

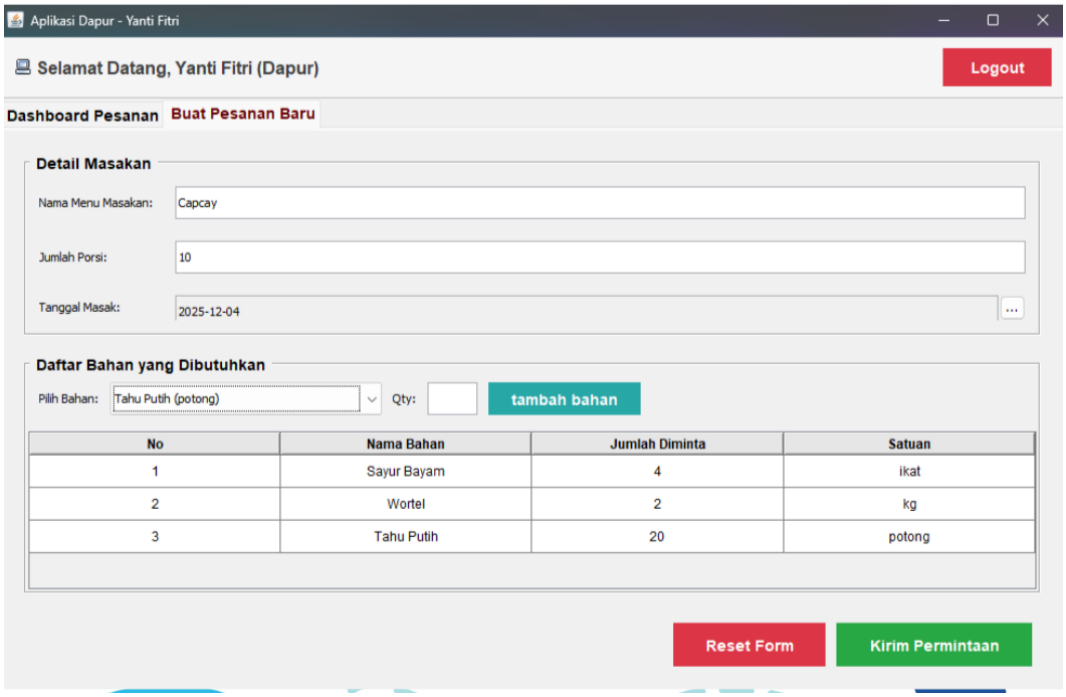
2. **Antarmuka Dapur Terdiri dari dua *tab* utama:**

   **a)** Dashboard Pesanan: Menampilkan tabel daftar permintaan yang pernah diajukan beserta statusnya (Menunggu, Disetujui, Ditolak). Terdapat fitur pencarian dan filter status. Jika baris tabel diklik, detail bahan baku yang diminta akan muncul dalam jendela *popup*.



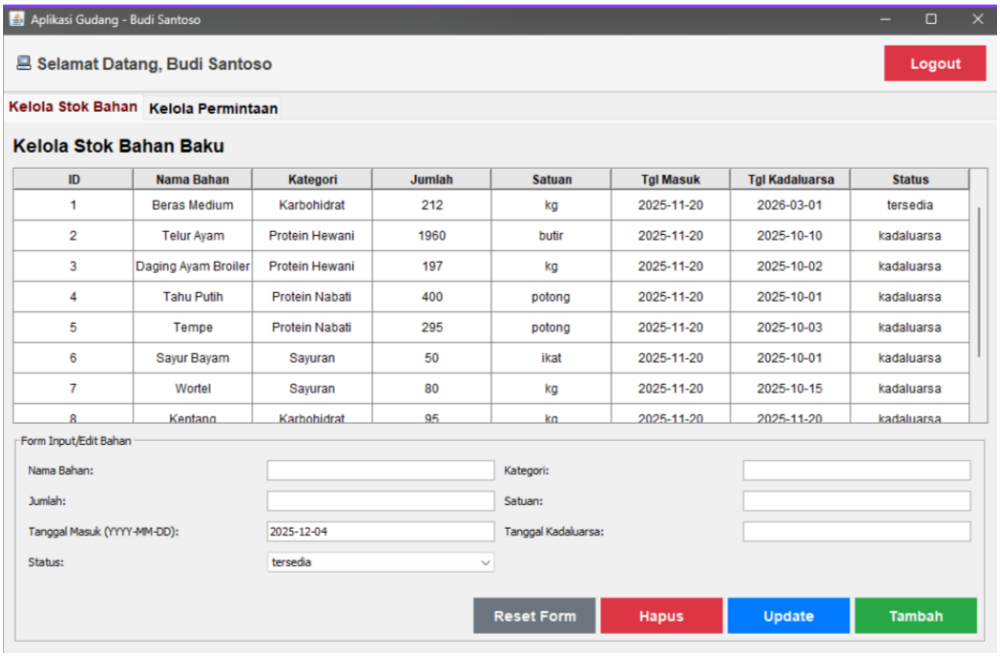



**b)** Buat Pesanan Baru: Berisi *form* input untuk Nama Menu, Jumlah Porsi, dan Tanggal Masak. Di bawahnya terdapat tabel keranjang belanja di mana pengguna dapat memilih bahan baku dari *dropdown* dan menentukan jumlahnya sebelum mengirimkan permintaan ke gudang.

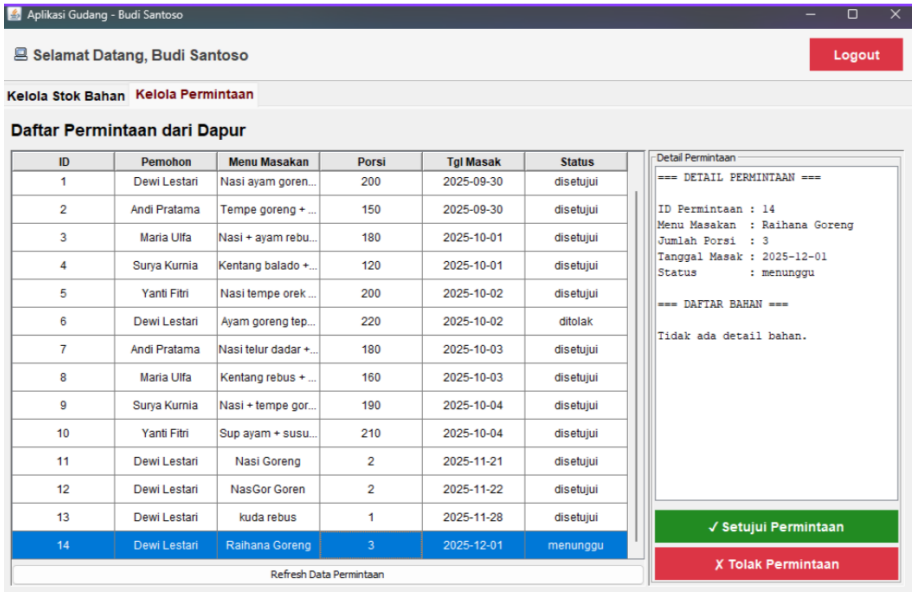


3. **Antarmuka Gudang Terdiri dari dua *tab* utama:**

   **a)** Kelola Stok Bahan: Menampilkan tabel stok bahan baku yang mencakup informasi nama, kategori, jumlah, satuan, tanggal masuk, tanggal kadaluarsa, dan status. Di bagian bawah terdapat *form* input untuk menambah atau memperbarui data bahan baku.



**b)** Kelola Permintaan: Menggunakan konsep *Split Pane*. Bagian kiri menampilkan daftar permintaan masuk dari dapur. Jika salah satu permintaan dipilih, bagian kanan akan menampilkan detail lengkap bahan yang diminta beserta tombol aksi untuk "Setujui" atau "Tolak".




## D. Penerapan Konsep Pemrograman (JUnit, JCF, Generic, Clean Code)

Aplikasi ini menerapkan berbagai konsep pemrograman tingkat lanjut untuk memastikan kode yang bersih, teruji, dan efisien:

### 1. Java Collection Framework (JCF)

Penggunaan JCF sangat ekstensif dalam pengelolaan data di memori:

- List dan ArrayList: Digunakan dalam CommandInvoker untuk menyimpan riwayat perintah (commandHistory) dan dalam DAO untuk menampung hasil *query* database (ResultSet).
- Stack: Digunakan dalam CommandInvoker untuk mengimplementasikan fitur undoStack dan redoStack, yang memungkinkan operasi LIFO (*Last In First Out*) untuk membatalkan aksi.
- Map dan HashMap: Digunakan pada kelas GenericDaoImpl untuk memetakan nama kolom database ke *field* pada *class* Java menggunakan refleksi.

**2. Generic Programming**

Penerapan *Generics* memungkinkan penggunaan kembali kode (*code reuse*) secara maksimal pada lapisan akses data.

- Kelas GenericDaoImpl<Pk, Entity> mendefinisikan operasi dasar database secara umum. Parameter Pk merepresentasikan tipe data *Primary Key* (seperti Integer), dan Entity merepresentasikan kelas model (seperti User, BahanBaku).
- Hal ini membuat kelas turunan seperti BahanBakuDao atau UserDao tidak perlu menulis ulang kode SQL dasar untuk operasi CRUD, cukup mewarisi dari kelas generik tersebut.

**3. JUnit Testing**

Pengujian unit dilakukan untuk memastikan logika bisnis dan akses data berjalan benar sebelum diintegrasikan ke GUI.

- Contoh pada UserDaoTest.java menguji skenario penyimpanan *user* baru, pencarian berdasarkan email, dan verifikasi *password* yang di-*hash* menggunakan *library* BCrypt.

```java
public class UserDaoTest {

    private UserDao userDao;
    private User testUser;

    @BeforeEach
    void setUp() {
        userDao = new UserDao();
        // User dummy untuk testing
        testUser = new User();
        testUser.setName("User Test JUnit");
        testUser.setEmail("junit.test@mbg.id");
        // Hash password
        String hashedPass = BCrypt.hashpw("password123", BCrypt.gensalt());
        testUser.setPassword(hashedPass);
        testUser.setRole("gudang");
        testUser.setCreatedAt(new Timestamp(System.currentTimeMillis()));
    }

    @Test
    void testSaveAndFindUser() {
        try {
            System.out.println("Menyimpan user test...");
            User savedUser = userDao.save(testUser);
            Assertions.assertNotNull(savedUser.getId(), "ID User tidak boleh null setelah save");
            System.out.println("Mencari user berdasarkan email...");
            User foundUser = userDao.getByEmail("junit.test@mbg.id");
            Assertions.assertNotNull(foundUser, "User harus ditemukan di database");
            Assertions.assertEquals("User Test JUnit", foundUser.getName());
            boolean passMatch = BCrypt.checkpw("password123", foundUser.getPassword());
            Assertions.assertTrue(passMatch, "Password harus valid");
            System.out.println("Test CRUD User Berhasil. ID Baru: " + savedUser.getId());
            testUser.setId(savedUser.getId());
        } catch (Exception e) {
            e.printStackTrace();
            Assertions.fail("Gagal melakukan operasi DB: " + e.getMessage());
        }
    }

    @AfterEach
    void tearDown() {
        if (testUser.getId() != null) {
            try {
                userDao.remove(testUser.getId());
                System.out.println("Data test berhasil dihapus (ID: " + testUser.getId() + ")");
            } catch (Exception e) {
                System.err.println("Gagal menghapus data test: " + e.getMessage());
            }
        }
    }
}
```

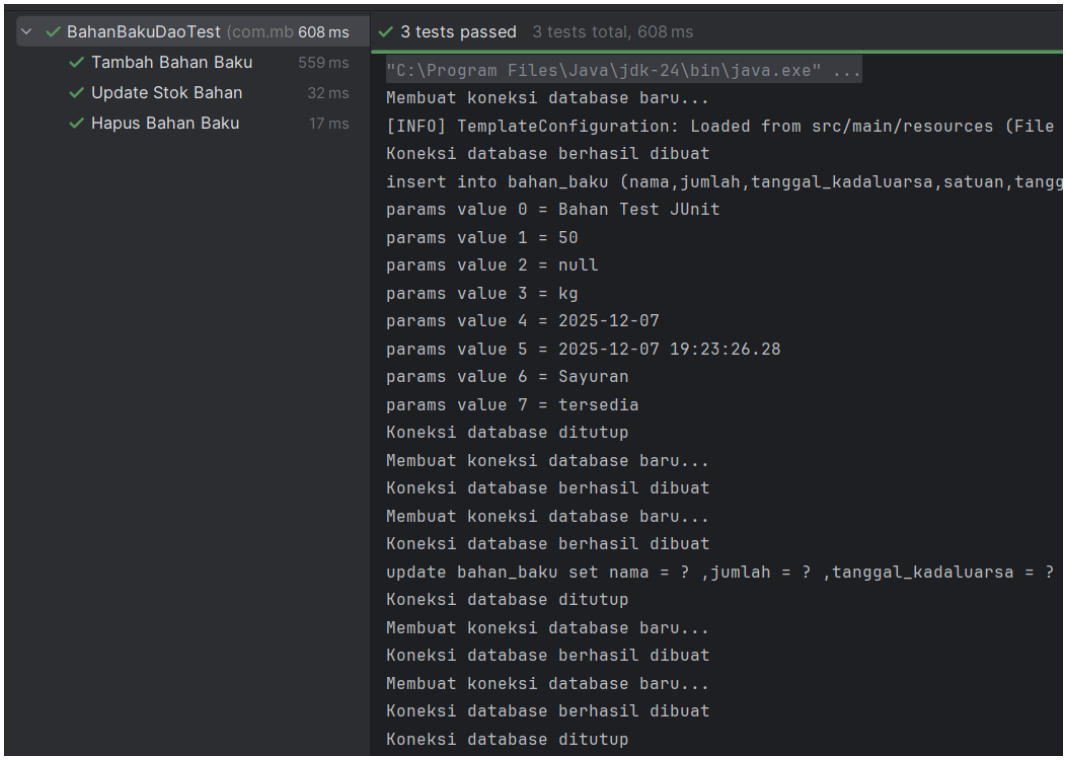
Hasil

```
"C:\Program Files\Java\jdk-24\bin\java.exe" ...
Menyimpan user test...
Membuat koneksi database baru...
[INFO] TemplateConfiguration: Loaded from src/main/resources (File System)
Koneksi database berhasil dibuat
insert into user (password,role,name,created_at,email)  values(?,?,?,?,?)
params value 0 = $2a$10$aEZ0RJO/EwJHyQdwgXO3IewP9KYdHpXBtntKssGkqMU2m/kWEIRJ.
params value 1 = gudang
params value 2 = User Test JUnit
params value 3 = 2025-12-07 19:22:31.417
params value 4 = junit.test@mbg.id
Koneksi database ditutup
Mencari user berdasarkan email...
Membuat koneksi database baru...
Koneksi database berhasil dibuat
query : select * from user where email = ?
values[0] junit.test@mbg.id
Test CRUD User Berhasil. ID Baru: 13
Membuat koneksi database baru...
Koneksi database berhasil dibuat
Koneksi database ditutup
Data test berhasil dihapus (ID: 13)

Process finished with exit code 0
```

- Contoh pada BahanBakuDaoTest.java memvalidasi bahwa bahan baku dapat disimpan, diperbarui stoknya, dan dihapus dengan asersi (Assertions.assertNotNull, Assertions.assertEquals).

```java
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class BahanBakuDaoTest {

    private static BahanBakuDao bahanDao;
    private static BahanBaku bahan;
    private static Integer bahanId;

    @BeforeAll
    static void init() {
        bahanDao = new BahanBakuDao();
        bahan = new BahanBaku();
        bahan.setNama("Bahan Test JUnit");
        bahan.setKategori("Sayuran");
        bahan.setJumlah(50);
        bahan.setSatuan("kg");
        bahan.setTanggalMasuk(Date.valueOf(LocalDate.now()));
        bahan.setStatus("tersedia");
        bahan.setCreatedAt(new Timestamp(System.currentTimeMillis()));
    }

    @Test
    @Order(1)
    @DisplayName("Tambah Bahan Baku")
    void testSaveBahan() throws Exception {
        BahanBaku saved = bahanDao.save(bahan);
        bahanId = saved.getId();
        Assertions.assertNotNull(bahanId, "ID Bahan harus ada");
    }

    @Test
    @Order(2)
    @DisplayName("Update Stok Bahan")
    void testUpdateBahan() throws Exception {
        BahanBaku b = bahanDao.get(bahanId);
        b.setJumlah(25);
        bahanDao.update(b);

        BahanBaku updated = bahanDao.get(bahanId);
        Assertions.assertEquals(25, updated.getJumlah());
    }

    @Test
    @Order(3)
    @DisplayName("Hapus Bahan Baku")
    void testDeleteBahan() throws Exception {
        if (bahanId != null) {
            bahanDao.remove(bahanId);
            Assertions.assertNull(bahanDao.get(bahanId));
        }
    }
}
```

Hasil

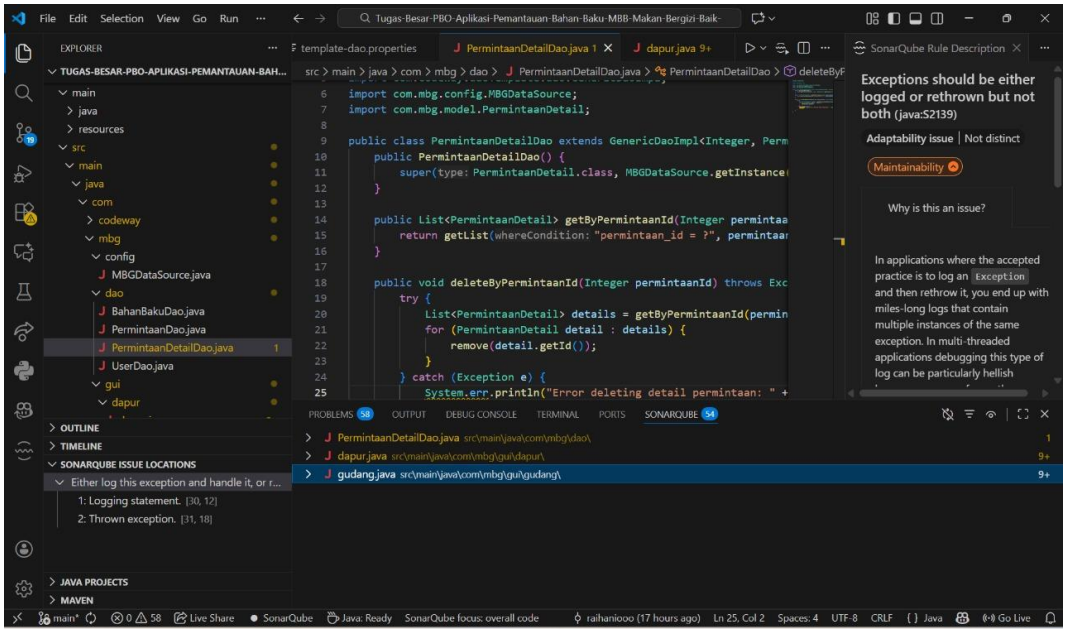



## 4. Analisis Kualitas Kode (SonarQube)

Penerapan *Clean Code* pada aplikasi ini diverifikasi menggunakan *static code analysis tool* SonarQube yang terintegrasi dalam lingkungan pengembangan (VS Code). Berikut adalah perbandingan kondisi kode sebelum dan sesudah dilakukan perbaikan (*refactoring*):

### a) Kondisi Sebelum Perbaikan (Before)

Pada tahap awal pengembangan, SonarQube mendeteksi sejumlah masalah integritas kode dengan total 54 *issues* aktif. Salah satu temuan kritis terdapat pada kelas PermintaanDetailDao.java.

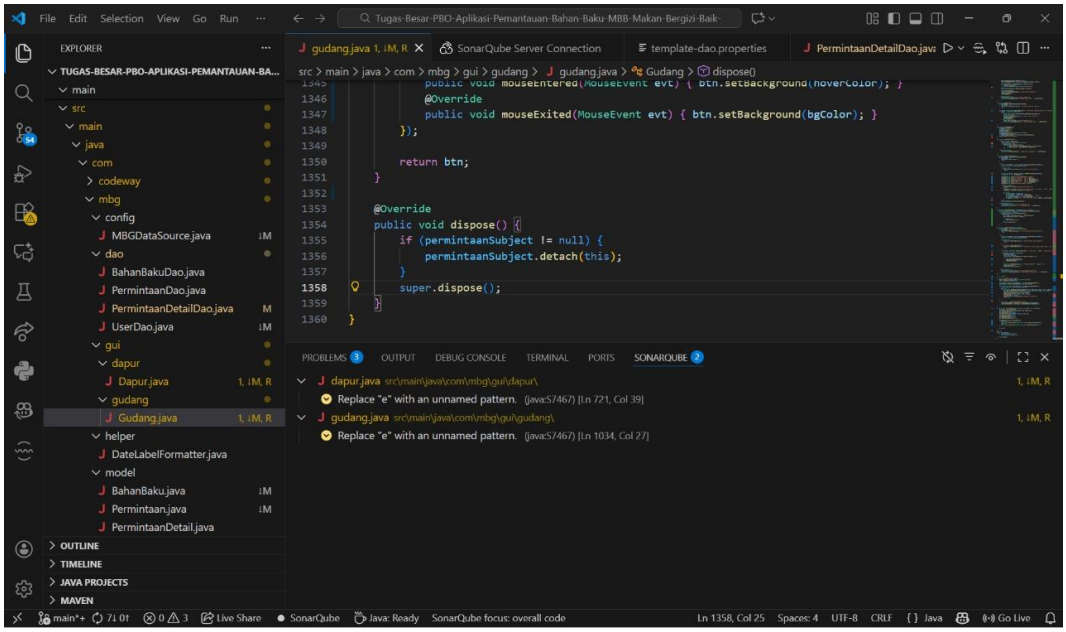



Berdasarkan gambar di atas, SonarQube mendeteksi pelanggaran aturan **java:S2139**: *"Exceptions should be either logged or rethrown but not both"*.

- **Masalah:** Kode melakukan *logging* error menggunakan System.err.println dan kemudian melempar ulang (*re-throw*) *exception* tersebut.
- **Dampak:** Hal ini menyebabkan duplikasi log (*noise*) di server produksi, di mana satu kesalahan dicatat berulang kali di berbagai lapisan aplikasi, menyulitkan proses *debugging*.

**b) Kondisi Setelah Perbaikan (After)**

Setelah dilakukan refaktorisasi berdasarkan rekomendasi SonarQube, kualitas kode meningkat secara signifikan. Jumlah *issues* berhasil diturunkan drastis dari 54 menjadi hanya 2 *issues* minor.




Berdasarkan gambar hasil akhir di atas:

- **Perbaikan Signifikan:** Isu-isu kritikal terkait *Exception Handling* dan *Resource Leak* telah diselesaikan.
- **Sisa Peringatan:** Peringatan yang tersisa pada gudang.java (seperti *Replace 'e' with an unnamed pattern*) hanyalah saran optimasi fitur Java modern untuk keterbacaan, bukan *bug* fungsional. Hal ini menunjukkan bahwa struktur logika utama aplikasi telah memenuhi standar *Clean Code* yang baik.

# BAB IV
# PENUTUP

## A. Kesimpulan

Berdasarkan hasil perancangan dan implementasi yang telah dilakukan, dapat disimpulkan bahwa:

1. Aplikasi Pemantauan Bahan Baku Makan Bergizi Baik (MBG) berhasil dibangun menggunakan bahasa pemrograman Java dengan arsitektur yang modular.
2. Penerapan *Design Patterns* terbukti meningkatkan kualitas struktur kode program:
    - **Singleton Pattern** pada MBGDataSource berhasil menjaga efisiensi koneksi database dengan memastikan hanya satu instansi koneksi yang aktif.
    - **DAO Pattern** (dengan GenericDaoImpl) berhasil memisahkan logika bisnis dari detail teknis akses database, memudahkan pengembangan modul baru.
    - **Command Pattern** memberikan fleksibilitas dalam pengelolaan aksi pengguna di modul Dapur, memungkinkan fitur *undo* yang meningkatkan *user experience*.
    - **Observer Pattern** memungkinkan modul Gudang untuk selalu menampilkan data permintaan terkini secara *real-time* tanpa intervensi manual.
3. Aplikasi ini mampu menjawab permasalahan pengelolaan bahan baku dengan menyediakan fitur pemantauan stok, pencatatan tanggal kadaluarsa, dan sistem permintaan bahan yang terstruktur antara dapur dan gudang.

## B. Saran

Untuk pengembangan selanjutnya, terdapat beberapa saran perbaikan sistem:

1. **Keamanan:** Menambahkan enkripsi pada konfigurasi database dan menerapkan level otorisasi yang lebih kompleks.

2. **Pelaporan:** Menambahkan fitur cetak laporan (PDF/Excel) untuk rekapitulasi penggunaan bahan baku bulanan.
3. **Notifikasi Eksternal:** Mengintegrasikan notifikasi via email atau aplikasi pesan instan jika ada bahan baku yang mendekati tanggal kadaluarsa atau stok menipis.

## DAFTAR PUSTAKA

[1] Pandey, A. (2018). *jdbc-dao-template*. GitHub Repository. Diakses dari https://github.com/abhishekpandey528/jdbc-dao-template

[2] Thamrin, I., Arsyad, Z., & Permana, Y. A. *Buku Ajar Mata Kuliah Pemrograman Berorientasi Objek (Teori)*. Bandung: Politeknik Negeri Bandung.