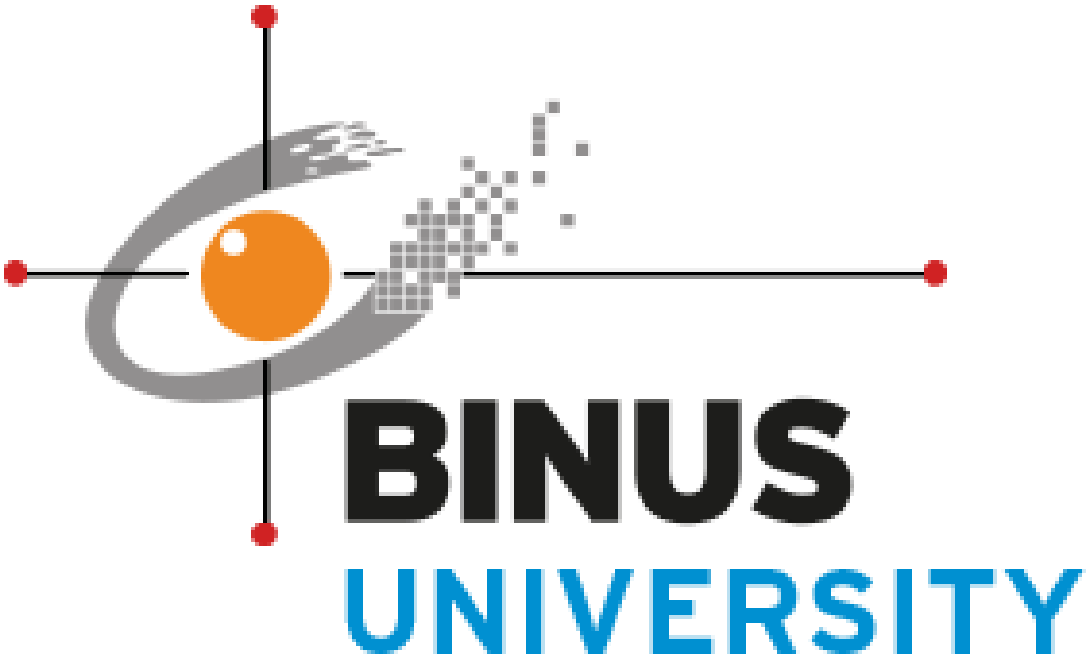
Laporan Project AOL

Framework Layer Architecture

GoodDeed: Sistem Donasi Uang atau Barang



Kelompok 15

1. Fadhel Baihaqi Rizanda - 2602086476
2. Khalid Fatur Rahman - 2602111654
3. Denny Suwando Pratama - 2602166211

2024

1.) Pendahuluan

1.1 Latar Belakang

GoodDeed adalah sebuah platform digital inovatif yang dirancang untuk memfasilitasi donasi uang dan barang dari para donatur kepada individu atau organisasi yang membutuhkan. Proyek ini bertujuan untuk mengatasi masalah transparansi, efisiensi, dan jangkauan dalam proses donasi tradisional.

1.2 Tujuan Proyek

Tujuan utama dari aplikasi GoodDeed adalah:

- **Meningkatkan Transparansi:** Memastikan setiap donasi dapat dilacak secara jelas, mulai dari donatur hingga detail donasinya.
- **Meningkatkan Efisiensi:** Mengoptimalkan proses donasi agar lebih efisien, cepat, dan mudah digunakan.
- **Memperluas Jangkauan:** Menjangkau lebih banyak donatur potensial dan penerima manfaat melalui platform digital yang mudah diakses.

2.) Metodologi

Metodologi yang digunakan dalam proyek GoodDeed adalah Object-Oriented Programming (OOP). Pendekatan ini dipilih karena kemampuannya dalam memodelkan sistem yang kompleks menjadi objek-objek yang saling berinteraksi, sehingga memudahkan pengembangan, pengujian, dan pemeliharaan. Aplikasi ini juga menggunakan design pattern seperti abstract factory, adapter, singleton, dan facade. Sehingga dapat menghasilkan sistem yang memnuhi prinsip SOLID.

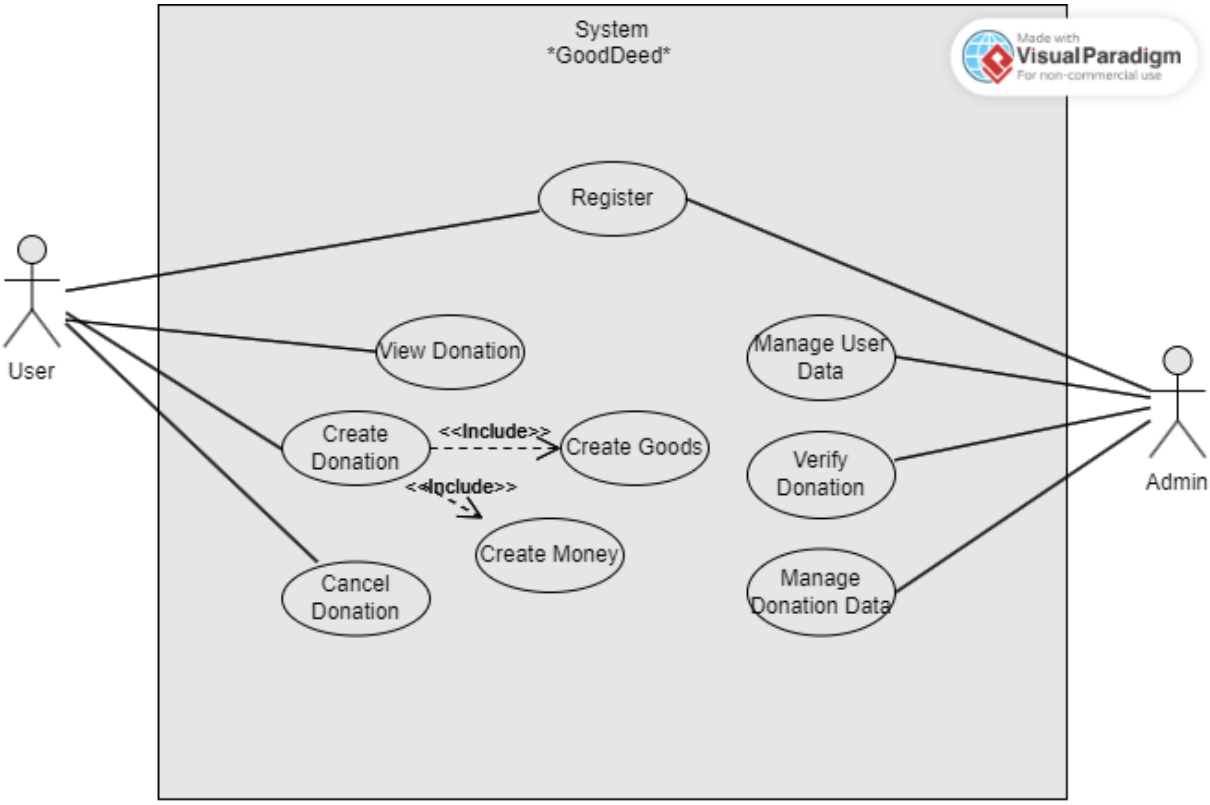
Analisis dan Perancangan Sistem

Tahap awal proyek akan difokuskan pada analisis kebutuhan pengguna dan perancangan model objek yang merepresentasikan entitas-entitas dalam sistem, seperti Donatur, Penerima, Donasi, dan sebagainya.

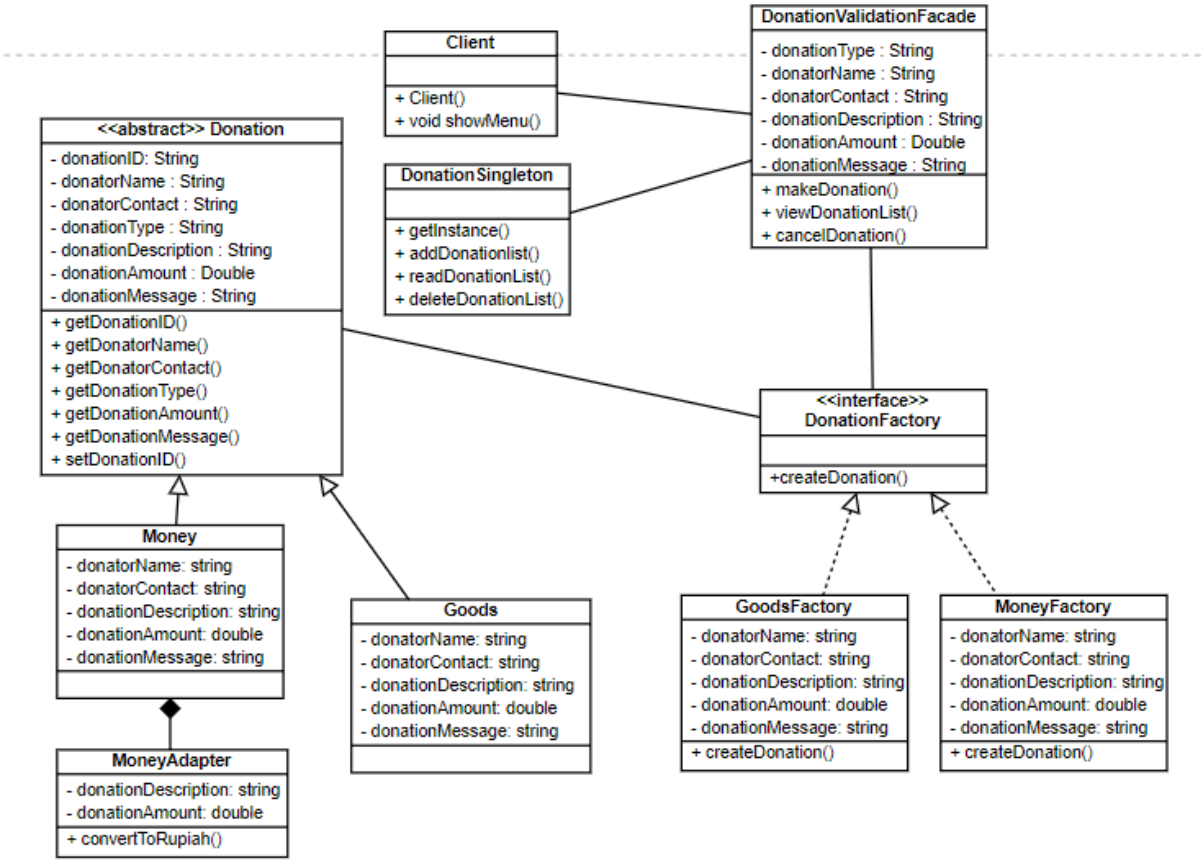
- Use Case Diagram: Digunakan untuk menggambarkan interaksi antara pengguna dan sistem.

- Class Diagram: Digunakan untuk memodelkan struktur kelas dan hubungan antar kelas.

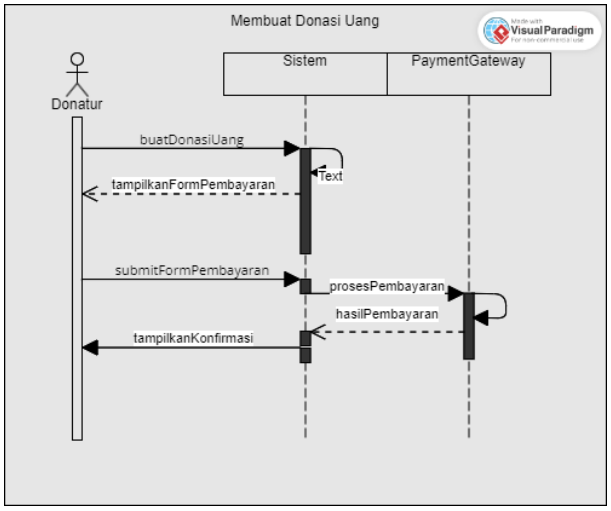
- Sequence Diagram: Digunakan untuk memodelkan urutan interaksi antara objek, contoh (Membuat Donasi Uang).



Use Case Diagram: Menunjukkan aktor-aktor (Donatur dan Admin) berinteraksi dengan sistem (GoodDeed).



Class Diagram: Diagram ini menggambarkan sistem pengelolaan donasi yang sederhana. Pengguna (Client) berinteraksi dengan sistem melalui antarmuka yang disederhanakan (DonationValidationFacade). Sistem ini dapat menangani donasi dalam bentuk uang (Money) atau barang (Goods), yang direpresentasikan oleh kelas turunan dari kelas abstrak Donation. DonationSingleton memastikan hanya ada satu daftar donasi yang dikelola, sementara MoneyAdapter menangani konversi mata uang jika diperlukan. Pembuatan objek donasi dilakukan melalui antarmuka DonationFactory dan kelas turunannya, MoneyFactory dan GoodsFactory, tergantung pada jenis donasi.



Sequence Diagram: Menunjukkan urutan interaksi antara objek-objek dalam sistem untuk skenario tertentu (membuat donasi uang). Dalam contoh ini, Donatur berinteraksi dengan Sistem, dan Sistem berinteraksi dengan PaymentGateway.

Tools dan Teknologi yang digunakan dalam membangun sistem:

- IDE: Eclipse untuk pengembangan Java.
- UML Tools: Visual Paradigm untuk membuat diagram UML.
- Microsoft Word: untuk penulisan laporan

3.) Solusi Design Pattern yang Dibentuk

Dalam menganalisis persyaratan perangkat lunak dan mengidentifikasi masalah untuk sistem donasi GoodDeed, pendekatan yang digunakan melibatkan beberapa langkah dan desain pattern yang terstruktur. Pertama, kami menentukan kelas abstract untuk nantinya diimplementasikan berdasarkan model donasi, yaitu Money dan Goods. Model ini memiliki atribut sebagai berikut:

- String donationID; // id donasi didapatkan secara random dari function setDonationID()
- String donatorName;
- String donatorContact;
- String donationType; // memilih antara Money atau Goods
- String donationDescription; // jika donation type yang dipilih adalah Money maka untuk description harus diisi dengan kurs mata uangnya(dollar, euro, rupiah). Sedangkan jika Goods maka diisi nama barangnya
- Double donationAmount; // Money (jumlah uang), Goods (jumlah barang)

- String donationMessage; // pesan pribadi (opsional)

Untuk pembuatan objek Money dan Goods, kami menggunakan desain pattern Abstract Factory Method. Abstract Factory Method memungkinkan pembuatan keluarga objek terkait atau bergantung tanpa menentukan kelas konkret mereka. Dengan ini, sistem dapat dengan mudah diperluas untuk jenis donasi baru di masa depan tanpa mengubah kode yang ada.
Selanjutnya, untuk menangani konversi nilai uang berdasarkan kursnya (dollar, euro, rupiah) menjadi rupiah, kami menggunakan desain pattern Adapter. Adapter memungkinkan objek dengan antarmuka yang tidak kompatibel untuk bekerja sama. Dalam konteks ini, Adapter digunakan untuk mengubah nilai donasi uang ke dalam satuan mata uang yang sama (rupiah) agar memudahkan pencatatan dan pelaporan.

Kemudian, untuk memastikan bahwa koneksi ke database (vektor list) hanya dibuat satu kali selama siklus hidup aplikasi dan menyediakan titik akses global ke basis data tersebut, kami menerapkan desain pattern Singleton. Singleton memastikan satu-satunya instance dari suatu kelas dibuat dan memberikan cara untuk mengakses instance tersebut.

Terakhir, untuk menyederhanakan interaksi dengan program menu dan validasi yang kompleks, kami menggunakan desain pattern Facade. Facade menyediakan antarmuka yang lebih sederhana ke subsistem yang lebih kompleks. Dengan menggunakan Facade, kami dapat menyembunyikan kerumitan logika bisnis dan validasi menu dari client, sehingga dapat membuat sistem lebih mudah digunakan dan dikelola.
Dengan kombinasi metodologi ini, kami memastikan bahwa sistem donasi GoodDeed dirancang dengan baik, mudah diperluas, dan dikelola, serta menyediakan pengalaman pengguna yang lancar dan intuitif.

4.) Implementasi Pada Java

Langkah langkah yang dilakukan dalam mengimplementasikan proses dari setiap layer pada sistem aplikasi GoodDeed:

1. Pembuatan Model Objek, dibuat 1 abstract kelas yang nantinya akan diimplementasikan ke setiap model objek. Pada interface telah diinisialisasi beberapa atribut serta penentuan random id yang nantinya akan digunakan pada donationID.

```
package model;

import java.util.Random;

public abstract class Donation {
    String donationID;
    String donatorName;
    String donatorContact;
    String donationType;
    String donationDescription;
    Double donationAmount;
    String donationMessage;

    public Donation(String donatorName, String donatorContact, String donationType,
            String donationDescription, Double donationAmount, String donationMessage) {
        super();
        this.donationID = setDonationID();
        this.donatorName = donatorName;
        this.donatorContact = donatorContact;
        this.donationType = donationType;
        this.donationDescription = donationDescription;
        this.donationAmount = donationAmount;
        this.donationMessage = donationMessage;
    }

    public String getDonationID() {
        return donationID;
    }

    public String getDonatorName() {
        return donatorName;
    }

    public String getDonatorContact() {
        return donatorContact;
    }

    public String getDonationType() {
        return donationType;
    }

    public String getDonationDescription() {
        return donationDescription;
    }

    public Double getDonationAmount() {
        return donationAmount;
    }

    public String getDonationMessage() {
        return donationMessage;
    }

    public String setDonationID() {
        Random random = new Random();
        StringBuilder accountNumber = new StringBuilder();
        for(int i=0;i<12;i++) {
            accountNumber.append(random.nextInt(10));
        }
        return accountNumber.toString();
    }
}
```

```
package model;

public class Goods extends Donation {

    public Goods(String donatorName, String donatorContact, String donationDescription, Double donationAmount, String donationMessage) {
        super(donatorName, donatorContact, "Goods", donationDescription, donationAmount, donationMessage);
        // TODO Auto-generated constructor stub
    }

}
```

```
package model;

public class Money extends Donation{

    public Money(String donatorName, String donatorContact, String donationDescription, Double donationAmount, String donationMessage) {
        super(donatorName, donatorContact, "Money", donationDescription, donationAmount, donationMessage);
        // TODO Auto-generated constructor stub
    }

}
```

2. Pengimplementasian Abstract Factory Method, dibuat 1 kelas interface yang memiliki function createDonation yang nantinya akan ditentukan isinya pada kelas factory turunannya, sehingga dapat membuat objek sesuai dengan kelasnya masing-masing

```java
package abstractFactory;

import model.Donation;

public interface DonationFactory {
    Donation createDonation();
}
```

```java
package abstractFactory;

import model.Donation;

public class GoodsFactory implements DonationFactory {

    String donatorName;
    String donatorContact;
    String donationDescription;
    Double donationAmount;
    String donationMessage;

    public GoodsFactory(String donatorName, String donatorContact, String donationDescription,
            Double donationAmount, String donationMessage) {
        super();
        this.donatorName = donatorName;
        this.donatorContact = donatorContact;
        this.donationDescription = donationDescription;
        this.donationAmount = donationAmount;
        this.donationMessage = donationMessage;
    }

    @Override
    public Donation createDonation() {
        // TODO Auto-generated method stub
        return new Goods(donatorName, donatorContact, donationDescription, donationAmount, donationMessage);
    }

}
```

```java
package abstractFactory;

import model.Donation;

public class MoneyFactory implements DonationFactory {

    String donatorName;
    String donatorContact;
    String donationDescription;
    Double donationAmount;
    String donationMessage;

    public MoneyFactory(String donatorName, String donatorContact, String donationDescription,
            Double donationAmount, String donationMessage) {
        super();
        this.donatorName = donatorName;
        this.donatorContact = donatorContact;
        this.donationDescription = donationDescription;
        this.donationAmount = donationAmount;
        this.donationMessage = donationMessage;
    }

    @Override
    public Donation createDonation() {
        // TODO Auto-generated method stub
        return new Money(donatorName, donatorContact, donationDescription, donationAmount, donationMessage);
    }

}
```

3. Pengimplementasian Singleton Method, singleton memastikan hanya satu instance dari setiap kelas yang dibuat. Pada kelas singleton terdapat juga fungsi untuk menambahkan, menampilkan, dan menghapus donasi pada vektor list yang telah ditentukan.

```java
package singleton;

import java.util.Iterator;

public class DonationSingleton {
    Vector<Donation> donationList;

    private static volatile DonationSingleton instance;

    public DonationSingleton(Vector<Donation> donationList) {
        super();
        this.donationList = donationList;
    }

    public static DonationSingleton getInstance(Vector<Donation> donationList) {
        if(instance == null) {
            synchronized (DonationSingleton.class) {
                if(instance == null) {
                    return new DonationSingleton(donationList);
                }
            }
        }
        return instance;
    }

    public void addDonationlist(Donation newDonation) {
        donationList.add(newDonation);
    }

    public Vector<Donation> readDonationList(){
        return this.donationList;
    }

    public void deleteDonationList(String donationID) {
        Iterator<Donation> iterator = donationList.iterator();
        while(iterator.hasNext()) {
            Donation donation = iterator.next();
            if(donation.getDonationID().equals(donationID)) {
                iterator.remove();
            }
        }
    }

}
```

4. Pengimplementasian Adapter Desing Pattern, Adapter digunakan untuk mengubah nilai donasi uang ke dalam satuan mata uang yang sama (rupiah) agar memudahkan pencatatan dan pelaporan.

```java
package adapter;

public class MoneyAdapter {
    String donationDescription;
    Double donationAmount;

    public MoneyAdapter() {

    }

    public Double convertToRupiah(String donationDescription, Double donationAmount) {
        if(donationDescription.equals("Dollar")) {
            return donationAmount = Math.round(donationAmount * 16257) * 1.0;
        }
        else if(donationDescription.equals("Euro")) {
            return donationAmount = Math.round(donationAmount * 17702) * 1.0;
        }
        else if(donationDescription.equals("Rupiah")) {
            return donationAmount = donationAmount * 1.0;
        }
        return donationAmount;
    }
}
```

5. Pengimplementasian Facade Design Pattern, Facade menyediakan tampilan menu yang kompleks sehingga menjadi lebih sederhana saat dipanggil ke subsistem;

```java
package facade;

import abstractFactory.GoodsFactory;

public class DonationValidationFacade {
    Vector<Donation> donationList = new Vector<>();
    DonationSingleton database = DonationSingleton.getInstance(donationList);

    Scanner scan = new Scanner(System.in);


    public void makeDonation() {
        String donationType;
        String donatorName;
        String donatorContact;
        String donationDescription;
        Double donationAmount;
        String donationMessage;

        do {
            System.out.print("Input Donation Type [Money | Goods]: ");
            donationType = scan.nextLine();
        } while (!(donationType.equals("Money") || donationType.equals("Goods")));

        do {
            System.out.print("Input Your Name[maximum 20 characters]: ");
            donatorName = scan.nextLine();
        } while (donatorName.trim().equals("") || donatorName.length() > 20);

        do {
            System.out.print("Input Your Phone Number [must contain 9 - 12 numbers]: +62 ");
            donatorContact = scan.nextLine();
        } while (donatorContact.length() < 9 || donatorContact.length() > 12);

        if (donationType.equals("Money")) {
            do {
                System.out.print("Input Your Currency [Dollar | Euro | Rupiah]: ");
                donationDescription = scan.nextLine();
            } while (!(donationDescription.equals("Dollar") || donationDescription.equals("Euro") || donationDescription.equals("Rupiah")));

            System.out.print("Input Money Amount[must be a number]: ");
            donationAmount = scan.nextDouble();
            scan.nextLine();
        } else {
            do {
                System.out.print("Input Your Goods Name[maximum 20 characters]: ");
                donationDescription = scan.nextLine();
            } while (donationDescription.equals("") || donationDescription.length() > 20);

            System.out.print("Input Goods Quantity[must be a number]: ");
            donationAmount = scan.nextDouble();
            scan.nextLine();
        }

        System.out.print("Input Your Message[optional]: ");
        donationMessage = scan.nextLine();

        if (donationType.equals("Money")) {
            MoneyFactory mf = new MoneyFactory(donatorName, donatorContact, donationDescription, donationAmount, donationMessage);
            Donation donateMoney = mf.createDonation();
            database.addDonationList(donateMoney);
        } else {
            GoodsFactory gf = new GoodsFactory(donatorName, donatorContact, donationDescription, donationAmount, donationMessage);
            Donation donateGoods = gf.createDonation();
            database.addDonationList(donateGoods);
        }
    }
```

```java
public void viewDonationList() {
    Vector<Donation> viewDonationList = database.readDonationList();
    MoneyAdapter moneyAdapter = new MoneyAdapter();
    if (viewDonationList.isEmpty()) {
        System.out.println("No Donations Made Yet");
        return;
    } else {
        System.out.println("=============================================================================================================================================");
        System.out.printf("|| %-167s ||\n","                                                                GoodsDeed Donation List");
        System.out.println("=============================================================================================================================================");
        System.out.printf("|| %-2s || %-12s || %-20s || %-15s || %-15s || %-20s || %-30s || %-25s ||\n",
                "No", "Donation ID", "    Donator Name", "Donator Contact", " Donation Type", "Donation Description", "        Donation Ammount", "    Donation Message");
        System.out.println("=============================================================================================================================================");

        int no = 1;

        for (Donation donation : viewDonationList) {
            String format = "|| %-2d || %-12s || %-20s || %-15s || %-15s || %-20s || %-30s || %-25s ||\n";
            if (donation.getDonationType().equals("Money")) {
                if(donation.getDonationDescription().equals("Dollar") || donation.getDonationDescription().equals("Euro")) {
                    System.out.printf(format, no++, donation.getDonationID(), donation.getDonatorName(), donation.getDonatorContact(), donation.getDonationType(), donation.getDonationDescription(), donation.getDonationAmount()
                            + " (Rp " + moneyAdapter.convertToRupiah(donation.getDonationDescription(), donation.getDonationAmount()) + ")", donation.getDonationMessage());
                }
                else {
                    System.out.printf(format, no++, donation.getDonationID(), donation.getDonatorName(), donation.getDonatorContact(), donation.getDonationType(), donation.getDonationDescription(),
                            "Rp " + donation.getDonationAmount(), donation.getDonationMessage());
                }
            } else {
                System.out.printf(format, no++, donation.getDonationID(), donation.getDonatorName(), donation.getDonatorContact(), donation.getDonationType(), donation.getDonationDescription(),
                        +donation.getDonationAmount() + " items", donation.getDonationMessage());
            }
        }

        System.out.println("=============================================================================================================================================");
    }
}
```

```java
    public void cancelDonation() {
        Vector<Donation> cancelDonationList = database.readDonationList();
        String cancelDonationID;
        String cancelProcess = "N";

        viewDonationList();

        if (cancelDonationList.isEmpty()) {
            return;
        }
        System.out.println();
        do {
            System.out.print("Input Donation ID to cancel donation: ");
            cancelDonationID = scan.nextLine();

            for (Donation donation : cancelDonationList) {
                if (cancelDonationID.equals(donation.getDonationID())) {
                    database.deleteDonationList(cancelDonationID);
                    System.out.println("Your Cancelation Process Succeed");
                    System.out.println("Your Donation Has Been Canceled");
                    return;
                }
            }

            System.out.println("Invalid Donation ID. Please try again");
            do {
                System.out.print("Do you want to continue Process [Y|N]: ");
                cancelProcess = scan.nextLine();
            } while (!(cancelProcess.equals("Y") || cancelProcess.equals("N")));

        } while (cancelProcess.equalsIgnoreCase("Y"));
        System.out.println("Your Cancelation Process Has Been Terminated");
    }
}
```

6. Pengimplementasian di kelas main, disini kelas main memanggil kelas facade yang telah menyimpan validasi menu sehingga yang sebelumnya kompleks menjadi lebih sederhana

```java
package client;

import java.util.Scanner;

public class Client {
    Scanner scan = new Scanner(System.in);
    DonationValidationFacade donationValidationFacade = new DonationValidationFacade();

    public void showMenu() {
        int mainMenu;
        do {
            for (int i = 0; i < 50; ++i) System.out.println();
            System.out.println("============================");
            System.out.println("||         GoodDeed       ||");
            System.out.println("============================");
            System.out.println("|| 1. Make Donation       ||");
            System.out.println("|| 2. View Donations      ||");
            System.out.println("|| 3. Cancel Donation     ||");
            System.out.println("|| 0. Exit                ||");
            System.out.println("============================");
            System.out.print(">> ");
            mainMenu = scan.nextInt();
            scan.nextLine();
            switch (mainMenu) {
            case 1:
                for (int i = 0; i < 50; ++i) System.out.println();
                donationValidationFacade.makeDonation();
                System.out.println("Press Enter to Continue...");
                scan.nextLine();
                break;

            case 2:
                for (int i = 0; i < 50; ++i) System.out.println();
                donationValidationFacade.viewDonationList();
                System.out.println("Press Enter to Continue...");
                scan.nextLine();
                break;

            case 3:
                for (int i = 0; i < 50; ++i) System.out.println();
                donationValidationFacade.cancelDonation();
                System.out.println("Press Enter to Continue...");
                 scan.nextLine();
                break;

            default:
                break;
            }
            if(mainMenu == 0) {
                System.out.println("Thank you for using GoodDeed! Your contributions make a difference.");
                System.out.println("Press Enter to Continue...");
                 scan.nextLine();
                 System.out.println("Bye !!!");
            }
        } while (mainMenu != 0);
    }

    public Client() {
        showMenu();
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        new Client();
    }

}
```

5.) Kesimpulan:

Proyek GoodDeed, sistem donasi uang dan barang, memiliki potensi yang signifikan untuk merevolusi lanskap filantropi. Dengan mengatasi tantangan transparansi, efisiensi, dan jangkauan dalam proses donasi tradisional, GoodDeed bertujuan untuk menciptakan platform yang memberdayakan donatur dan penerima manfaat.

Pendekatan layered architecture yang diadopsi dalam pengembangan GoodDeed memastikan skalabilitas, modularitas, dan kemudahan pemeliharaan sistem. Ini memungkinkan tim pengembang untuk fokus pada peningkatan dan penambahan fitur secara independen, tanpa mengganggu fungsi keseluruhan platform.

Diharapkan bahwa GoodDeed akan menjadi katalisator dalam mendorong partisipasi masyarakat dalam kegiatan donasi. Dengan menyediakan platform yang aman, transparan, dan

mudah digunakan, GoodDeed dapat menginspirasi lebih banyak individu untuk berbagi kebaikan dan memberikan dampak positif bagi mereka yang membutuhkan.

Keberhasilan proyek ini tidak hanya akan diukur dari jumlah donasi yang difasilitasi, tetapi juga dari dampak nyata yang dihasilkan bagi penerima manfaat. GoodDeed berkomitmen untuk terus berinovasi dan mengembangkan platform ini agar dapat melayani kebutuhan masyarakat secara lebih baik dan berkontribusi pada terciptanya dunia yang lebih baik.

6.) Referensi:

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley.
- Fowler, M. (2006). *Patterns of enterprise application architecture*. Addison-Wesley.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *A pattern language for object-oriented programming.* Addison-Wesley.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley. (pp. 109-114)
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley. (pp. 135-140)
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley. (pp. 141-146)
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley. (pp. 243-248)
- Vaskaran Sarcar (2018). Java Design Patterns: A Hands-On Experience with Real-World Examples. -: Apress.