PROGRESS TUBES PBO

NAMA KELOMPOK : Kelompok 10
ANGGOTA :

Nama	NIM	Progress
Edelweiss Salsabila	103052300038	Desain arsitektur dan implementasi BaseEntity, User, DataStore dan manajemen saldo.
Muthia Rezi Aisyah	103052300114	Implementasi Transaction serta penerapan Strategy Pattern dengan ISplitStrategy
Amalia Ananda Putri	103052330078	Pembuatan EvenSplitStrategy dan UnevenSplitStrategy serta uji logika pembagian.
Annisa Azzahra Rahmah	103052300056	Implementasi Group dan pengelolaan anggota serta transaksi (komposisi).
Alishadena Chandrani Noor Riva Hutomo	103052300032	Pembuatan DebtCalculator, SummaryServlet, DashboardServlet integrasi perhitungan akhir, serta dokumentasi dan testing.

DETAIL PROGRESS :

1. Edelweiss Salsabila

Peran: Desain arsitektur sistem serta implementasi BaseEntity, User, dan manajemen saldo.

Tanggung Jawab:

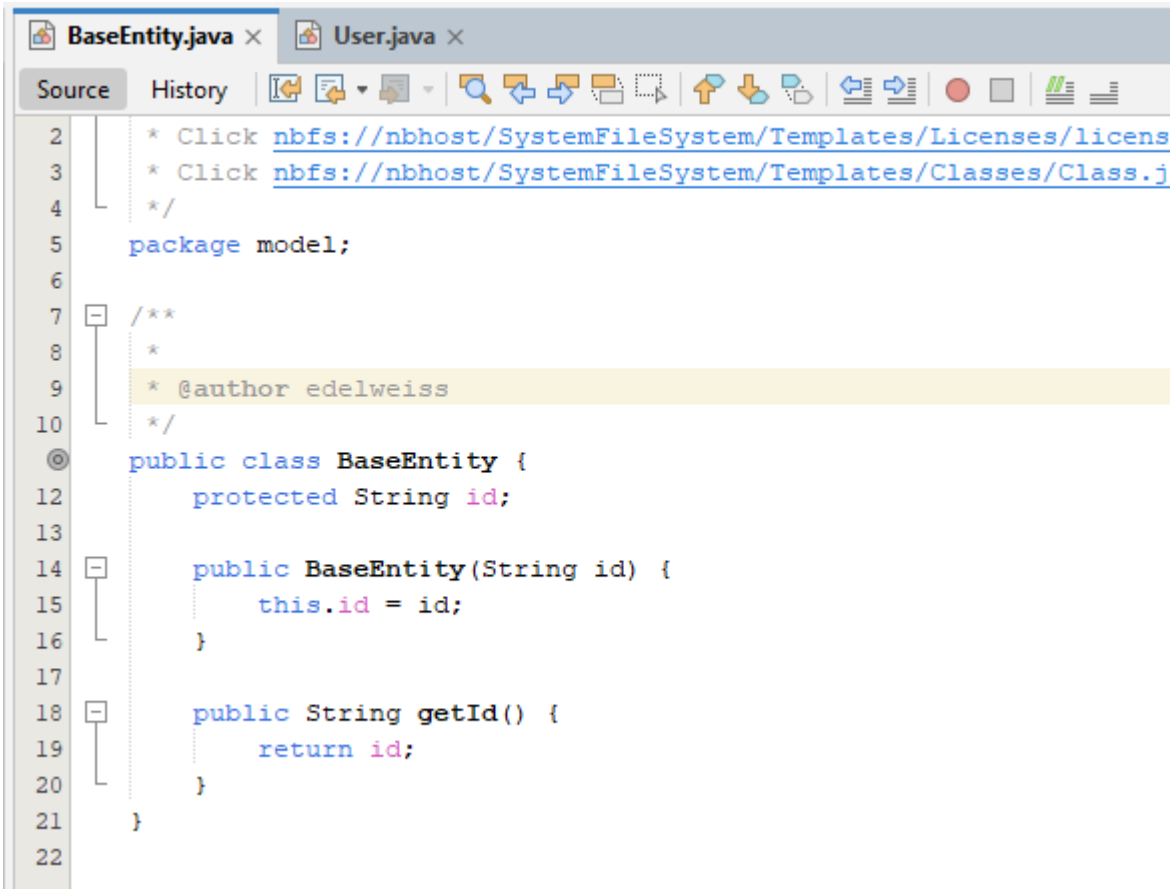
- Merancang struktur kelas utama pada aplikasi SplitBill
- Membuat class BaseEntity sebagai parent class
- Mengimplementasikan class User
- Mengelola fitur saldo (balance) pengguna

Hasil yang Sudah Dikerjakan

- Class BaseEntity berhasil dibuat untuk menyimpan atribut umum seperti id dan createdAt
- Class User berhasil diimplementasikan dengan atribut:
    - name
    - balance
- Method untuk:
    - Menambah saldo
    - Mengurangi saldo
    - Melihat saldo user
- Class User telah digunakan oleh class lain (Transaction dan Group)

Screenshot yang disertakan:

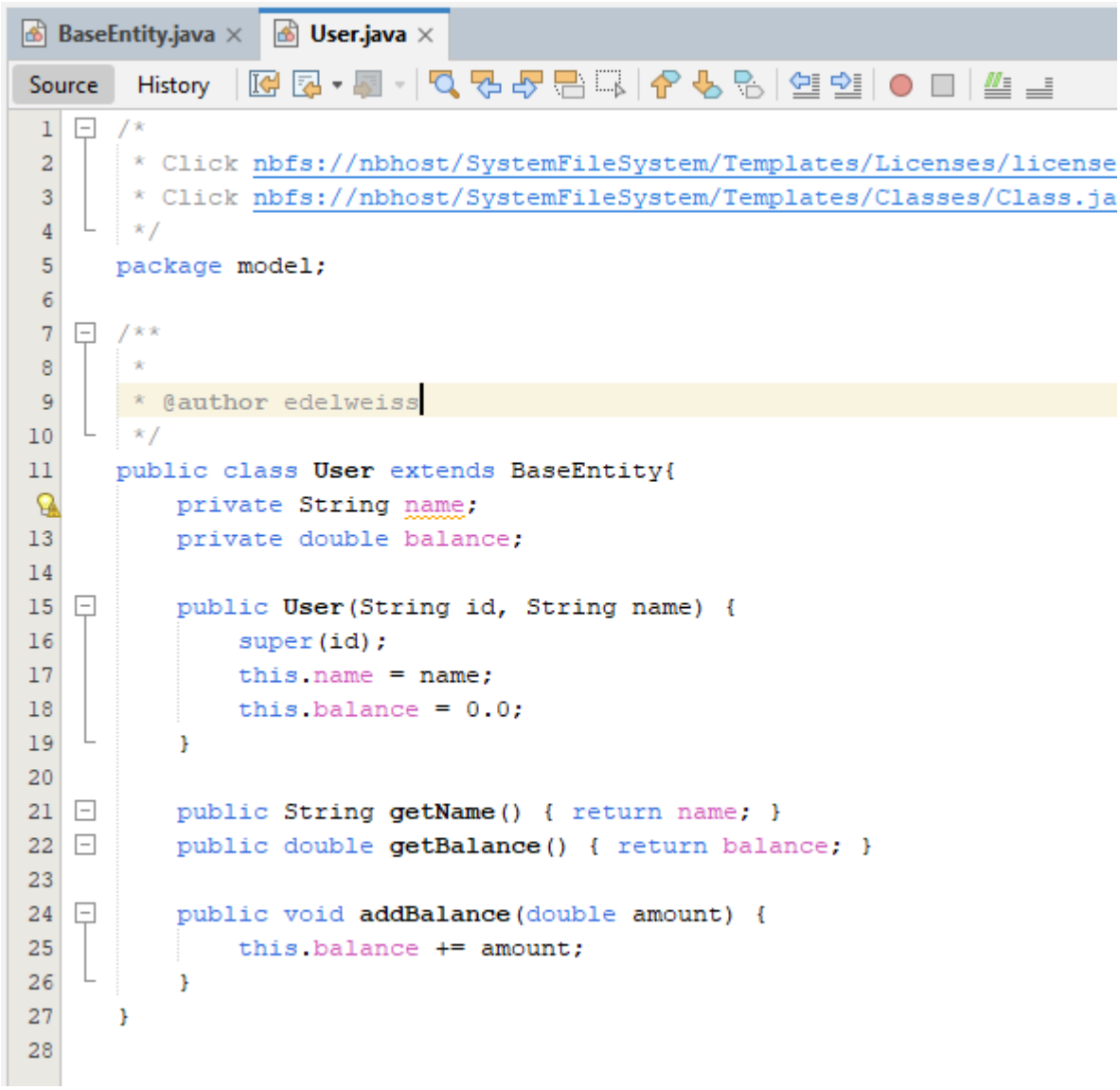
1. baseEntity.java



File BaseEntity.java adalah class dasar yang fungsinya itu sebagai parent class bagi entitas lain dalam aplikasi. Class ini menyimpan atribut id sebagai identitas unik yang dimiliki oleh setiap objek turunan.

Dengan menjadikan BaseEntity sebagai superclass, desain arsitektur aplikasi menjadi lebih terstruktur dan konsisten karena atribut identitas tidak perlu didefinisikan ulang pada setiap class. Konsep pewarisan (inheritance) ini mempermudah pengelolaan data dan meningkatkan keterbacaan kode.

2. user.java (ini ada isinya manajemen saldo)



File User.java adalah class model yang merepresentasikan pengguna pada aplikasi SplitBill. Class ini mewarisi (extends) BaseEntity, sehingga setiap user memiliki identitas unik yang dikelola oleh superclass.

Class User memiliki atribut name untuk menyimpan nama pengguna serta balance untuk menyimpan saldo. Pada saat object User dibuat, saldo secara default diinisialisasi dengan nilai nol.

Method addBalance() digunakan untuk menambahkan saldo pengguna berdasarkan transaksi yang terjadi. Dengan adanya method ini, pengelolaan saldo dilakukan secara terenkapsulasi di dalam class User sehingga perubahan saldo tidak dilakukan secara langsung dari luar class. Pendekatan ini mendukung prinsip enkapsulasi dalam pemrograman berorientasi objek dan memudahkan integrasi saldo user dengan proses perhitungan transaksi pada fitur lain di aplikasi.

3. DataStore.java

DataStore.java

Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package store;

import model.Group;
import model.User;

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

/**
 *
 * @author edelweiss
 */
public class DataStore {
    private static final List<Group> groups = new ArrayList<>();

    static {
        Group g = new Group("G1", "Trip Bandung");
        g.addMember(new User("U1", "Andi"));
        g.addMember(new User("U2", "Budi"));
        g.addMember(new User("U3", "Cici"));
        groups.add(g);
    }

    public static List<Group> getGroups() { return groups; }

    public static Group findGroup(String groupId) {
        for (Group g : groups) if (g.getId().equals(groupId)) return g;
        return null;
    }

    public static User findUserInGroup(Group group, String userId) {
        if (group == null) return null;
        for (User u : group.getMembers()) if (u.getId().equals(userId)) return u;
        return null;
    }
}

```

File DataStore.java berfungsi sebagai penyimpanan data utama (in-memory) dalam aplikasi SplitBill. Class ini menyimpan daftar group beserta anggota di dalamnya dan menyediakan method untuk mengambil serta mencari data group dan user.

DataStore digunakan oleh berbagai servlet sebagai sumber data terpusat, sehingga mendukung desain arsitektur aplikasi yang terstruktur dan memisahkan antara logika bisnis dan penyimpanan data.

2. Muthia Rezi Aisyah

Peran: Implementasi Transaction serta penerapan Strategy Pattern menggunakan ISplitStrategy.

Tanggung Jawab:

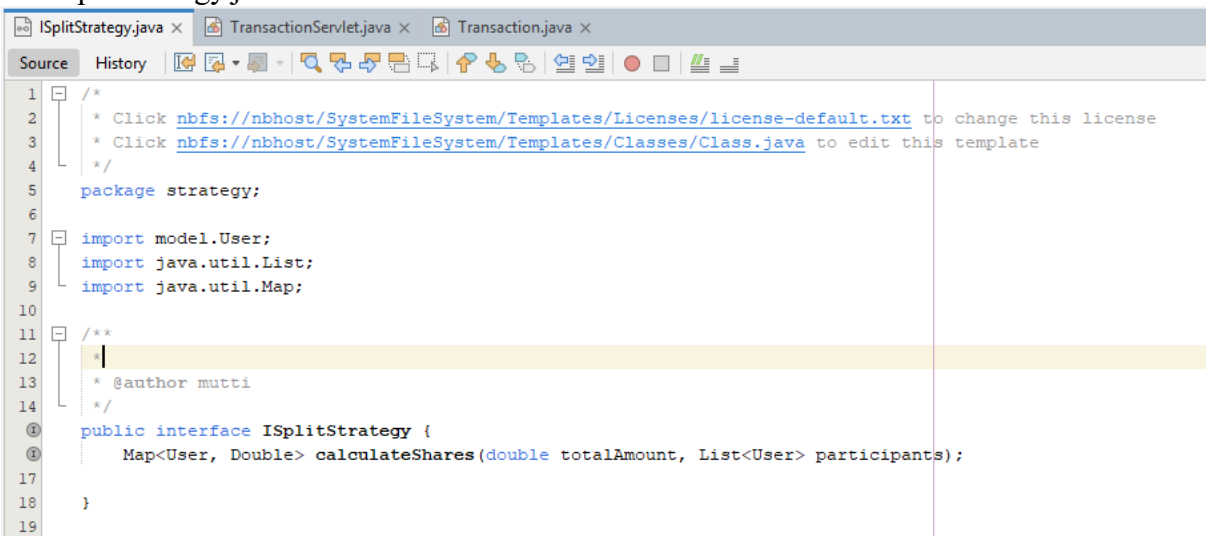
- Membuat class Transaction
- Menerapkan Strategy Pattern untuk metode pembagian tagihan
- Membuat interface ISplitStrategy

Hasil yang Sudah Dikerjakan:

- Class Transaction berhasil dibuat dengan atribut:
  - payer
  - amount
  - participants
  - splitStrategy
- Interface ISplitStrategy berhasil dibuat sebagai kontrak pembagian
- Transaction dapat menggunakan strategy berbeda tanpa mengubah logic utama

Screenshot yang disertakan:

1. ISplitStrategy.java



File ISplitStrategy.java itu sebagai interface yang digunakan untuk menerapkan Strategy Pattern pada proses pembagian tagihan. Interface ini mendefinisikan method calculateShares() sebagai kontrak yang harus diimplementasikan oleh setiap strategi pembagian.

Dengan menggunakan interface ini, aplikasi dapat menerapkan berbagai metode pembagian tagihan tanpa mengubah struktur utama pada class transaksi. Pendekatan ini membuat sistem lebih fleksibel dan mudah dikembangkan di kemudian hari.

2. TransactionServlet.java

ISplitStrategy.java | TransactionServlet.java | Transaction.java

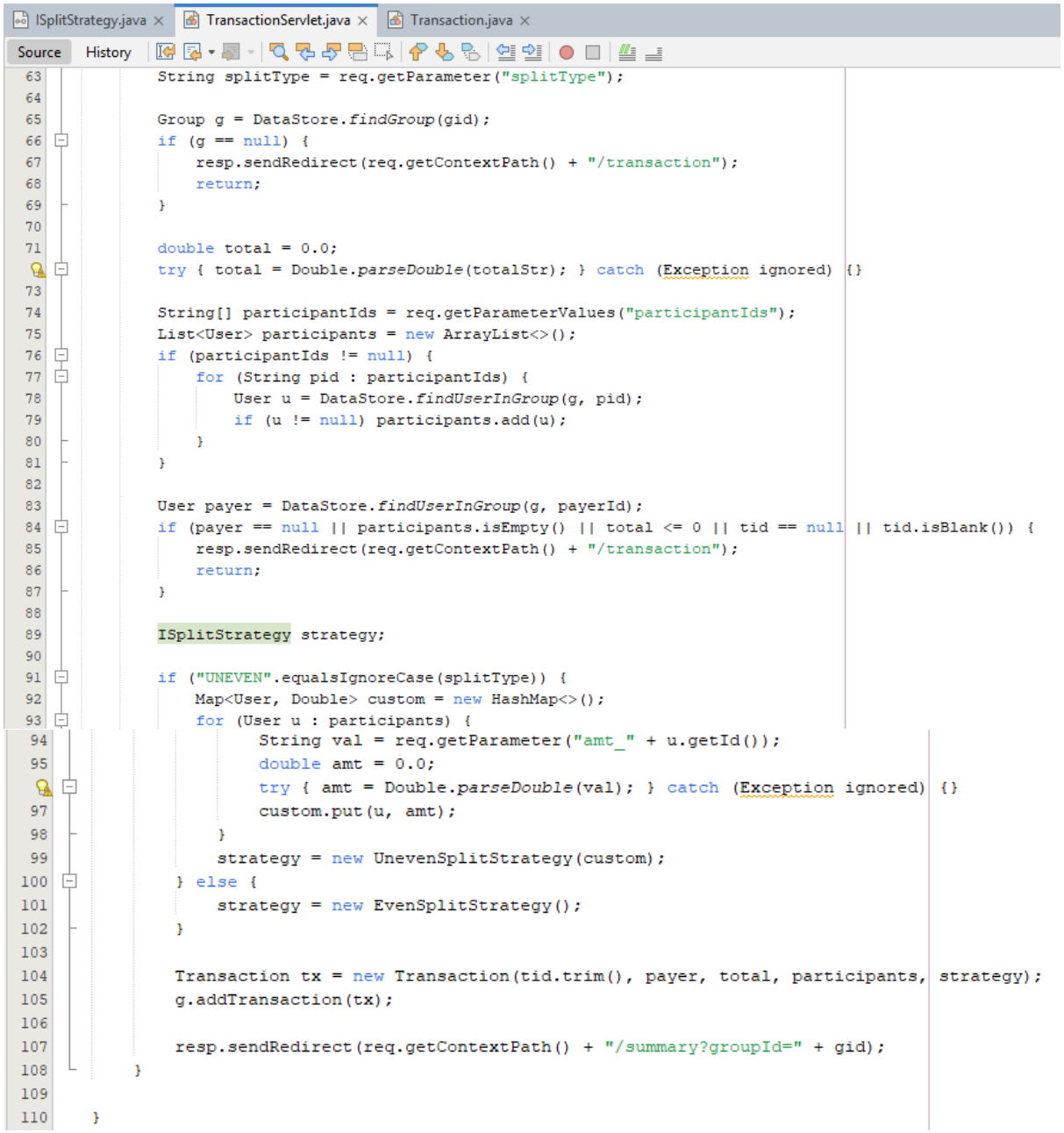
Source | History

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/JSP_Servlet/Servlet.java to edit this template
 */
package servlet;

import model.Group;
import model.Transaction;
import model.User;
import store.DataStore;
import strategy.EvenSplitStrategy;
import strategy.ISplitStrategy;
import strategy.UnevenSplitStrategy;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author mutti
 */
@WebServlet("/transaction")
```

ISplitStrategy.java | TransactionServlet.java | Transaction.java

Source | History

```java
public class TransactionServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {

        List<Group> groups = DataStore.getGroups();
        req.setAttribute("groups", groups);

        String gid = req.getParameter("groupId");
        Group selected = null;

        if (gid != null && !gid.isBlank()) {
            selected = DataStore.findGroup(gid);
        }
        if (selected == null && !groups.isEmpty()) {
            selected = groups.get(0);
        }

        req.setAttribute("selectedGroup", selected);
        req.setAttribute("members", (selected == null) ? new ArrayList<User>() : selected.getMembers());

        RequestDispatcher rd = req.getRequestDispatcher("/WEB-INF/transaction_add.jsp");
        rd.forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String gid = req.getParameter("groupId");
        String tid = req.getParameter("txId");
        String payerId = req.getParameter("payerId");
        String totalStr = req.getParameter("totalAmount");
```



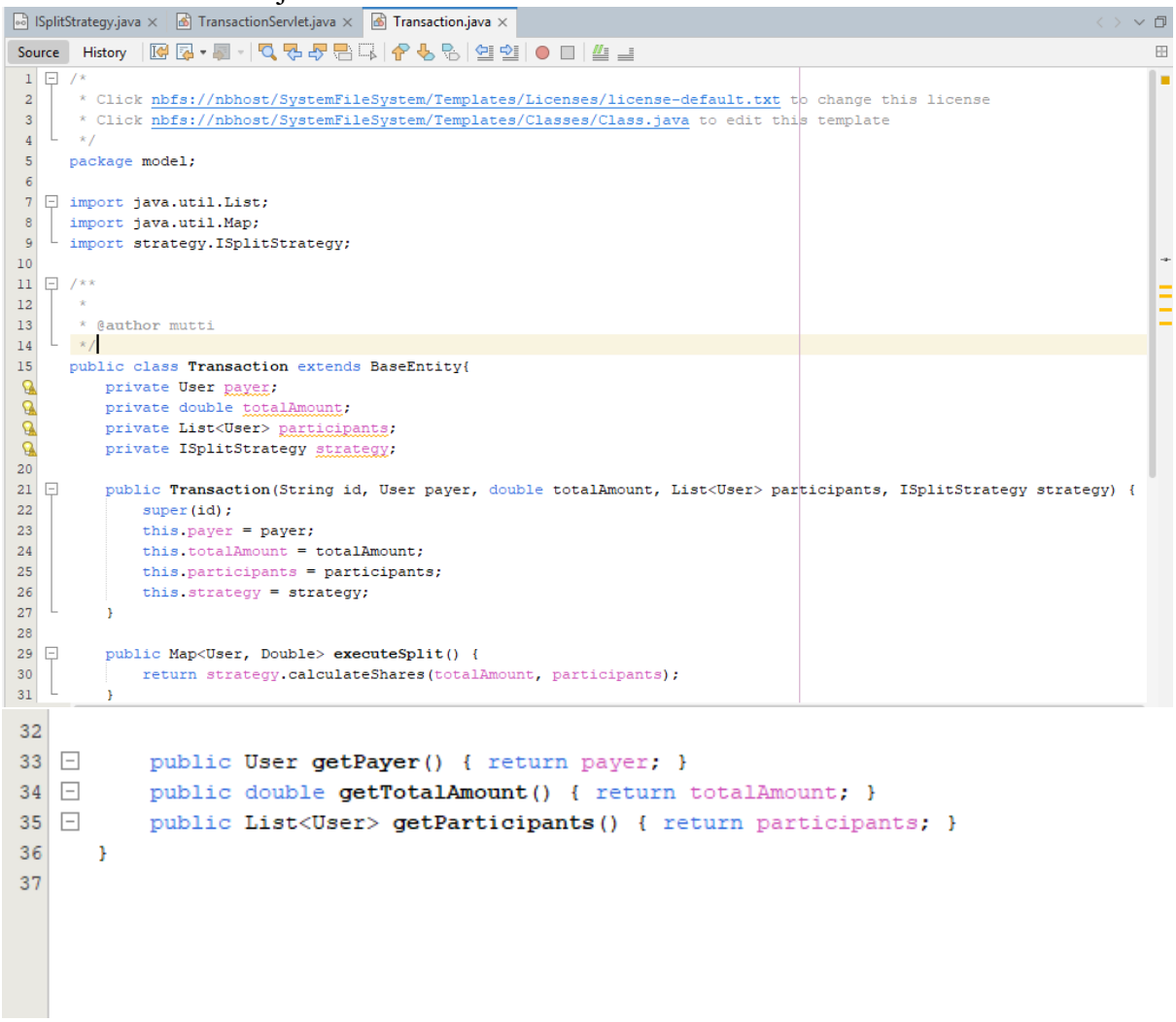
TransactionServlet menentukan strategi pembagian yang digunakan berdasarkan input dari user. Jika tipe pembagian tidak rata dipilih, maka digunakan UnevenSplitStrategy, sedangkan pembagian rata menggunakan EvenSplitStrategy.

Pemilihan strategy dilakukan secara dinamis tanpa mengubah logic transaksi, sehingga memperlihatkan penerapan Strategy Pattern pada level controller.

Setelah strategy ditentukan, servlet membuat object Transaction dengan menyertakan strategy yang dipilih. Transaksi kemudian ditambahkan ke dalam group terkait.

Proses ini menunjukkan integrasi antara servlet, model transaksi, dan strategy pembagian dalam satu alur yang terstruktur sesuai arsitektur MVC.

3. Transaction.java



```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package model;

import java.util.List;
import java.util.Map;
import strategy.ISplitStrategy;

/**
 *
 * @author mutti
 */
public class Transaction extends BaseEntity{
    private User payer;
    private double totalAmount;
    private List<User> participants;
    private ISplitStrategy strategy;

    public Transaction(String id, User payer, double totalAmount, List<User> participants, ISplitStrategy strategy) {
        super(id);
        this.payer = payer;
        this.totalAmount = totalAmount;
        this.participants = participants;
        this.strategy = strategy;
    }

    public Map<User, Double> executeSplit() {
        return strategy.calculateShares(totalAmount, participants);
    }

    public User getPayer() { return payer; }
    public double getTotalAmount() { return totalAmount; }
    public List<User> getParticipants() { return participants; }
}
```

Class Transaction merepresentasikan sebuah transaksi dalam aplikasi SplitBill. Class ini menyimpan informasi mengenai pihak yang membayar, total jumlah transaksi, daftar peserta, serta strategi pembagian yang digunakan.

Penggunaan atribut bertipe ISplitStrategy menunjukkan bahwa proses pembagian tidak terikat pada satu metode tertentu, melainkan dapat diganti sesuai kebutuhan, sehingga menerapkan konsep Strategy Pattern.

Method executeSplit() digunakan untuk menjalankan proses pembagian tagihan dengan memanggil strategy yang telah ditentukan. Dengan cara ini, class Transaction tidak perlu mengetahui detail implementasi pembagian, sehingga tetap sederhana dan mudah dirawat.
Implementasi ini mendukung prinsip *open–closed principle*, di mana sistem terbuka untuk penambahan strategi baru tanpa perlu mengubah kode yang sudah ada.

3. Amalia Ananda Putri
Peran: Pembuatan *EvenSplitStrategy* dan *UnevenSplitStrategy* serta pengujian logika pembagian tagihan.

Tanggung Jawab:
1. Mengimplementasikan interface ISplitStrategy pada dua jenis strategi pembagian tagihan
2. Mengembangkan logika pembagian tagihan secara merata (*Even Split*) dan tidak merata (*Uneven Split*)
3. Mengelola pembagian berbasis jumlah peserta dan nominal khusus per user
4. Menangani validasi data input untuk mencegah kesalahan saat proses pembagian
5. Mengintegrasikan strategi pembagian ke dalam proses transaksi

Hasil yang Sudah Dikerjakan:
• Class EvenSplitStrategy berhasil dibuat dengan:
- Logika pembagian tagihan secara merata
- Perhitungan berdasarkan jumlah peserta

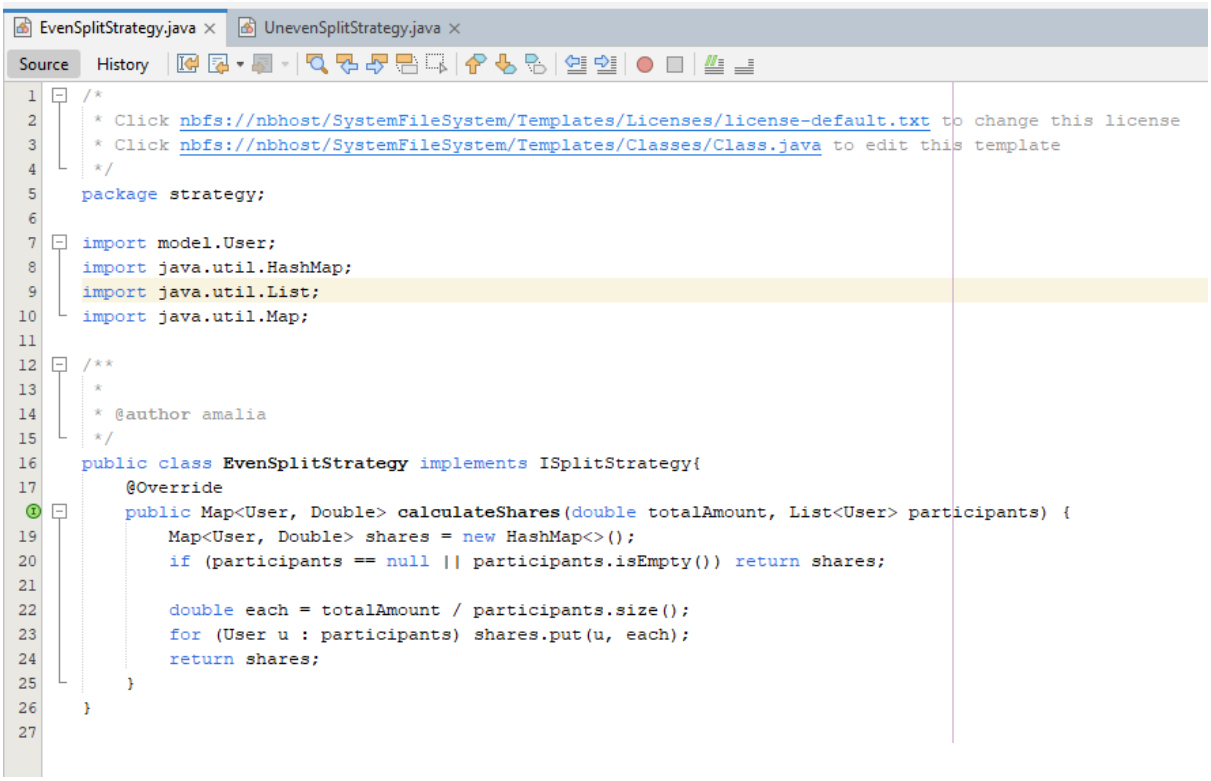
• EvenSplitStrategy dapat:
- Membagi total tagihan sama rata ke seluruh anggota
- Menghasilkan nilai pembagian untuk setiap user

• Class UnevenSplitStrategy berhasil dibuat dengan:
- Logika pembagian tagihan tidak merata
- Penggunaan nominal khusus per user

• UnevenSplitStrategy dapat:
- Membagi tagihan berdasarkan nilai yang ditentukan
- Memberikan nilai default jika nominal user tidak tersedia

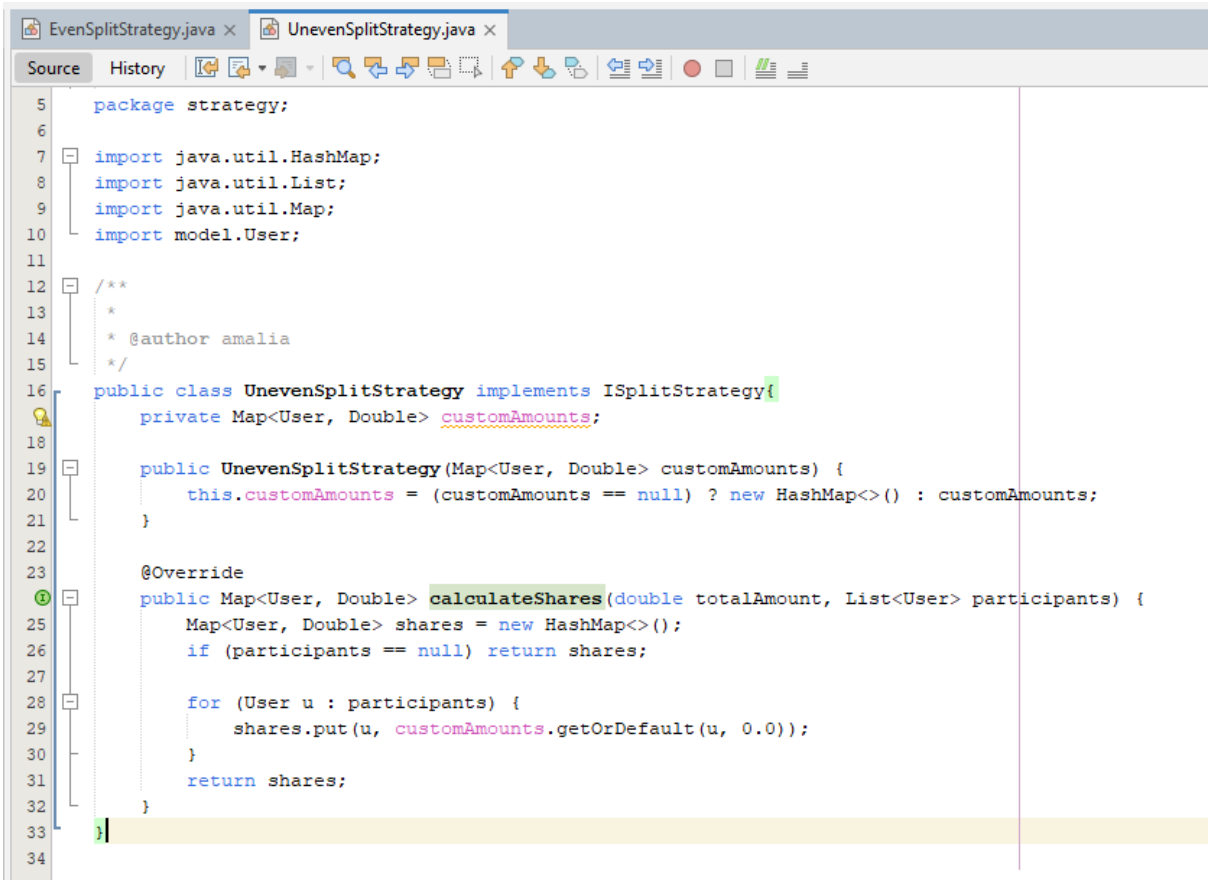
• Strategi pembagian telah:
- Terintegrasi dengan proses transaksi
- Diuji menggunakan beberapa skenario pembagian tagihan

Screenshot yang disertakan:
1.EvenSplitStrategy.java



Class EvenSplitStrategy adalah salah satu implementasi dari interface ISplitStrategy yang digunakan untuk membagi total tagihan secara rata kepada seluruh peserta transaksi.
Metode calculateShares() akan menghitung jumlah tagihan per orang dengan membagi total tagihan dengan jumlah peserta, kemudian menyimpan hasilnya ke dalam struktur data Map.
Dengan pendekatan ini, aplikasi mampu mendukung berbagai metode pembagian tagihan secara fleksibel menggunakan Strategy Pattern.

2. UnevenSplitStrategy.java



```java
package strategy;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import model.User;

/**
 *
 * @author amalia
 */
public class UnevenSplitStrategy implements ISplitStrategy{
    private Map<User, Double> customAmounts;

    public UnevenSplitStrategy(Map<User, Double> customAmounts) {
        this.customAmounts = (customAmounts == null) ? new HashMap<>() : customAmounts;
    }

    @Override
    public Map<User, Double> calculateShares(double totalAmount, List<User> participants) {
        Map<User, Double> shares = new HashMap<>();
        if (participants == null) return shares;

        for (User u : participants) {
            shares.put(u, customAmounts.getOrDefault(u, 0.0));
        }
        return shares;
    }
}
```

Class UnevenSplitStrategy adalah implementasi dari interface ISplitStrategy yang digunakan untuk menangani pembagian tagihan secara tidak merata.
Class ini menyimpan nilai pembagian khusus untuk setiap user dalam bentuk Map<User, Double>. Saat metode calculateShares() dijalankan, sistem akan mengembalikan jumlah yang harus dibayar oleh masing-masing peserta sesuai nilai yang telah ditentukan.
Implementasi ini menerapkan konsep Strategy Pattern, sehingga jenis pembagian tagihan dapat diganti tanpa mengubah class Transaction.

4. Annisa Azzahra Rahmah

Peran: Implementasi Group dan pengelolaan anggota serta transaksi (komposisi).

Tanggung Jawab :

1. Membuat class Group
2. Mengelola relasi antara group, user, dan transaction
3. Mengatur penambahan anggota dan transaksi ke dalam group

Hasil yang Sudah Dikerjakan:

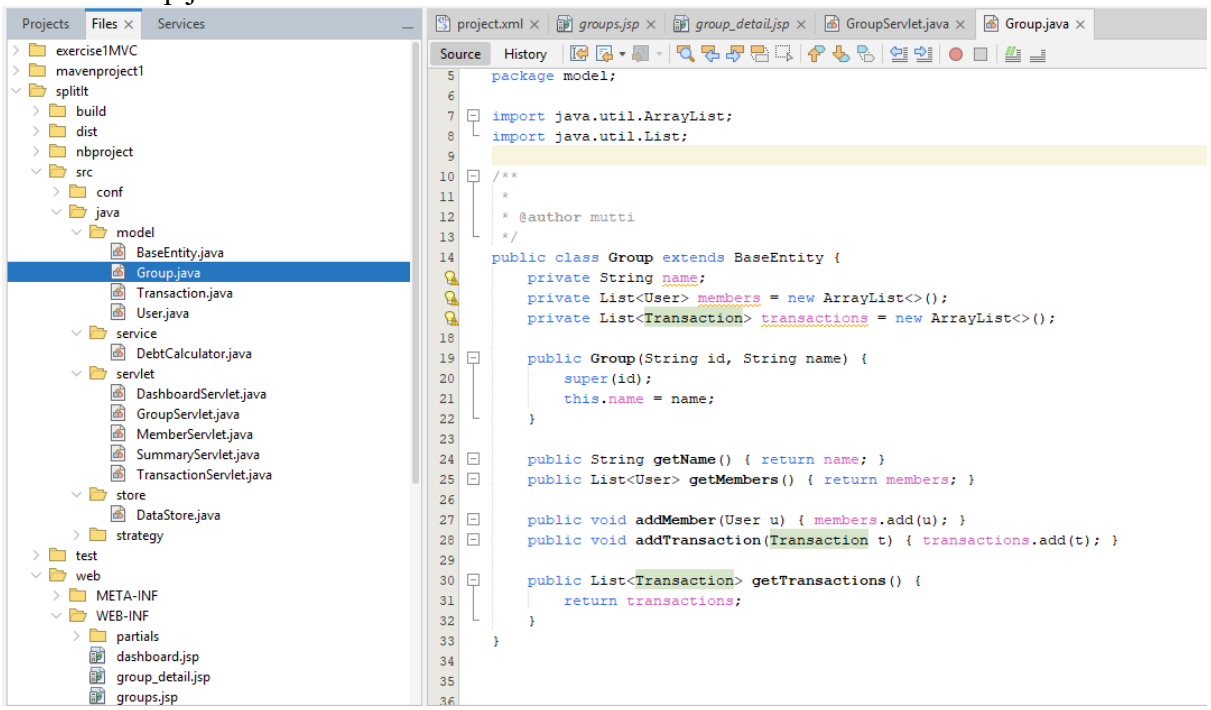
- Class Group berhasil dibuat dengan:
  - Daftar anggota
  - Daftar transaksi

- Group dapat:
  - Menambah anggota
  - Menambahkan transaksi

- Menampilkan daftar transaksi

Screenshot yang disertakan:

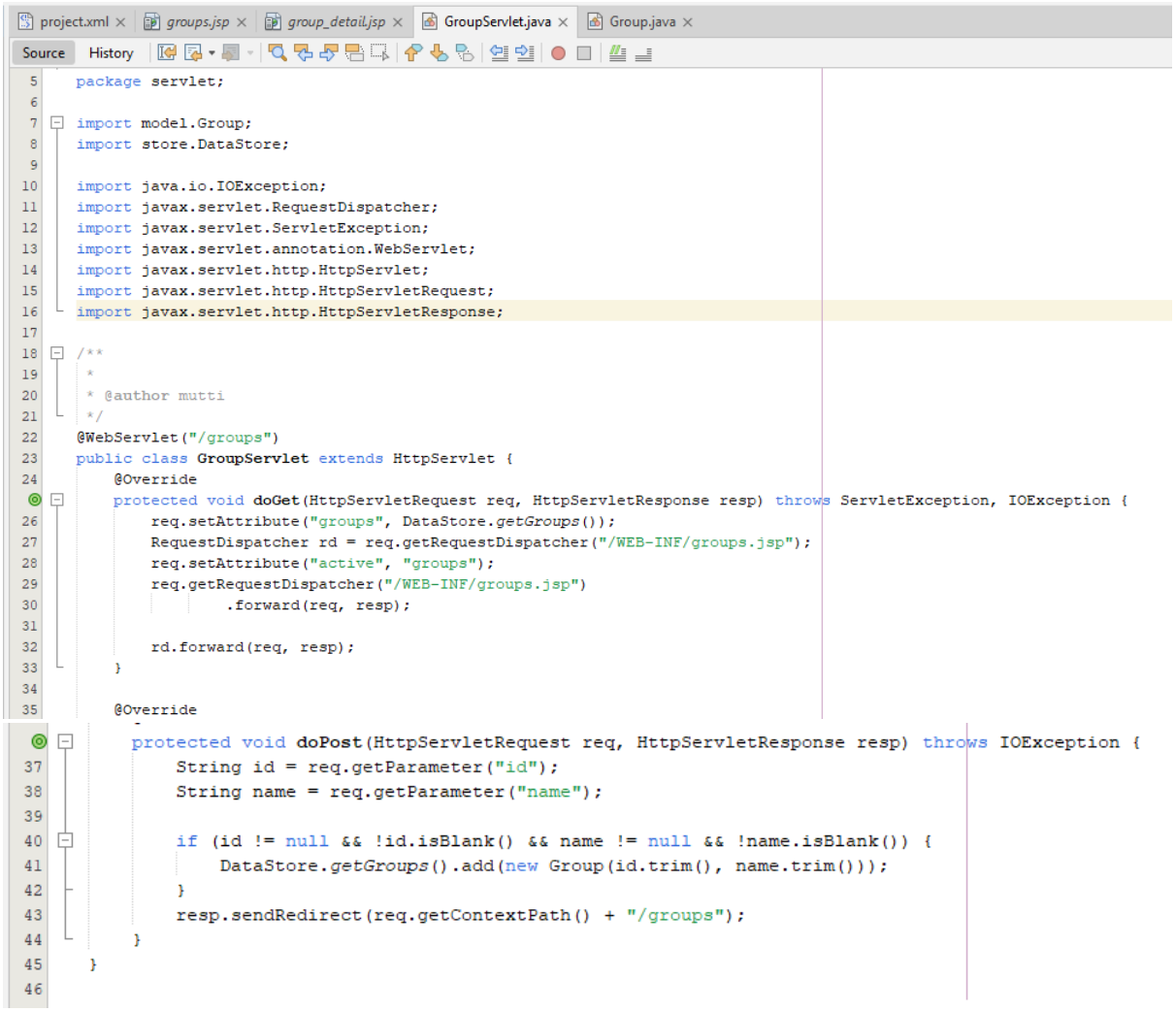
1. Group.java



File Group.java adalah class model yang merepresentasikan entitas group pada aplikasi SplitBill. Class ini mewarisi (extends) BaseEntity sehingga setiap group memiliki identitas unik.

Di dalam class ini terdapat relasi komposisi dengan class User dan Transaction yang direpresentasikan dalam bentuk List<User> dan List<Transaction>. Relasi ini menunjukkan bahwa sebuah group terdiri dari beberapa anggota dan dapat memiliki banyak transaksi.
Selain itu, class Group menyediakan method addMember() untuk menambahkan anggota ke dalam group serta addTransaction() untuk mencatat transaksi yang terjadi di dalam group. Dengan adanya method tersebut, pengelolaan anggota dan transaksi dapat dilakukan secara terpusat melalui object group.

2. GroupServlet.java

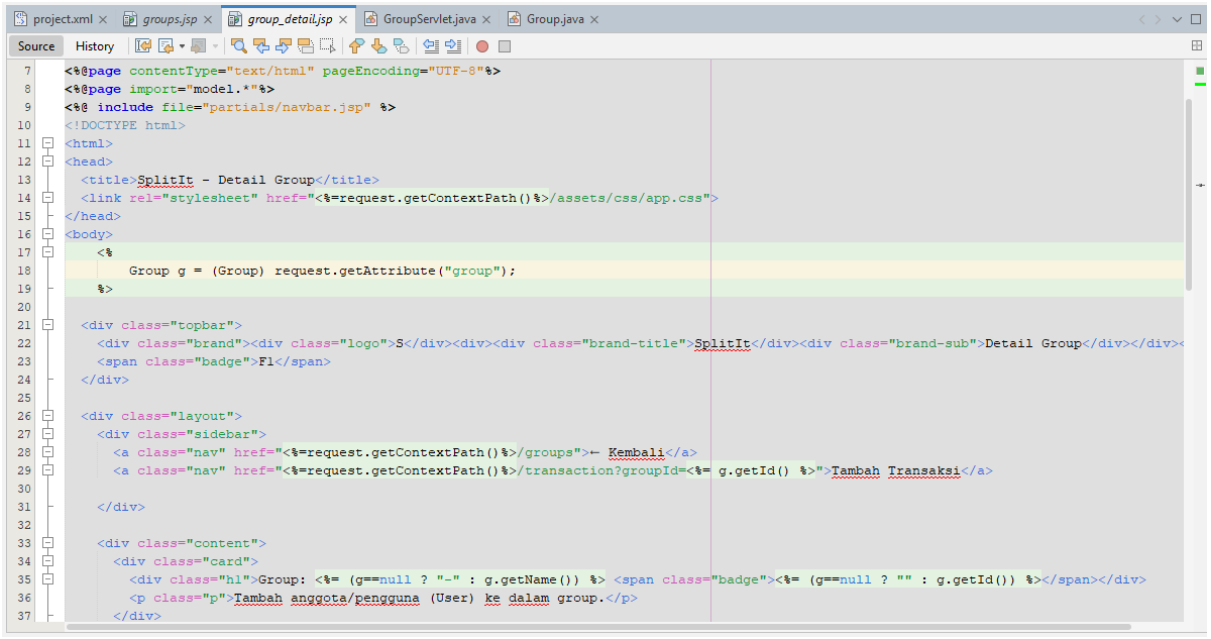


```java
package servlet;

import model.Group;
import store.DataStore;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author mutti
 */
@WebServlet("/groups")
public class GroupServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.setAttribute("groups", DataStore.getGroups());
        RequestDispatcher rd = req.getRequestDispatcher("/WEB-INF/groups.jsp");
        req.setAttribute("active", "groups");
        req.getRequestDispatcher("/WEB-INF/groups.jsp")
                .forward(req, resp);

        rd.forward(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws IOException {
        String id = req.getParameter("id");
        String name = req.getParameter("name");

        if (id != null && !id.isBlank() && name != null && !name.isBlank()) {
            DataStore.getGroups().add(new Group(id.trim(), name.trim()));
        }
        resp.sendRedirect(req.getContextPath() + "/groups");
    }
}
```

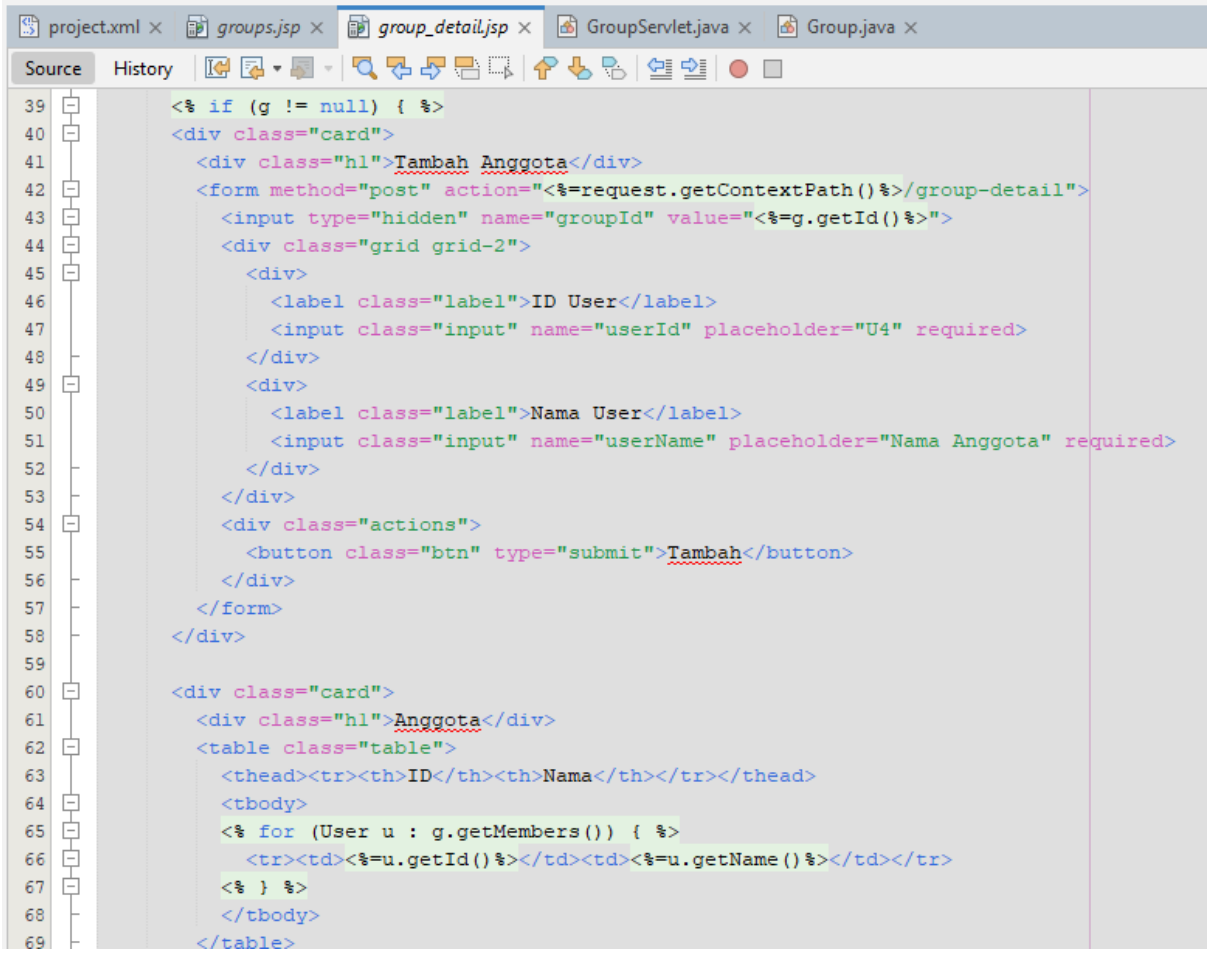
File GroupServlet.java fungsinya adalah sebagai controller di arsitektur MVC yang menangani proses manajemen group. Servlet ini bertugas sebagai penghubung atau connector antara user interface (JSP) dan model Group.

Pada method doGet(), servlet mengambil daftar group dari DataStore kemudian mengirimkannya ke halaman groups.jsp menggunakan mekanisme request.setAttribute(). Proses ini memungkinkan data group ditampilkan pada halaman web.
Pada method doPost(), servlet menangani proses penambahan group baru berdasarkan input dari user. Data yang diterima akan divalidasi secara sederhana kemudian dibuatkan object Group baru dan disimpan ke dalam DataStore. Setelah proses selesai, user akan diarahkan kembali ke halaman daftar group.

3. Group_detail.jsp



```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="model.*"%>
<%@ include file="partials/navbar.jsp" %>
<!DOCTYPE html>
<html>
<head>
  <title>SplitIt - Detail Group</title>
  <link rel="stylesheet" href="<%=request.getContextPath()%>/assets/css/app.css">
</head>
<body>
    <%
        Group g = (Group) request.getAttribute("group");
    %>

  <div class="topbar">
    <div class="brand"><div class="logo">S</div><div><div class="brand-title">SplitIt</div><div class="brand-sub">Detail Group</div></div><
    <span class="badge">Fl</span>
  </div>

  <div class="layout">
    <div class="sidebar">
      <a class="nav" href="<%=request.getContextPath()%>/groups">← Kembali</a>
      <a class="nav" href="<%=request.getContextPath()%>/transaction?groupId=<%= g.getId() %>">Tambah Transaksi</a>

    </div>

    <div class="content">
      <div class="card">
        <div class="hl">Group: <%= (g==null ? "-" : g.getName()) %> <span class="badge"><%= (g==null ? "" : g.getId()) %></span></div>
        <p class="p">Tambah anggota/pengguna (User) ke dalam group.</p>
      </div>
```



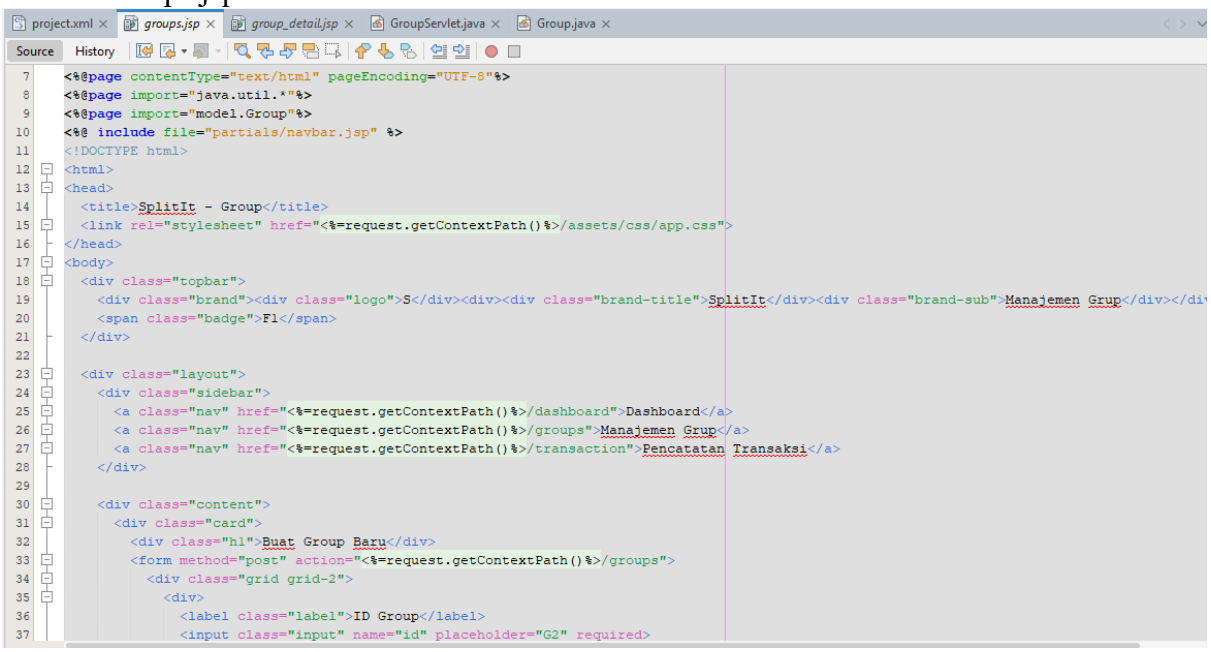
```
      <% if (g != null) { %>
      <div class="card">
        <div class="hl">Tambah Anggota</div>
        <form method="post" action="<%=request.getContextPath()%>/group-detail">
          <input type="hidden" name="groupId" value="<%=g.getId()%>">
          <div class="grid grid-2">
            <div>
              <label class="label">ID User</label>
              <input class="input" name="userId" placeholder="U4" required>
            </div>
            <div>
              <label class="label">Nama User</label>
              <input class="input" name="userName" placeholder="Nama Anggota" required>
            </div>
          </div>
          <div class="actions">
            <button class="btn" type="submit">Tambah</button>
          </div>
        </form>
      </div>

      <div class="card">
        <div class="hl">Anggota</div>
        <table class="table">
          <thead><tr><th>ID</th><th>Nama</th></tr></thead>
          <tbody>
          <% for (User u : g.getMembers()) { %>
            <tr><td><%=u.getId()%></td><td><%=u.getName()%></td></tr>
          <% } %>
          </tbody>
        </table>
```

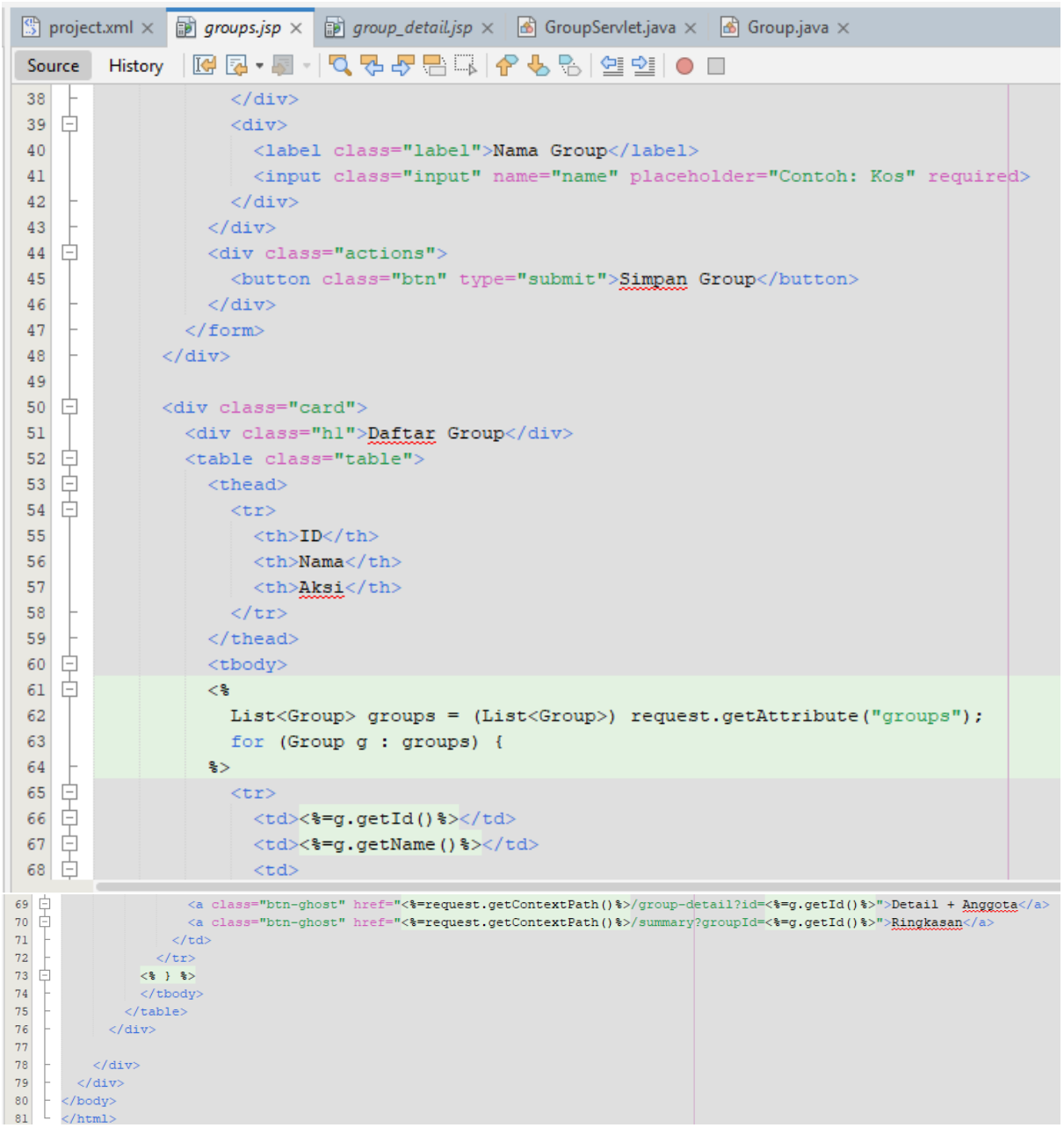
```
        </div>
        <% } %>

      </div>
    </div>
</body>
</html>
```

File group_detail.jsp dipakai untuk menampilkan detail sebuah group, termasuk daftar anggota yang tergabung di dalamnya. Object Group dikirimkan dari servlet melalui request attribute dan diambil kembali pada halaman JSP.

Halaman ini menyediakan form untuk menambahkan anggota baru ke dalam group dengan memasukkan ID dan nama user. Data anggota kemudian ditampilkan dalam bentuk tabel dengan melakukan iterasi terhadap daftar member yang dimiliki oleh group.
Implementasi ini menunjukkan relasi antara Group dan User, di mana sebuah group dapat memiliki banyak anggota yang dikelola secara dinamis melalui antarmuka web.

4. Groups.jsp



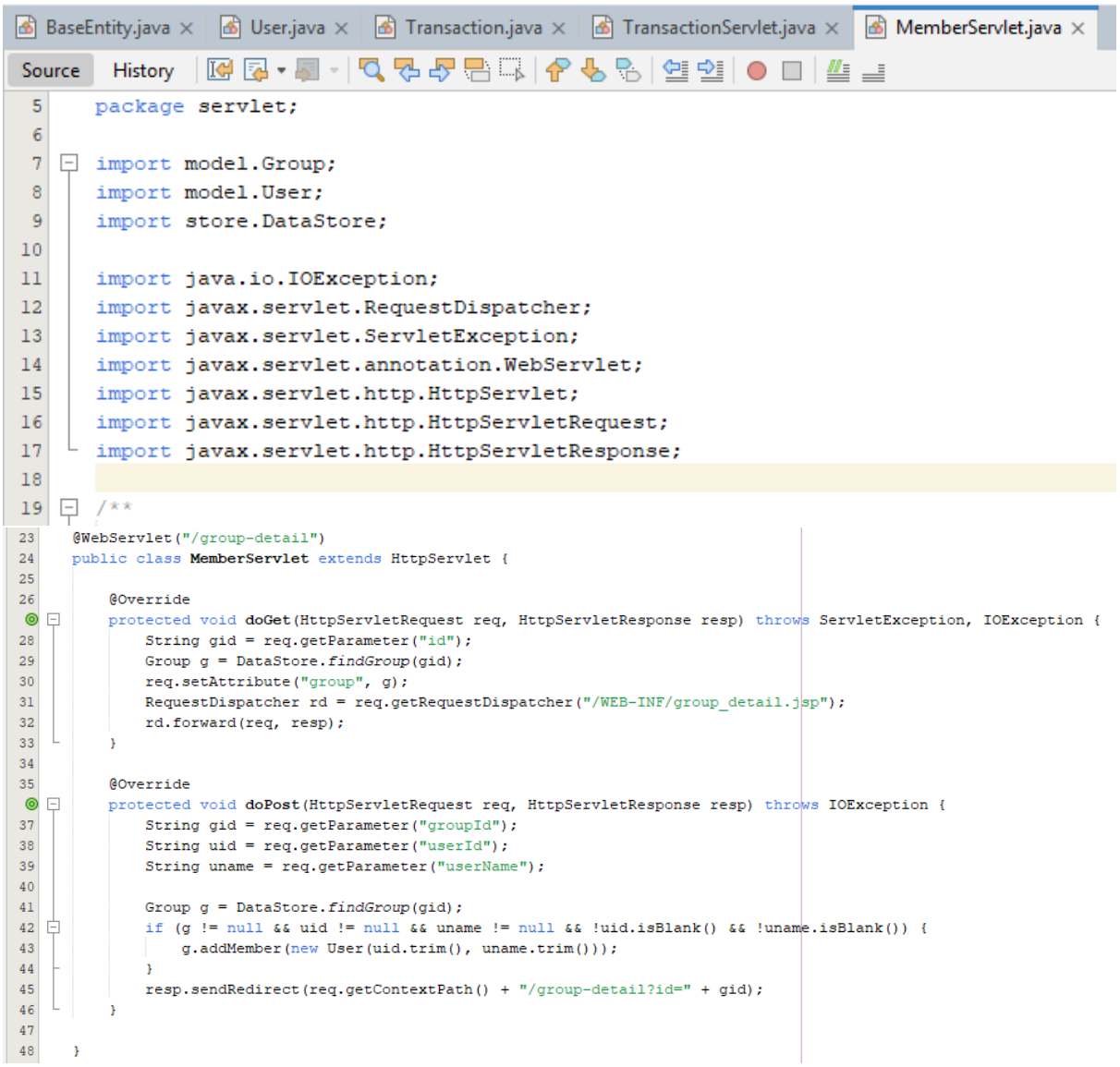


```
project.xml | groups.jsp | group_detail.jsp | GroupServlet.java | Group.java
Source  History
38              </div>
39              <div>
40                <label class="label">Nama Group</label>
41                <input class="input" name="name" placeholder="Contoh: Kos" required>
42              </div>
43            </div>
44            <div class="actions">
45              <button class="btn" type="submit">Simpan Group</button>
46            </div>
47          </form>
48        </div>
49
50        <div class="card">
51          <div class="hl">Daftar Group</div>
52          <table class="table">
53            <thead>
54              <tr>
55                <th>ID</th>
56                <th>Nama</th>
57                <th>Aksi</th>
58              </tr>
59            </thead>
60            <tbody>
61            <%
62              List<Group> groups = (List<Group>) request.getAttribute("groups");
63              for (Group g : groups) {
64            %>
65              <tr>
66                <td><%=g.getId()%></td>
67                <td><%=g.getName()%></td>
68                <td>
69                  <a class="btn-ghost" href="<%=request.getContextPath()%>/group-detail?id=<%=g.getId()%>">Detail + Anggota</a>
70                  <a class="btn-ghost" href="<%=request.getContextPath()%>/summary?groupId=<%=g.getId()%>">Ringkasan</a>
71                </td>
72              </tr>
73            <% } %>
74            </tbody>
75          </table>
76        </div>
77
78      </div>
79    </div>
80  </body>
81  </html>
```

File groups.jsp adalah codingan buat halaman tampilan (view) yang digunakan untuk mengelola dan menampilkan daftar group. Halaman ini menerima data berupa list group yang dikirimkan oleh GroupServlet melalui request attribute.

Pada halaman ini terdapat form untuk menambahkan group baru yang akan diproses oleh servlet melalui method POST. Selain itu, data group ditampilkan dalam bentuk tabel menggunakan perulangan JSP untuk menampilkan ID dan nama group.
Halaman ini menunjukkan penerapan konsep MVC, di mana JSP hanya bertugas untuk menampilkan data tanpa mengelola logika bisnis.

5. memberServlet.java



File MemberServlet.java itu servlet yang berfungsi untuk menangani proses penambahan anggota (User) ke dalam sebuah group. Servlet ini berperan sebagai controller yang menghubungkan input dari user pada halaman detail group dengan model Group dan User.
Pada proses yang ditangani oleh servlet ini, data berupa ID group serta informasi user (ID dan nama) diterima melalui request dari halaman group_detail.jsp. Selanjutnya, servlet akan mencari object Group yang sesuai dan menambahkan object User baru ke dalam daftar anggota group dengan memanfaatkan method addMember() yang tersedia pada class Group.
Dengan adanya servlet ini, pengelolaan anggota group dapat dilakukan secara terstruktur dan terintegrasi, sehingga setiap group dapat memiliki daftar anggota yang dikelola secara dinamis sesuai dengan konsep pemrograman berorientasi objek dan arsitektur MVC.

5.Alishadena Chandrani Noor Riva Hutomo

Peran:
Pembuatan DebtCalculator, DashboardServlet, dan SummaryServlet, integrasi perhitungan akhir sistem, serta dokumentasi dan pengujian.

Tanggung Jawab:
1. Mengimplementasikan class DebtCalculator untuk menghitung saldo bersih (utang–piutang) setiap user
2. Mengintegrasikan hasil perhitungan saldo ke dalam tampilan aplikasi melalui SummaryServlet
3. Mengembangkan DashboardServlet sebagai halaman ringkasan awal aplikasi
4. Mengelola alur data antara model, service, dan servlet
5. Menyusun dokumentasi teknis serta melakukan pengujian fungsional sistem

Hasil yang Sudah Dikerjakan:
• Class DebtCalculator berhasil dibuat dengan:
- Logika perhitungan saldo bersih (utang–piutang)
- Integrasi hasil pembagian transaksi dari *split strategy*

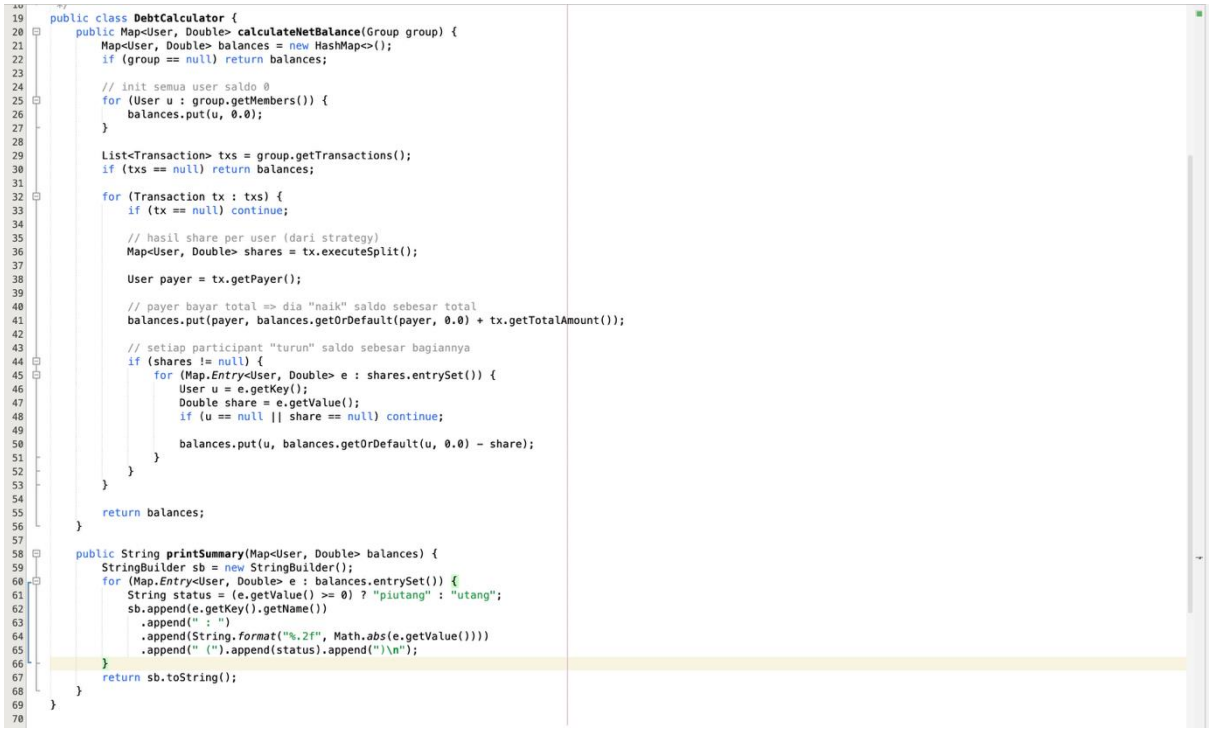
• DebtCalculator dapat:
- Menghitung saldo akhir setiap anggota grup
- Menentukan status utang atau piutang pengguna
- Menghasilkan ringkasan hasil perhitungan'

• SummaryServlet berhasil dibuat untuk:
- Mengambil data grup aktif
- Menjalankan perhitungan saldo menggunakan DebtCalculator
- Mengirim hasil perhitungan ke halaman summary

• DashboardServlet berhasil dibuat untuk:
- Menampilkan informasi ringkasan awal aplikasi
- Menampilkan jumlah grup yang tersedia

Screenshot yang disertakan:

1. DebtCalculator.java



```java
public class DebtCalculator {
    public Map<User, Double> calculateNetBalance(Group group) {
        Map<User, Double> balances = new HashMap<>();
        if (group == null) return balances;

        // init semua user saldo 0
        for (User u : group.getMembers()) {
            balances.put(u, 0.0);
        }

        List<Transaction> txs = group.getTransactions();
        if (txs == null) return balances;

        for (Transaction tx : txs) {
            if (tx == null) continue;

            // hasil share per user (dari strategy)
            Map<User, Double> shares = tx.executeSplit();

            User payer = tx.getPayer();

            // payer bayar total => dia "naik" saldo sebesar total
            balances.put(payer, balances.getOrDefault(payer, 0.0) + tx.getTotalAmount());

            // setiap participant "turun" saldo sebesar bagiannya
            if (shares != null) {
                for (Map.Entry<User, Double> e : shares.entrySet()) {
                    User u = e.getKey();
                    Double share = e.getValue();
                    if (u == null || share == null) continue;

                    balances.put(u, balances.getOrDefault(u, 0.0) - share);
                }
            }
        }

        return balances;
    }

    public String printSummary(Map<User, Double> balances) {
        StringBuilder sb = new StringBuilder();
        for (Map.Entry<User, Double> e : balances.entrySet()) {
            String status = (e.getValue() >= 0) ? "piutang" : "utang";
            sb.append(e.getKey().getName())
              .append(" : ")
              .append(String.format("%.2f", Math.abs(e.getValue())))
              .append(" (").append(status).append(")\n");
        }
        return sb.toString();
    }
}
```

Program DebtCalculator merupakan class yang bertanggung jawab untuk menghitung saldo bersih setiap pengguna dalam sebuah grup dengan mengolah seluruh data transaksi yang ada. Proses perhitungan dimulai dengan inisialisasi struktur data Map<User, Double> untuk menyimpan saldo akhir setiap user, di mana seluruh anggota grup diberikan nilai awal nol guna memastikan perhitungan dilakukan secara konsisten dan terisolasi dari transaksi sebelumnya.

Selanjutnya, sistem melakukan iterasi terhadap setiap transaksi di dalam grup. Pada setiap transaksi, strategi pembagian tagihan dijalankan untuk menghasilkan nilai tanggungan masing-masing peserta. Nilai total transaksi ditambahkan ke saldo pengguna yang bertindak sebagai pembayar, sedangkan saldo setiap peserta dikurangi sesuai nilai pembagian yang diperoleh dari strategi pembagian yang digunakan, baik *EvenSplitStrategy* maupun *UnevenSplitStrategy*.

Setelah seluruh transaksi selesai diproses, hasil perhitungan disimpan dalam Map<User, Double> yang merepresentasikan posisi keuangan akhir setiap user. Nilai saldo positif menunjukkan piutang, sedangkan nilai negatif menunjukkan utang. Untuk mempermudah interpretasi hasil, class ini juga menyediakan method printSummary() yang menampilkan ringkasan saldo beserta status utang atau piutang dalam format teks yang terstruktur.

2. DashboardServlet.java

```java
package servlet;

import store.DataStore;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import java.io.IOException;
import javax.servlet.RequestDispatcher;

@WebServlet("/dashboard")
public class DashboardServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        req.setAttribute("groupsCount", DataStore.getGroups().size());
        req.setAttribute("active", "dashboard");
        req.getRequestDispatcher("/WEB-INF/dashboard.jsp").forward(req, resp);
    }

}
```

DashboardServlet berfungsi sebagai pengendali halaman dashboard pada aplikasi SplitBill. Servlet ini menangani permintaan HTTP dengan mengambil data ringkasan dari sistem,

khususnya jumlah grup yang tersimpan, kemudian meneruskannya ke halaman tampilan dashboard. Informasi ini digunakan untuk memberikan gambaran awal kepada pengguna mengenai kondisi data yang ada di dalam aplikasi.

Pada method doGet, DashboardServlet mengambil jumlah grup dari DataStore dan menyimpannya sebagai atribut request agar dapat diakses oleh halaman dashboard.jsp. Selain itu, servlet ini juga menetapkan status halaman aktif untuk kebutuhan navigasi antarmuka. Setelah seluruh data disiapkan, permintaan diteruskan ke halaman JSP menggunakan mekanisme *request forwarding*, sehingga dashboard dapat ditampilkan secara dinamis berdasarkan data aktual sistem.

3.SummaryServlet.java

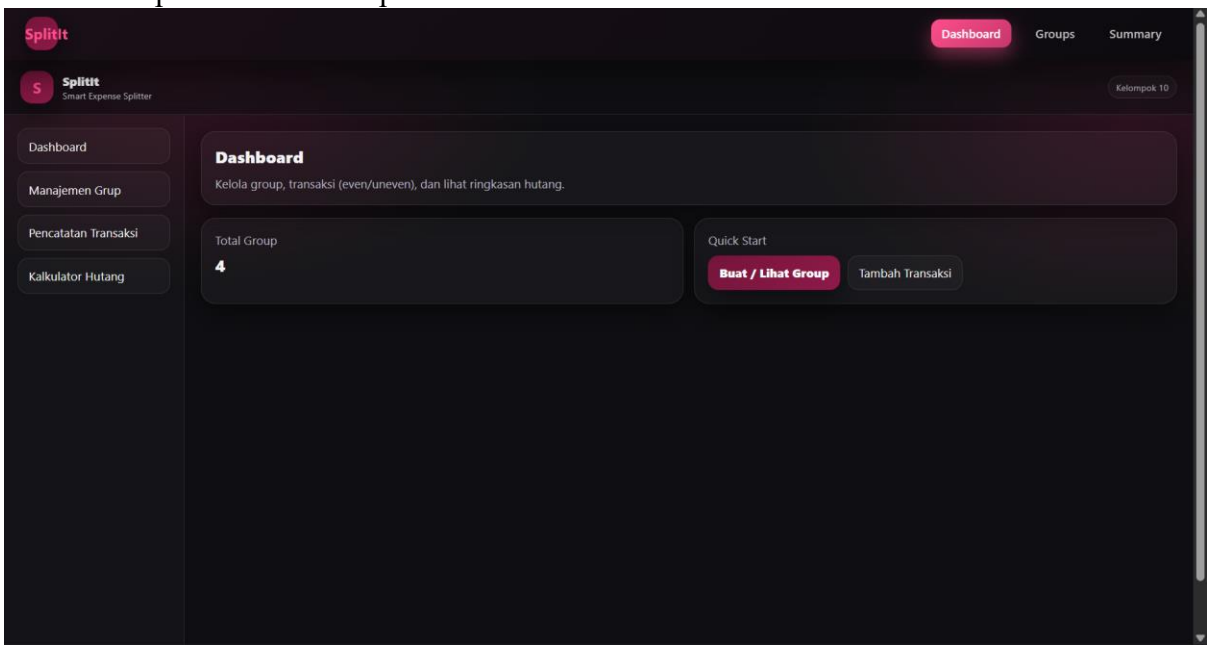
```java
package servlet;

import model.Group;
import model.User;
import service.DebtCalculator;
import store.DataStore;

import java.io.IOException;
import java.util.Map;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/summary")
public class SummaryServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String gid = req.getParameter("groupId");

        if (gid == null || gid.isBlank()) {
            Object last = req.getSession().getAttribute("activeGroupId");
            if (last != null) {
                gid = last.toString();
            }
        }

        Group g = null;
        if (gid != null && !gid.isBlank()) {
            g = DataStore.findGroup(gid);
        }
        if (g == null && !DataStore.getGroups().isEmpty()) {
            g = DataStore.getGroups().get(0);
            gid = g.getId();
        }

        req.setAttribute("group", g);

        if (g != null) {
            DebtCalculator calc = new DebtCalculator();
            Map<User, Double> balances = calc.calculateNetBalance(g);
            req.setAttribute("balances", balances);

            req.getSession().setAttribute("activeGroupId", g.getId());
        }

        req.setAttribute("active", "summary");
        req.getRequestDispatcher("/WEB-INF/summary.jsp").forward(req, resp);
    }

}
```

SummaryServlet berfungsi untuk menampilkan ringkasan hasil perhitungan utang dan piutang pada sebuah grup dalam aplikasi SplitBill. Servlet ini bertanggung jawab mengambil data grup yang aktif, menjalankan proses perhitungan saldo akhir menggunakan DebtCalculator, serta meneruskan hasil perhitungan tersebut ke halaman tampilan ringkasan.

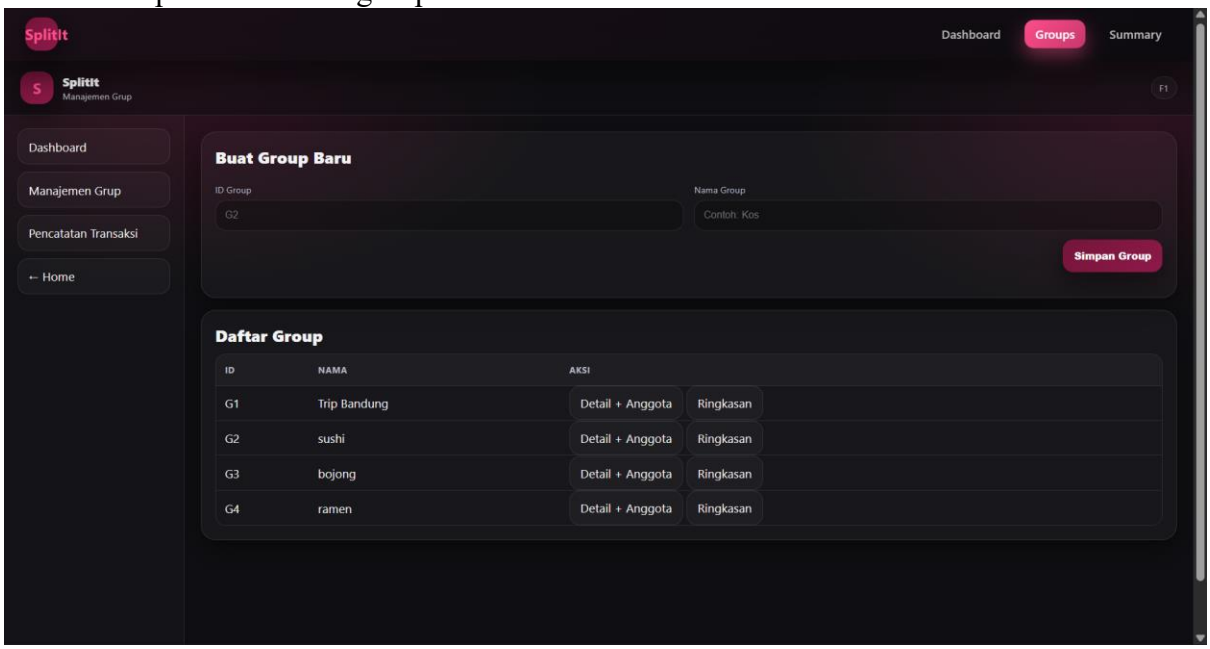
Pada method doGet, SummaryServlet mengambil parameter groupId dari request atau dari session jika parameter tidak tersedia. Servlet kemudian mencari data grup yang sesuai dari DataStore. Jika grup ditemukan, sistem menjalankan perhitungan saldo bersih setiap user melalui DebtCalculator dan menyimpan hasilnya sebagai atribut request. Selain itu, servlet juga menyimpan groupId aktif ke dalam session untuk menjaga konteks grup yang sedang digunakan. Setelah seluruh data siap, permintaan diteruskan ke halaman summary.jsp menggunakan *RequestDispatcher* untuk ditampilkan kepada pengguna.

Screenshot hasil codingan sementara :

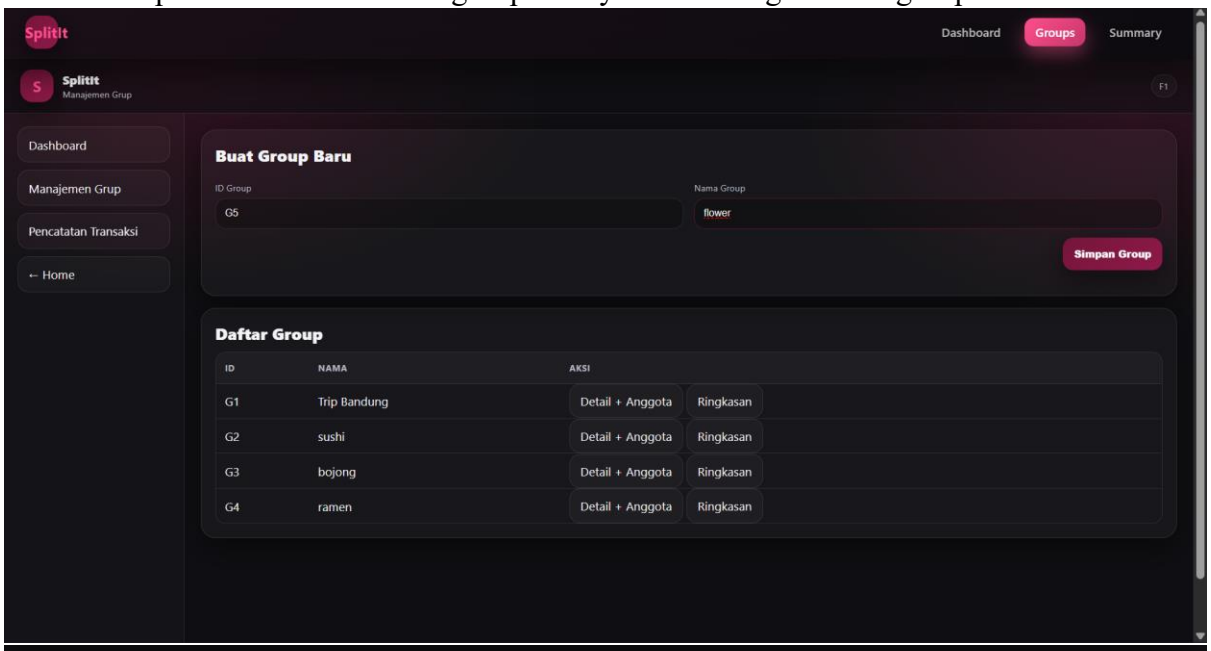
Berikut tampilan dahsboard splitIt

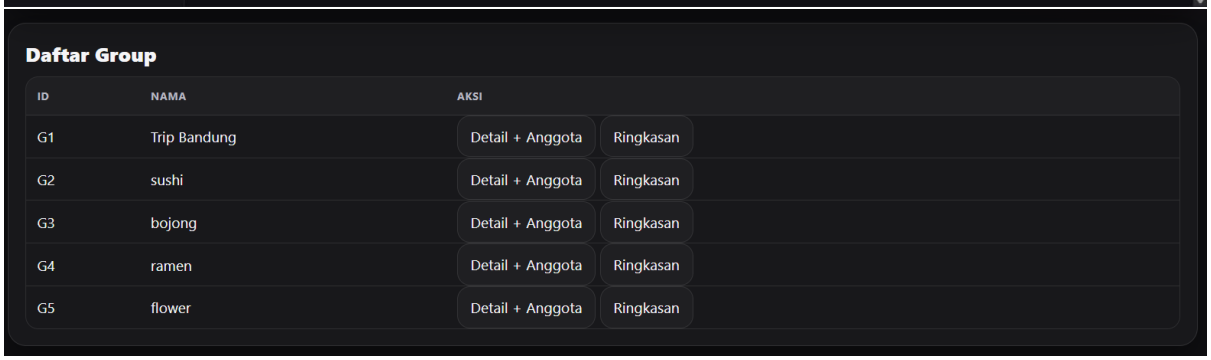


Berikut tampilan buat/lihat group

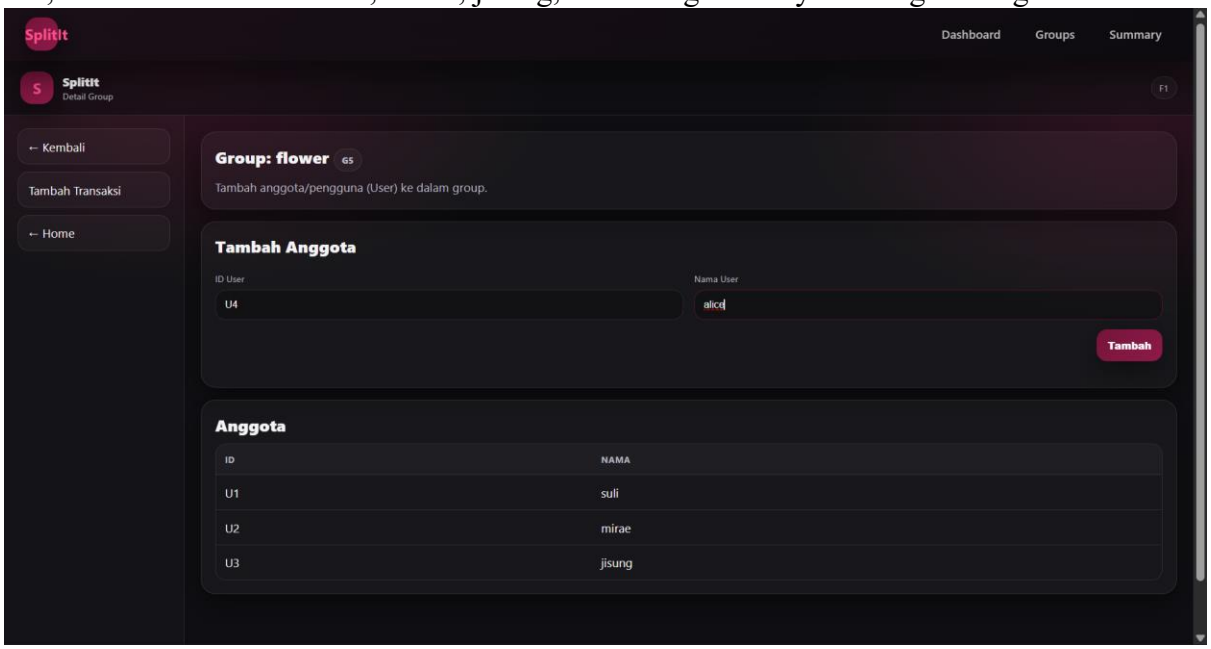


Berikut tampilan setelah membuat group baru yaitu G5 dengan nama group flower

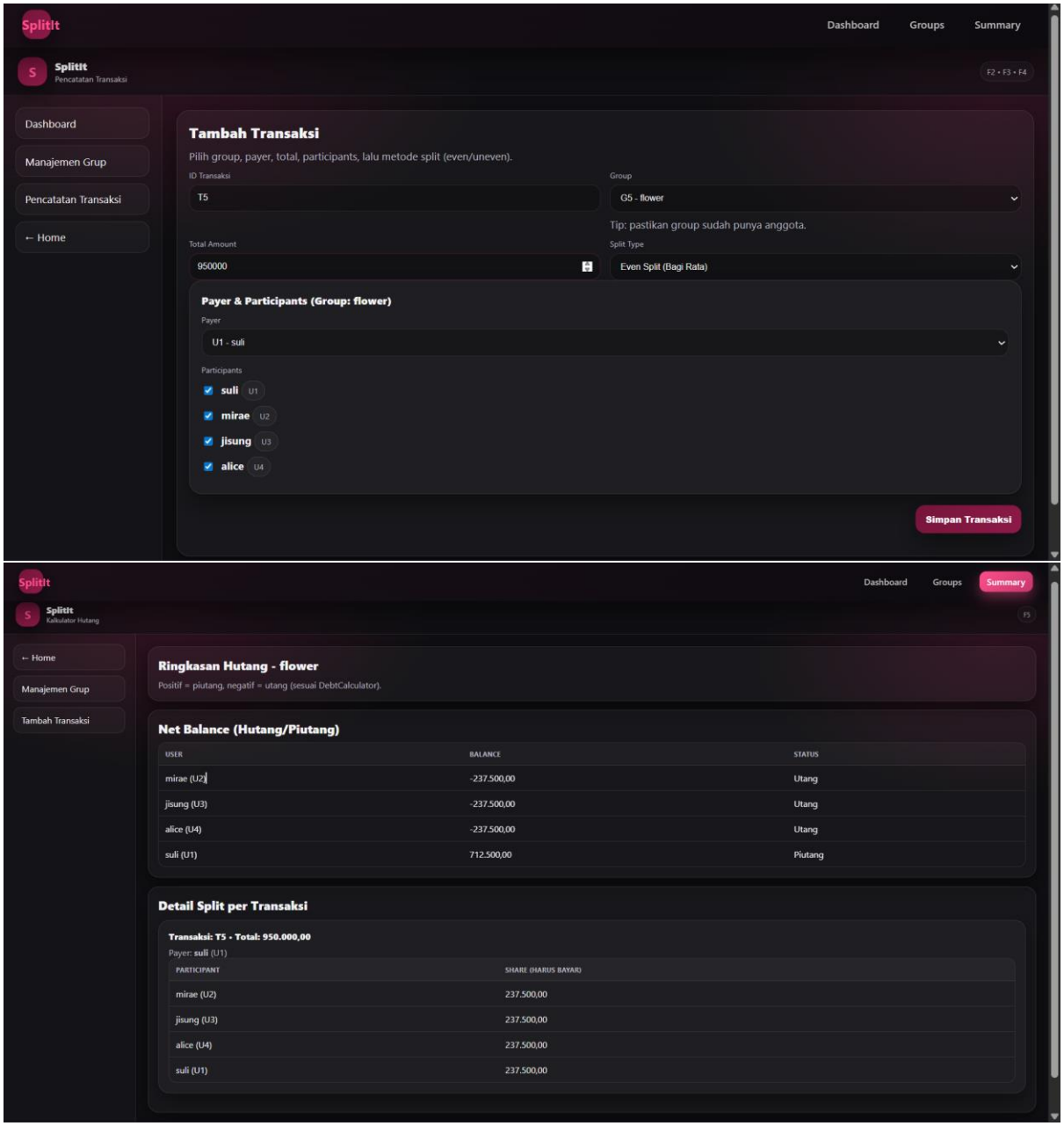




Berikut tampilan detail + anggota, tempat dimana bisa menambahkan anggota yang akan split bill, di sini ditambahkan suli, mirae, jisung, alice dengan ID nya masing-masing

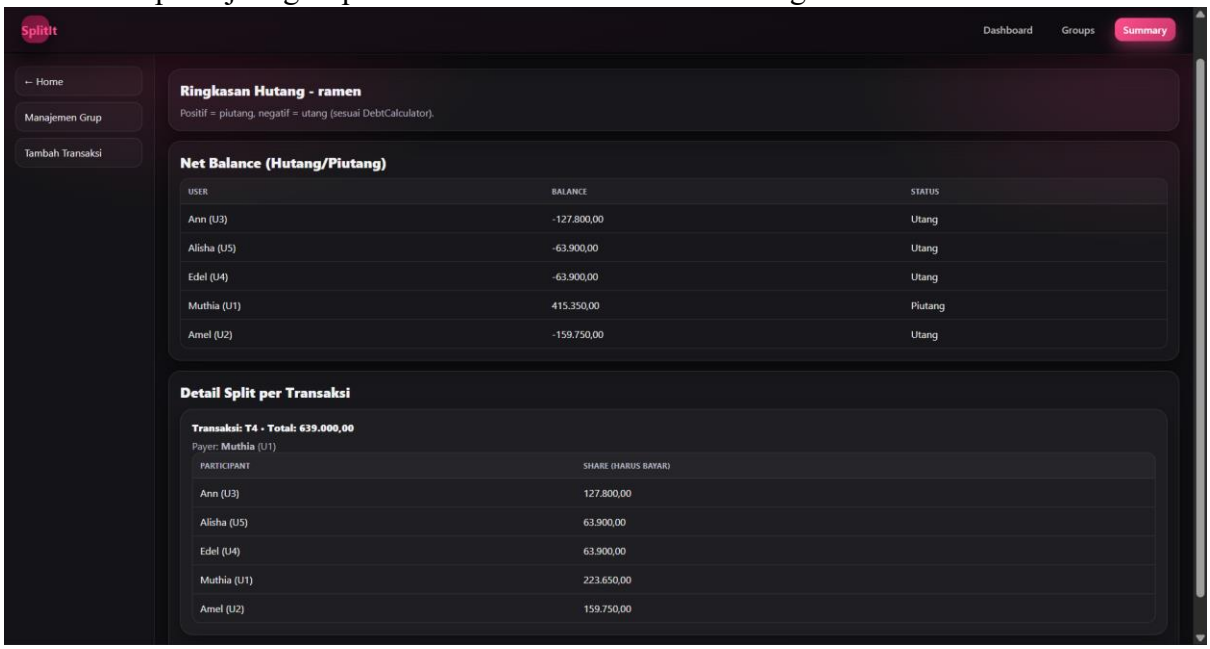


Setelah group dan anggota nya dibuat, selanjutnya mengisi transaksi untuk di split bill



ini adalah tampilan jika group memilih split even/bagi rata per orang

Berikut tampilan jika group memilih untuk uneven/tidak dibagi rata



NOTES :

- Codingan masih akan terus di perbarui (belum fix)
- Untuk bagian split uneven masih akan diperbarui karena masih ada beberapa fitur yang belum di implementasikan
- Dan masih ada beberapa fitur dan tampilan UI untuk splitIt yang akan diperbarui lagi