

Aplikasi Pengelolaan Keuangan Pribadi

Presented By:
Fizri Rosdiansyah
Iftah Maulana
Maula Afiif

financeApp.java

financeApp sebagai main class memiliki fungsi utama Menjalankan aplikasi Spring Boot dan memulai tampilan GUI menggunakan financeView.

Alur :

- Mengatur aplikasi agar bisa menjalankan GUI meskipun berbasis server.
- Memulai Spring Boot, yang otomatis memuat semua komponen aplikasi.
- Membuka tampilan GUI utama (financeView) dan menghubungkannya dengan controller (financeController).

Logika :

Aplikasi dijalankan dari sini. Spring Boot menyiapkan semua komponen di belakang layar, lalu GUI dibuka untuk user.

```
package com.mahasiswa;

import com.mahasiswa.controller.financeController;
import com.mahasiswa.service.financeService;
import com.mahasiswa.view.financeView;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class financeApp implements ApplicationRunner {
    @Autowired
    private financeService financeService;

    public static void main(String[] args) {
        System.setProperty("java.awt.headless", "false");

        ApplicationContext context = SpringApplication.run(financeApp.class, args);

        financeController controller = context.getBean(financeController.class);
        financeView financeView = new financeView(controller);
        financeView.setVisible(true);
    }

    @Override
    public void run(ApplicationArguments args) throws Exception {
    }
}
```

financeController.java

financeController berfungsi sebagai jembatan yang menghubungkan antara service dan view

Alur :

- **saveTransaction:** Ketika user menekan tombol "Save", data dikirim ke metode `saveTransaction()`, yang lalu diteruskan ke `financeService` untuk disimpan di database.
- **deleteTransaction:** Ketika user menekan tombol "Delete", ID transaksi dikirim ke metode `deleteTransaction()`, yang lalu diteruskan untuk dihapus.
- **getAllTransactions:** Mengambil data dari database melalui metode `getAllTransactions()` dan dalam bentuk `List<modelFinance>`.

Logika :

Controller ini bertugas mengatur data yang keluar masuk antara tampilan (view) dan database.

```
package com.mahasiswa.controller;

import com.mahasiswa.model.modelFinance;
import com.mahasiswa.service.financeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import java.util.List;

@Controller
public class financeController {
    @Autowired
    private financeService transactionService;

    public void saveTransaction(modelFinance transaction) {
        transactionService.saveTransaction(transaction);
    }

    public void deleteTransaction(int id) {
        transactionService.deleteTransaction(id);
    }

    public List<modelFinance> getAllTransactions() {
        return transactionService.getAllTransactions();
    }
}
```

modelFinance.java

modelFinance ini bertujuan menggambarkan struktur data transaksi

Alur :

- Setiap transaksi disimpan dengan informasi: id, date, dan amount.
 - Spring JPA akan otomatis menyimpan atau mengambil data dari database sesuai struktur ini.

Logika :

- **Annotation JPA:**
 - **@Entity** menandai kelas sebagai entitas JPA.
 - **@Table** menentukan nama tabel (**finance**).
 - **@Id** dan **@GeneratedValue** menentukan ID sebagai primary key dengan auto-increment.

- **Field Properties:**

- id: Integer, primary key

- date: String, tanggal transaksi.

- amount: Integer, jumlah transaksi.

- **Getter & Setter:** Untuk manipulasi data.

- model ini digunakan untuk memetakan data dari database ke objek Java (ORM).

modelTableFinance.java

modelTableFinance ini bertujuan Mengatur tampilan tabel di GUI.

Alur :

- Data transaksi disiapkan dalam format yang bisa ditampilkan di tabel.
- Metode seperti `getValueAt()` memastikan kolom di tabel menampilkan data yang benar.
- Ketika data berubah, tabel diperbarui otomatis dengan `fireTableDataChanged()`.

Logika :

- Konstruktor: Menerima `List<modelFinance>` dan menyimpannya dalam `financeList`.
- `getValueAt`: Mengembalikan nilai berdasarkan indeks baris dan kolom.
- `fireTableDataChanged`: Memperbarui tampilan tabel jika ada perubahan data.
- Menyediakan struktur data untuk menampilkan transaksi di tabel Swing.
- Model ini menggambarkan bagaimana data terlihat di tabel GUI dan bagaimana data itu terhubung ke database.

```
package com.mahasiswa.model;

import javax.swing.table.AbstractTableModel;
import java.util.List;

public class modelTableFinance extends AbstractTableModel{
    private List<modelFinance> financeList;
    private final String[] columnNames = {"ID", "Tanggal", "Amount"};

    public modelTableFinance(List<modelFinance> financeList) {
        this.financeList = financeList;
    }

    @Override
    public int getRowCount() {
        return financeList.size();
    }

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        modelFinance entry = financeList.get(rowIndex);
        switch (columnIndex) {
            case 0:
                return entry.getId();
            case 1:
                return entry.getDate();
            case 2:
                return entry.getAmount();
        }
        return null;
    }

    @Override
    public String getColumnName(int columnIndex) {
        return columnNames[columnIndex];
    }

    @Override
    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return false; // Semua sel tidak dapat diedit langsung dari tabel
    }

    public void setFinanceList(List<modelFinance> financeList) {
        this.financeList = financeList;
        fireTableDataChanged(); // Memberi tahu tabel untuk memperbarui
    }
}
```

financeRepository.java

financeRepository ini bertujuan Mengelola database

Alur :

- Mengelola operasi CRUD untuk entitas modelFinance.
 - Secara otomatis menyediakan implementasi save(), deleteById(), dan findAll().
 - save() untuk menyimpan transaksi.
 - findAll() untuk mendapatkan semua data transaksi.
 - deleteById() untuk menghapus transaksi berdasarkan ID.

Logika :

- **JpaRepository<modelFinance, Integer>:**
 - Memiliki metode bawaan untuk operasi database seperti save, findAll dan deleteById.
 - Spring Boot otomatis membuat kode untuk mengakses database, jadi kita tidak perlu menulis query SQL secara manual.

The screenshot shows a Java code editor with the following code:

```
1 package com.mahasiswa.repository;
2
3 import com.mahasiswa.model.modelFinance;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 public interface financeRepository extends JpaRepository<modelFinance, Integer> {
8
9 }
10
```

The code defines a repository interface named `financeRepository` that extends `JpaRepository<modelFinance, Integer>`. The `modelFinance` class is imported from `com.mahasiswa.model.modelFinance`. The `Repository` stereotype is imported from `org.springframework.stereotype.Repository`. The code editor has a toolbar at the top with various icons for file operations like new, open, save, and search.

financeService.java

financeService ini bertujuan Mengatur logika bisnis aplikasi.

Alur :

- **saveTransaction:** Meneruskan data transaksi ke financeRepository.save().
- **deleteTransaction:** Meneruskan ID ke financeRepository.deleteById().
- **getAllTransactions:** Memanggil financeRepository.findAll().

Logika :

- Service ini bertugas memastikan data yang diproses benar, misalnya memeriksa format atau validitas data sebelum diteruskan ke database.

```
1 package com.mahasiswa.service;
2
3 import com.mahasiswa.model.modelFinance;
4 import com.mahasiswa.repository.financeRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7 import java.util.List;
8
9 @Service
10 public class financeService {
11     @Autowired
12     private financeRepository repository;
13
14     public void saveTransaction(modelFinance transaction) {
15         repository.save(transaction);
16     }
17
18     public void deleteTransaction(int id) {
19         repository.deleteById(id);
20     }
21
22     public List<modelFinance> getAllTransactions() {
23         return repository.findAll();
24     }
25
26 }
```

design financeView.java

Komponen:

- Form input untuk tanggal dan amount.
- Tombol "Save" untuk menyimpan transaksi.
- Tombol "Delete" untuk menghapus transaksi berdasarkan ID.
- Tabel (JTable) untuk menampilkan semua data transaksi.

Logika Button :

- Save:
 - Mengambil data dari jTextField1 dan jTextField2.
 - Membuat objek modelFinance.
 - Memanggil saveTransaction() dari controller.
 - Memperbarui tabel dengan loadFinanceTable().
- Delete:
 - Membuka dialog input untuk memasukkan ID transaksi yang ingin dihapus.
 - Memanggil deleteTransaction() dari controller.
 - Memperbarui tabel setelah penghapusan.



financeView.java

financeView ini bertujuan sebagai Antarmuka pengguna (UI) untuk mengelola transaksi.

Alur :

- Menampilkan data:
- Ketika aplikasi dibuka, tabel memuat semua transaksi dari database.
- Menambah transaksi:
- User mengisi tanggal dan jumlah, lalu menekan tombol "Save".
- Data dikirim ke controller.saveTransaction(), lalu tabel diperbarui.
- Menghapus transaksi:
- User menekan tombol "Delete" dan memasukkan ID.
- ID dikirim ke controller.deleteTransaction(), lalu tabel diperbarui.

Logika :

- GUI ini memudahkan user untuk menambah, melihat, atau menghapus data transaksi.
- Event Handling:
- **jButton1ActionPerformed**: Menyimpan data baru dari input pengguna.
- **jButton2ActionPerformed**: Menghapus data berdasarkan ID yang dimasukkan pengguna melalui dialog.

```
1 package com.mahasiswa.view;
2
3 import com.mahasiswa.controller.financeController;
4 import com.mahasiswa.model.modelFinance;
5 import com.mahasiswa.model.modelTableFinance;
6 import javax.swing.*;
7 import java.awt.*;
8 import java.util.List;
9
10 public class financeView extends javax.swing.JFrame {
11     private financeController controller;
12
13     public financeView(financeController controller) {
14         this.controller = controller;
15         initComponents();
16         loadFinanceTable();
17     }
18
19     private financeView() {
20         throw new UnsupportedOperationException("Not supported yet.");
21     }
22
23     private void loadFinanceTable() {
24         List<modelFinance> financeList = controller.getAllTransactions();
25         modelTableFinance tableModel = new modelTableFinance(financeList);
26         dataTable.setModel(tableModel);
27     }
28
29     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
30         String date = jTextField1.getText();
31         int amount = Integer.parseInt(jTextField2.getText());
32         modelFinance finance = new modelFinance(0, date, amount);
33         System.out.println(finance.getDate());
34         System.out.println(finance.getAmount());
35
36         controller.saveTransaction(finance);
37         loadFinanceTable();
38
39         jTextField1.setText("");
40         jTextField2.setText("");
41     }
42
43     private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
44         JTextField idField = new JTextField(10);
45
46         JPanel panel = new JPanel();
47         panel.add(new JLabel("Masukkan ID yang ingin dihapus : "));
48         panel.add(idField);
49
50         int result = JOptionPane.showConfirmDialog(null, panel, "Hapus Mahasiswa", JOptionPane.OK_CANCEL_OPTION, JOptionPane.PLAIN_MESSAGE);
51
52         if(result == JOptionPane.OK_OPTION){
53             try{
54                 int id = Integer.parseInt(idField.getText());
55                 controller.deleteTransaction(id);
56                 JOptionPane.showMessageDialog(null, "Data berhasil dihapus", "Success", JOptionPane.INFORMATION_MESSAGE);
57                 loadFinanceTable();
58             }catch (NumberFormatException e){
59                 JOptionPane.showMessageDialog(null, "ID Harus Berupa Angka!", "Error", JOptionPane.ERROR_MESSAGE);
60             }
61         }
62     }
63 }
```

pom.xml

Logika :

- Parent
- Parent di sini adalah Spring Boot, yang menyediakan konfigurasi dasar untuk aplikasi Spring Boot.
- **version="3.3.3"**: Menggunakan versi Spring Boot 3.3.3.
- Dependencies
- **spring-boot-starter-data-jpa**: Mendukung akses database menggunakan JPA dan Hibernate.
- **mysql-connector-java**: Driver JDBC untuk koneksi ke database MySQL.
- **spring-boot-starter-web**: Mendukung pengembangan aplikasi berbasis web (tidak digunakan langsung di aplikasi ini, tapi bisa untuk API di masa depan).
- **spring-boot-starter-test**: Menyediakan pustaka untuk pengujian (seperti JUnit dan Mockito).
- Build
- **spring-boot-maven-plugin**: Plugin yang digunakan untuk membangun aplikasi Spring Boot, termasuk menjalankan, mengemas, dan mem-build proyek.

Tujuan :

Dengan konfigurasi ini, Maven akan membangun proyek Spring Boot yang mendukung JPA untuk mengelola data ke MySQL, termasuk kemampuan untuk mengembangkan API web di masa depan.

```
14 <parent>
15   <groupId>org.springframework.boot</groupId>
16   <artifactId>spring-boot-starter-parent</artifactId>
17   <version>3.3.3</version>
18   <relativePath/>
19 </parent>
20
21 <dependencies>
22   <!-- Hibernate + Spring Data JPA -->
23   <dependency>
24     <groupId>org.springframework.boot</groupId>
25     <artifactId>spring-boot-starter-data-jpa</artifactId>
26   </dependency>
27
28   <!-- MySQL Connector -->
29   <dependency>
30     <groupId>mysql</groupId>
31     <artifactId>mysql-connector-java</artifactId>
32     <version>8.0.33</version>
33   </dependency>
34
35   <!-- Spring Boot Web dependency (for MVC if needed) -->
36   <dependency>
37     <groupId>org.springframework.boot</groupId>
38     <artifactId>spring-boot-starter-web</artifactId>
39   </dependency>
40
41
42   <!-- Testing dependencies -->
43   <dependency>
44     <groupId>org.springframework.boot</groupId>
45     <artifactId>spring-boot-starter-test</artifactId>
46     <scope>test</scope>
47   </dependency>
48 </dependencies>
49
50
51 <build>
52   <plugins>
53     <plugin>
54       <groupId>org.springframework.boot</groupId>
55       <artifactId>spring-boot-maven-plugin</artifactId>
56     </plugin>
57   </plugins>
58 </build>
59 </project>
```

application.properties

application.properties Mengatur

- Lokasi database.
- Username dan password untuk mengakses database.
- Bagaimana Spring Boot mengelola tabel (update berarti tabel akan diperbarui otomatis jika ada perubahan)

Alur :

- **spring.datasource.url:** Lokasi database MySQL ke port 3306
- **spring.datasource.username & password:** Kredensial database dengan username root dan tidak memiliki password
- **spring.jpa.hibernate.ddl-auto=update:** Hibernate akan otomatis memperbarui skema tabel.
- **spring.jpa.show-sql=true:** Menampilkan SQL yang dieksekusi di konsol.

Logika :

- Konfigurasi ini membuat aplikasi bisa terhubung dengan database MySQL tanpa perlu pengaturan manual tambahan.

```
#Konfigurasi MySQL Hibernate
spring.datasource.url=jdbc:mysql://localhost:3306/pertemuan6_db?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver

#Hibernate settings
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

**Thank you
very much!**