TUGAS BESAR REKAYASA PERANGKAT LUNAK

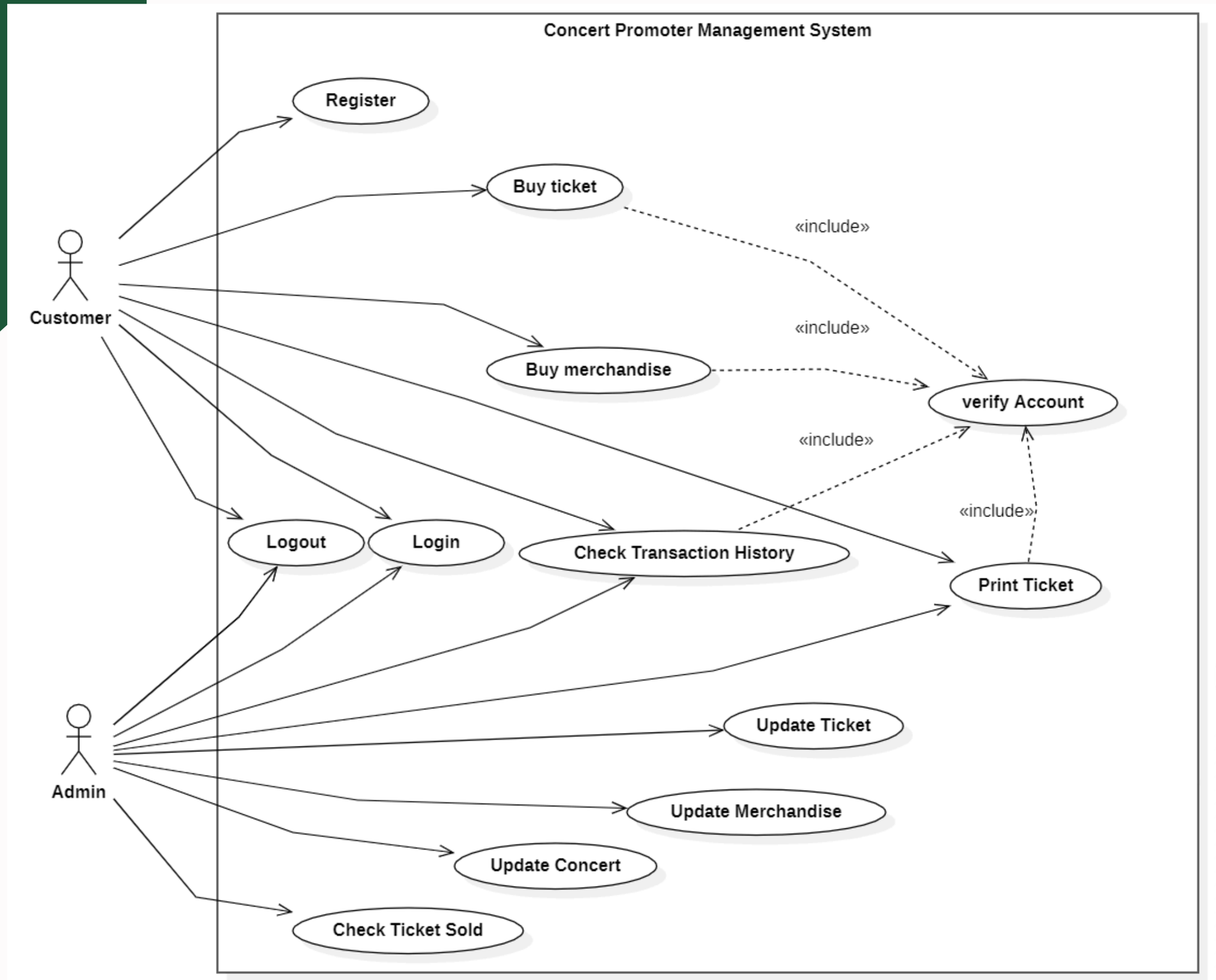
Kelompok:

- 1122006 Jason Enrico
- 1122014 Mikael Alexander
- 1122017 Marcel Andrean

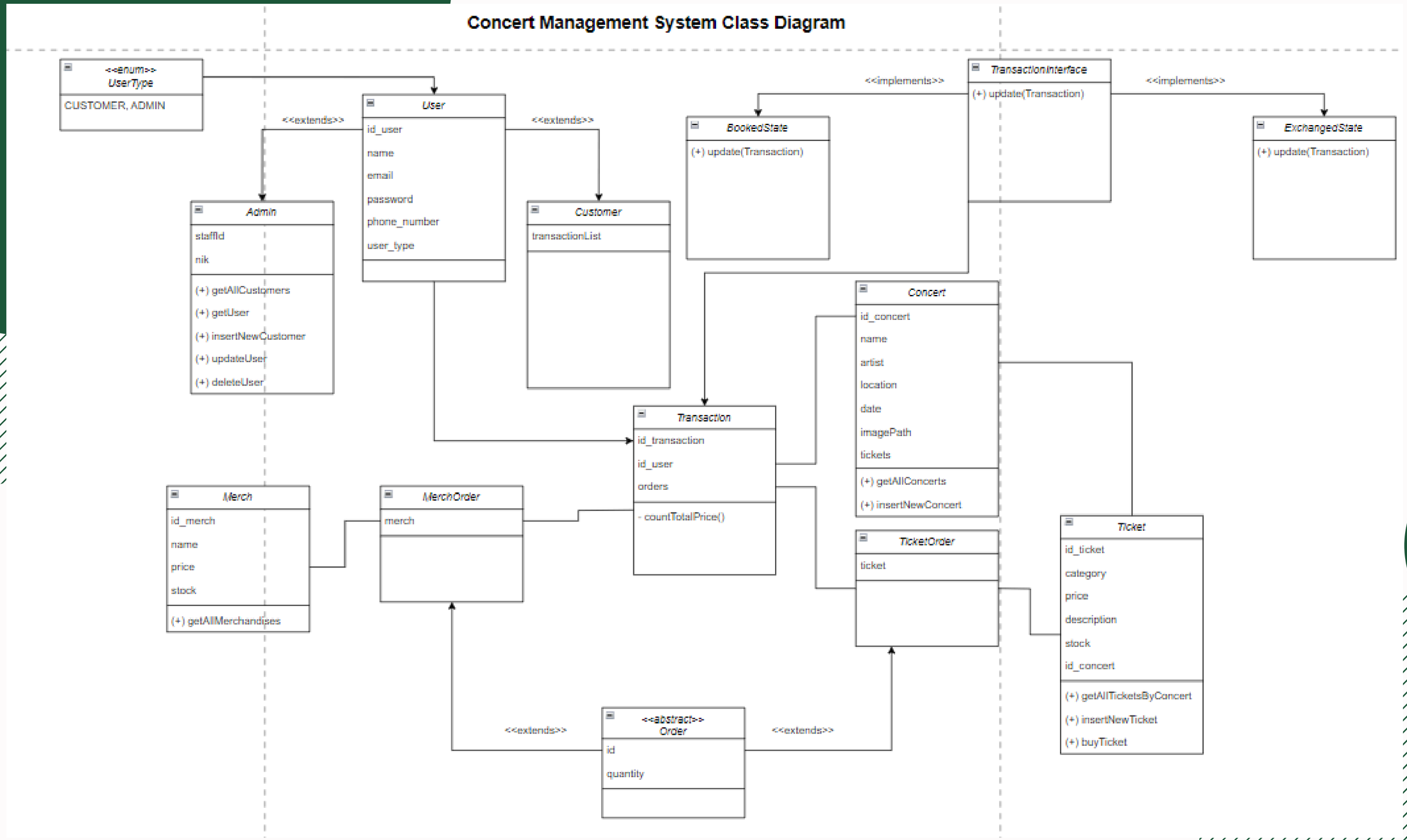
APLIKASI CONCERT

Aplikasi yang digunakan untuk pemesanan tiket
konser serta merchandise secara online

Use Case Diagram



Class Diagram



Overview for Customer

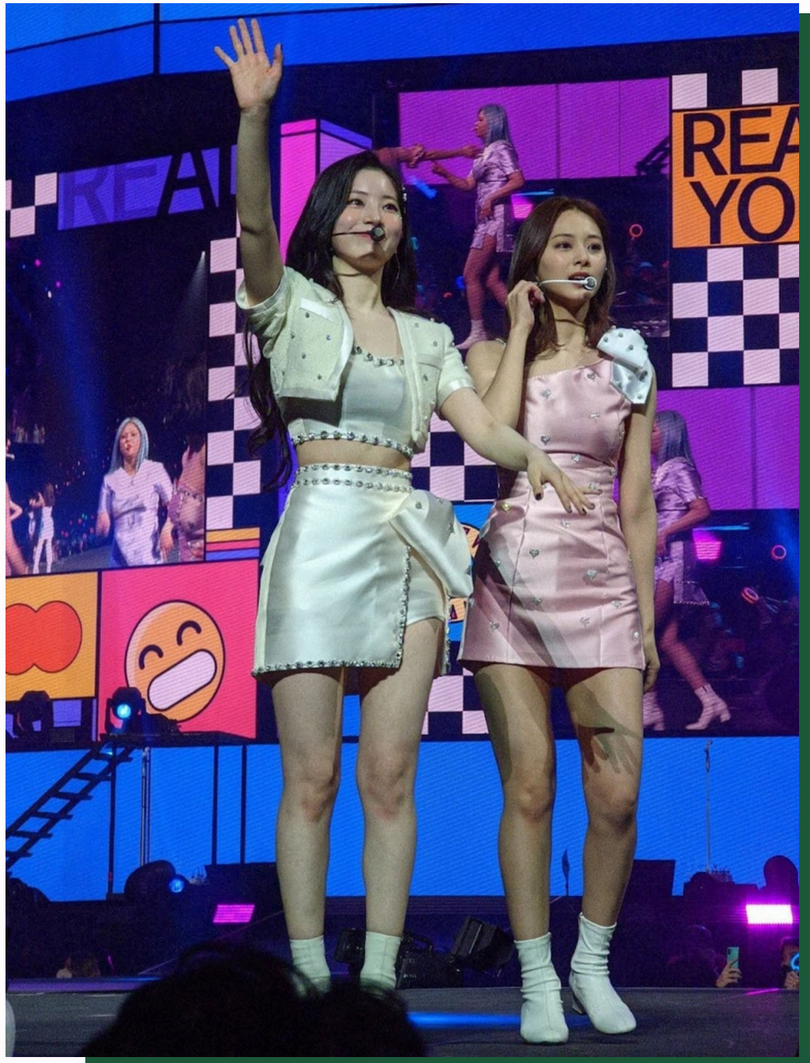
01 Main Screen

02 Registration / Login

03 Buy Ticket

04 Buy Merchandise

05 Transaction + Print Ticket



Overview for Admin

01 Main Screen

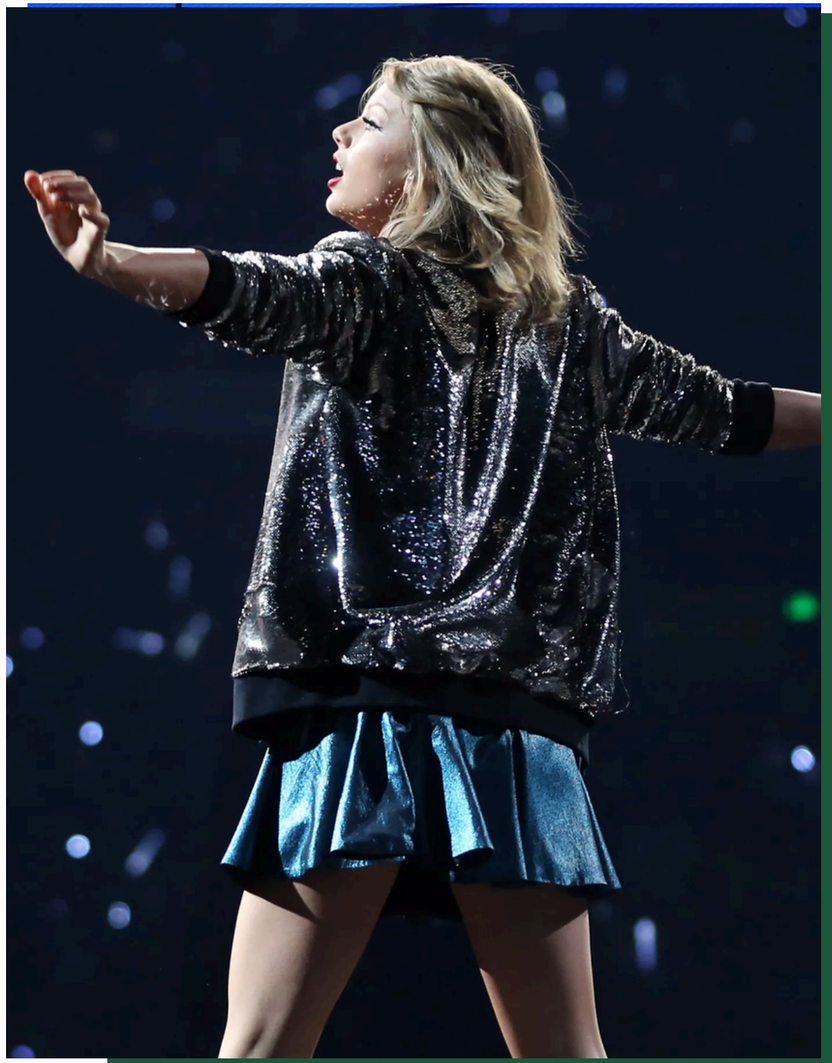
02 Login

03 Update Ticket

04 Update Concert

05 Update Merchandise

06 Check Ticket Sold




DESIGN PATTERN

Singleton

Singleton digunakan untuk menyimpan user session selama aplikasi berjalan, serta membedakan yang mana admin dan customer.

```java
public class UserSessionManager {

    private static UserSessionManager instance;
    private User user = null;

    public UserSessionManager() {}

    public static UserSessionManager getInstance() {
        if (instance == null) {
            instance = new UserSessionManager();
        }
        return instance;
    }

    public void reset() {
        this.user = null;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
```

```java
logout.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent ae) {
        UserSessionManager.getInstance().reset();
        frame.dispose();
        new WelcomeScreen();
    }
});
```

```java
@Override
public void actionPerformed(ActionEvent ae) {
    User user = new UserController().getUser(credentialField.getText(), passwordField.getText());

    UserSessionManager.getInstance().setUser(user);
```

State

Dalam kasus kode kita, design pattern State digunakan pada class Transaction untuk mengganti status dari Transaction, mulai dari Booked State sampai Exchanged State.

```java
public interface TransactionInterface {

    void update(Transaction transaction);
}
```

```java
public class ExchangedState implements TransactionInterface {

    @Override
    public void update(Transaction transaction) {
        System.out.println(x:"Ticket already exchanged");
    }
}
```

```java
public class BookedState implements TransactionInterface {

    @Override
    public void update(Transaction transaction) {
        transaction.state = new ExchangedState();
    }
}
```

```java
public Transaction(String id, String username, ArrayList<Order> orders, Timestamp transactionDate, double totalPrice) {
    this.id = id;
    this.username = username;
    this.orders = orders;
    this.transactionDate = transactionDate;
    this.totalPrice = totalPrice;
    this.state = new BookedState();
}
```

CLEAN CODE

Penamaan variable

Penamaan variable pada project ini konsisten menggunakan case sensitive.
Contoh: transactionList

```
public class Customer extends User {

    private ArrayList<Transaction> transactionList;
```

Meaningful name

Penamaan pada project ini mudah dimengerti.
Contoh: UserSessionManager

```
public class UserSessionManager {

    private static UserSessionManager instance;
    private User user = null;
```

Penggunaan comment

Project ini menggunakan comment yang jelas pada setiap bagiannya agar mudah dimengerti.
Contoh: // Perform action when merchandise button is clicked

```
// Get all from table Ticket
public ArrayList<Merchandise> getAllMerchandises(){
    ArrayList<Merchandise> merchandises = new ArrayList<>();
    try {
        conn.connect();
        String query = "SELECT * FROM merchandises";
        Statement stmt = conn.con.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        while (rs.next()) {
            Merchandise merchandise = new Merchandise();
            merchandise.setId(rs.getString(columnLabel:"ID"));
            merchandise.setName(rs.getString(columnLabel:"name"));
            merchandise.setPrice(rs.getInt(columnLabel:"Category"));
            merchandise.setStock(rs.getInt(columnLabel:"Stock"));
```

Penamaan Variable & Function

Penamaan variable dan function pada project ini konsisten menggunakan camel case.

```
public class TicketController {

    static DatabaseHandler conn = new DatabaseHandler();

    // Get all tickets from table Tickets
    public ArrayList<Ticket> getAllTickets() {···

    // Get all tickets by ConcertID from the table Ticket
    public ArrayList<Ticket> getAllTicketsByConcertId(int concertId) {···

    // Get a single ticket by ID from the Tickets table
    public Ticket getTicketById(int ticketId) {···

    // Insert ticket into table Tickets
    public boolean insertNewTicket(Ticket ticket) {···

    // Update ticket in table Tickets
    public boolean updateTicket(Ticket ticket, int oldId) {···
```

Penggunaan comment

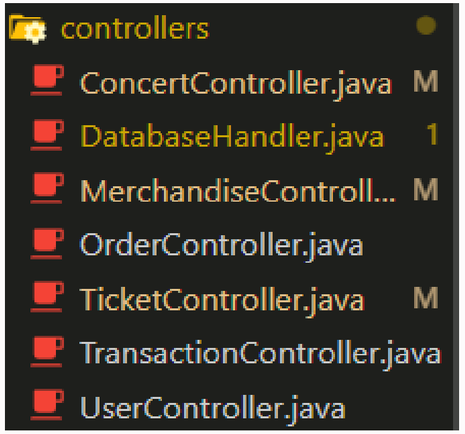
Misalnya, comment yang jelas pada setiap function agar mudah tujuan/maksud dari function tersebut mudah dimengerti.

SOLID Principle

Single Responsibility

Bertujuan agar setiap kelas atau modul hanya memiliki satu alasan untuk berubah.

```java
public class ConcertController {

    static DatabaseHandler conn = new DatabaseHandler();

    // Get all upcoming events from table Concerts
    public ArrayList<Concert> getAllConcerts() {···

    // Get concert by ID in table Concerts
    public Concert getConcertById(int concertId) {···

    // Insert concert into table Concerts
    public boolean insertNewConcert(Concert concert) {···

    // Update concert in table Concerts
    public boolean updateConcert(Concert concert, int oldId) {···

    // Update many concerts (ArrayList) in table Concerts
    public boolean updateConcerts(ArrayList<Concert> concerts, ArrayList<Integer> oldIds) {···
}
```



Open/Closed Principle

Open to Extension, Closed to Modification.
Dibandingkan kita mengubah-ubah suatu class, lebih baik kita membuat subclass dari class tersebut.

```java
public class Admin extends User {

    private String staffId;
    private String nik;
```

Liskov Substitution

```java
public class Customer extends User {

    private ArrayList<Transaction> transactionList;
```

Interface Segregation

```java
public interface TransactionInterface {

    void update(Transaction transaction);
}
```

Dependency Inversion

```java
public class TicketOrder extends Order {
```

```java
public class MerchandiseOrder extends Order {
```

```java
public abstract class Order {

    private int quantity;

    public Order() {
    }

    public Order(int quantity) {
        this.quantity = quantity;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}
```

THANK YOU

https://github.com/marcelandrean/RPL-1122006-1122014-1122017/