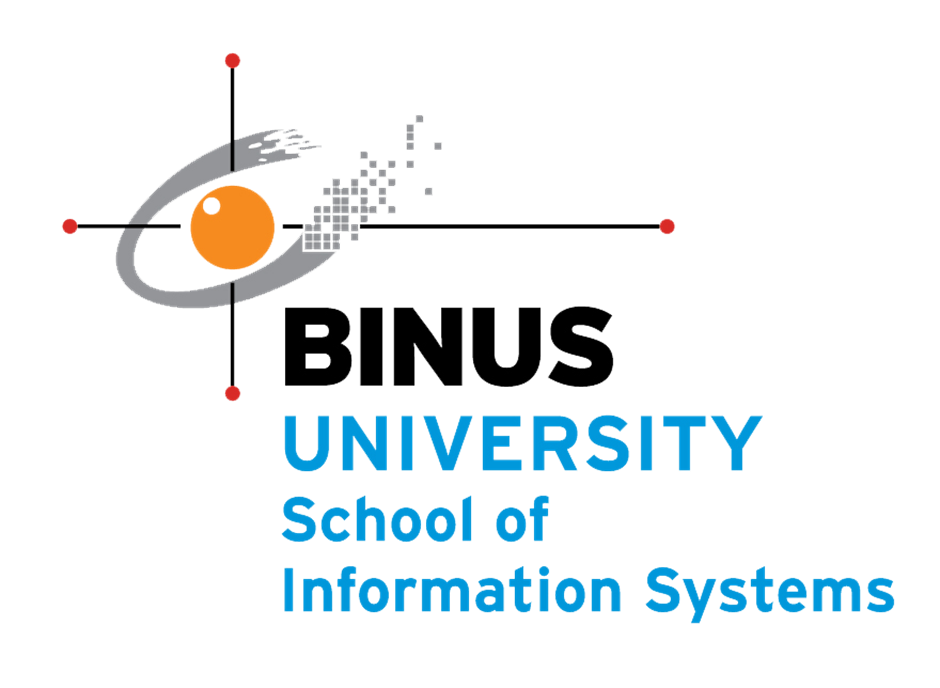
**ASSURANCE OF LEARNING**

**Eventpedia**



**2440035091 - Muhammad Rizal Rizky Naufal**

**2440033451 - Matthew Togi**

**2440023431 - Kenneth Calvin Santoso**

**2440027764 - Michael Nathaniel Hartono**

**BINA NUSANTARA UNIVERSITY**

**TANGERANG**

**2022/2023**

**BAB I**

**PENDAHULUAN**

**1.1 Latar Belakang**

Setelah pandemi Covid-19 mereda, banyak sekali event offline yang mulai diadakan kembali. Contohnya, saat ini sudah banyak event bazaar, konser, festival maupun event-event lainnya yang diadakan secara offline. Hal ini tentunya menyulitkan peserta event untuk mengatur jadwal untuk menghadiri event, apalagi jika banyak event yang berlangsung dalam kurun waktu sangat singkat. Selain itu, event offline juga memiliki kendala lain yaitu waktu menunggu antrian yang sangat lama ketika ingin membeli tiket. Oleh sebab itu, kami ingin membantu masyarakat untuk membuat aplikasi yang bukan hanya membantu user untuk membeli tiket atau mendaftarkan diri ke event tersebut, tetapi juga membantu mengingatkan user untuk datang ke event yang telah didaftarkan. Melalui aplikasi kami, user juga bisa melihat daftar event-event apa saja yang bisa diikuti, sehingga membantu user menemukan event yang ingin dihadiri sesuai dengan minat mereka.

**1.2 Tujuan**

* Membuat aplikasi untuk mempermudah user untuk mencari event-event mendatang dengan aplikasi kami
* Menampilkan informasi dari event yang akan datang seperti judul, lokasi, nama organisasi, durasi event dan harga tiket masuk
* Mempermudah user untuk mendaftar ke dalam suatu event
* Membantu organisasi untuk mempromosikan eventnya

**1.3 Manfaat**

* Mempermudah user untuk mencari event - event mendatang karena setiap event yang akan diselenggarakan di Indonesia dapat di daftarkan di aplikasi kami
* Membantu user dalam mencari informasi lebih mendalam mengenai event yang ingin dihadiri, seperti judul, lokasi, nama organisasi penyelenggara, dll.
* User bisa mendaftar ke suatu event dengan lebih mudah
* Organisasi akan mendapatkan peserta-peserta yang lebih banyak

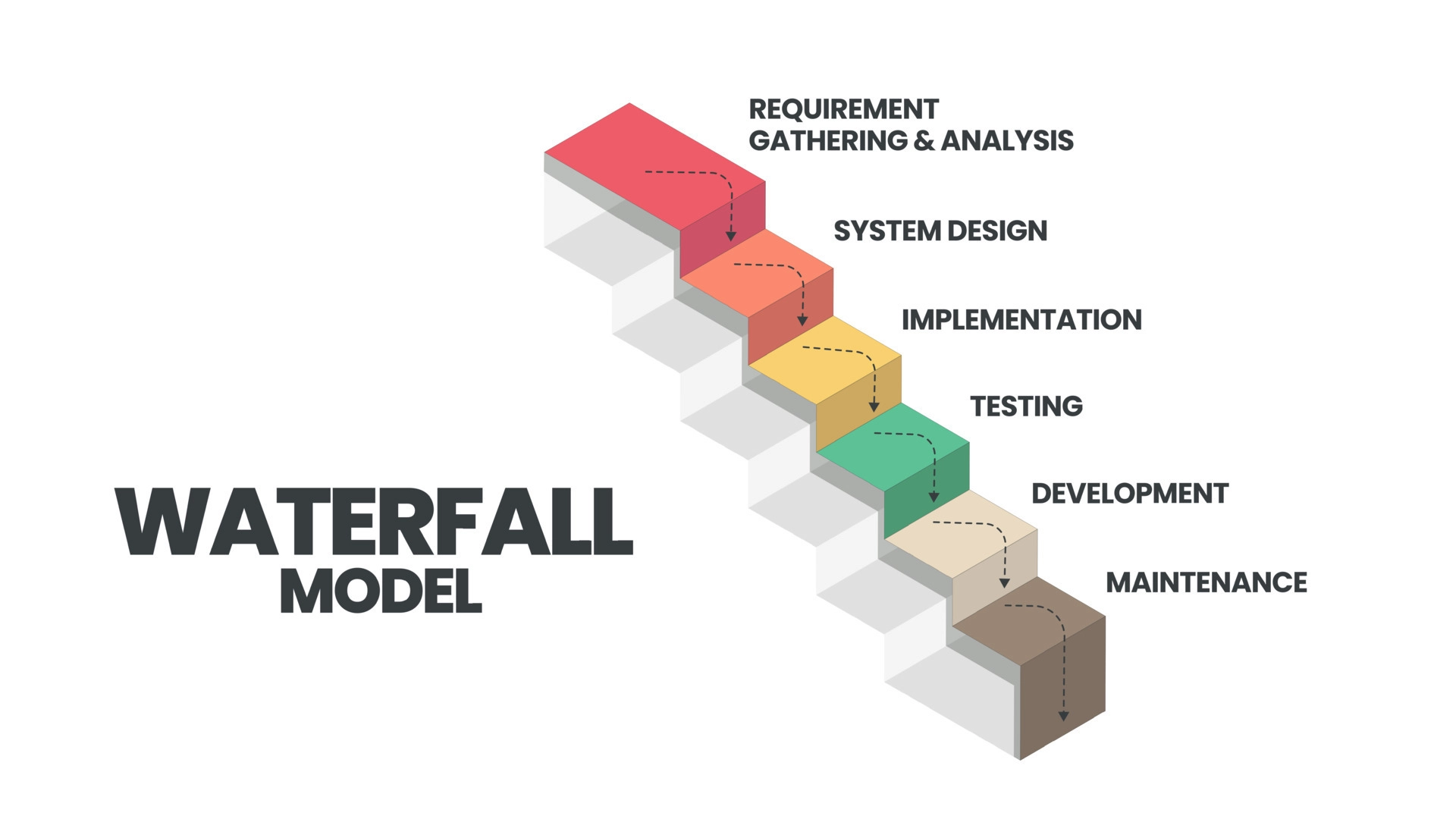
**BAB II**

**METODOLOGI**

**2.2. Metodologi Manajemen Proyek**

Model pengembangan dan perancangan software yang kami pilih adalah model Waterfall. Metode waterfall merupakan metode pengembangan software yang menekankan fase yang berurutan dan sistematis. Untuk mengembangkan software dengan waterfall model, diperlukan requirement yang jelas dan tidak berubah dalam jangka waktu pengembangan proyek.

Alasan kami memilih model Waterfall adalah, karena requirement-requirement dari Eventpedia ini sudah jelas dan tidak akan berubah, sehingga proses untuk menganalisa requirement dari proyek tidak perlu diulang. Selain itu, model Waterfall juga merupakan model yang paling sederhana untuk pengembangan proyek.



Dalam suatu proyek yang menggunakan waterfall model, terbagi lima tahap utama yang wajib untuk dilakukan:

* **Requirement**

Dalam tahapan requirements ini, kami mengumpulkan kebutuhan dari user seperti flow aplikasinya akan seperti apa, data apa saja yang diperlukan dalam aplikasi, dan lain-lain. Untuk mengetahui requirement yang diperlukan dalam aplikasi Eventpedia, kami akan menggunakan kuesioner.

Forms response chart. Question title: Apakah anda pernah mengalami kesulitan dalam mencari informasi mengenai event? 
. Number of responses: 41 responses.

Berdasarkan data yang kami dapatkan dari kuesioner, dari total 41 reponsi, masih ada sekitar setengah dari total responden yang mengalami kesusahan dalam mencari acara-acara yang akan diadakan.

Forms response chart. Question title: Apakah anda pernah mengalami kesulitan dalam melakukan pendaftaran event?
. Number of responses: 41 responses.

Dari responden yang ada, masih ada sekitar 56% juga yang mengalami masalah terkait pendaftaran suatu acara.

Berdasarkan data yang sudah dikumpulkan, Eventpedia akan dibuat untuk memastikan agar masalah pengguna terkait pencarian dan pendaftaran event dapat diselesaikan.

* **Design**

Dalam tahapan design, kami melakukan perancangan aplikasi sesuai dengan requirement yang sudah kami kumpulkan sebelumnya. Implementasi aplikasi akan dilakukan melalui bahasa pemrograman Java dengan mengaplikasikan prinsip-prinsip object oriented programming. Sebagai bagian dari object oriented programming kami membuat Class Diagram untuk merancang objek-objek yang akan kami gunakan dan hubungannya dengan obyek lain.

* **Implementation**

Implementasi proyek ini akan menggunakan bahasa pemrograman Java. Alasan kami menggunakan bahasa pemrograman ini adalah karena Java merupakan bahasa pemrograman yang paling dikuasai oleh anggota tim. Selain itu, aplikasi juga cocok jika diimplementasikan menggunakan Java.

Implementasi dalam Java dan hubungan antar class akan dibuat sesuai dengan rancangan Class Diagram yang sudah dibuat sebelumnya, dan disesuaikan dengan requirement yang telah dikumpulkan.

* **Testing**

Pada tahapan testing, kami melakukan pengujian apakah aplikasi yang kami buat sudah sesuai dengan kebutuhan user, flow aplikasi sudah sesuai dengan requirements, dan memastikan aplikasi berjalan dengan lancar

* **Deployment**

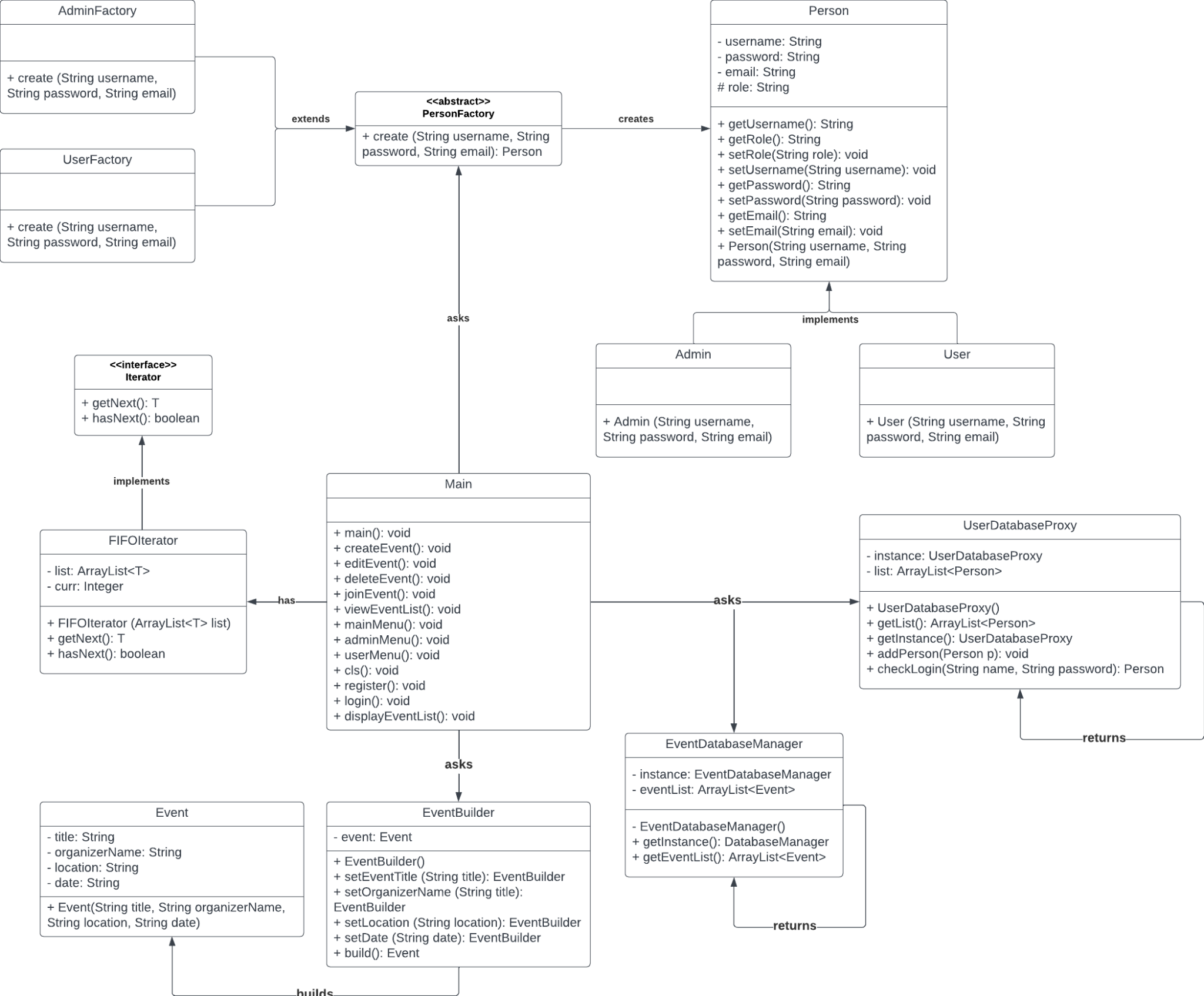
Aplikasi yang sudah melalui tahapan testing dan perbaikan akan dirilis kepada user dengan tujuan mendapatkan umpan balik dari user mengenai aplikasi kami.

* **Maintenance**

Pada tahapan maintenance, tim pengembangan aplikasi harus memastikan bahwa aplikasi masih berjalan dengan semestinya. Apabila ada bug yang terdapat di aplikasi, maka tim pengembang harus melakukan *bug fixing* atau sesuai dengan kesepakatan dengan user.

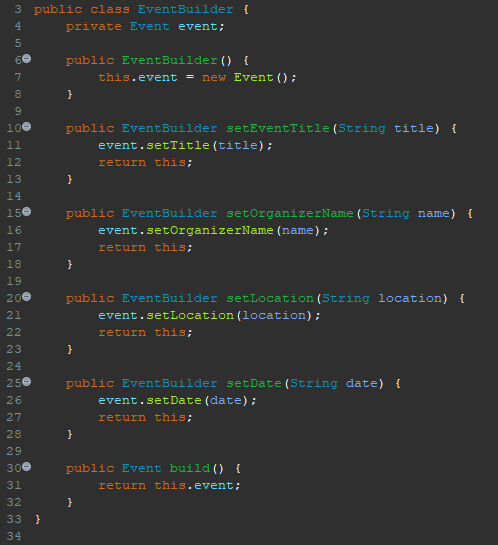
**BAB III**

**Design Pattern & Class Diagram**

****

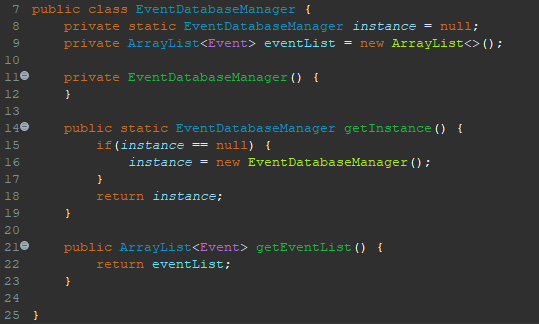
**3.1. Builder**

Kami menggunakan design pattern builder untuk dapat membuat event secara bertahap, sehingga pembuatan event akan menjadi lebih mudah dengan adanya design pattern builder



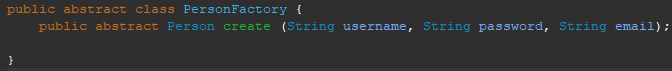
**3.2 Singleton**

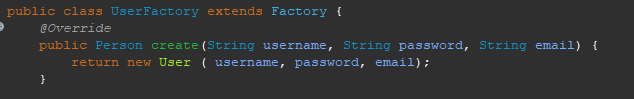
Kami menggunakan singleton untuk menyimpan data event yang sudah dibuat, sehingga data tersebut bisa diubah sewaktu- waktu dan di delete oleh admin. Tujuannya agar membatasi penggunaan class menjadi sebuah instance saja pada sebuah aplikasi. Sehingga pada suatu class dapat dipastikan hanya menghasilkan satu objek saja, dan tidak menghasilkan duplikasi data.

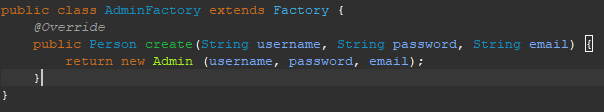


**3.3 Factory**

Kami menggunakan factory karena dengan menggunakan factory berarti kami cukup menentukan interface atau Abstract class untuk membuat objek tetapi biarkan subkelas memutuskan kelas mana yang akan dibuat. Dengan kata lain, subclass bertanggung jawab untuk membuat instance dari kelas tersebut.

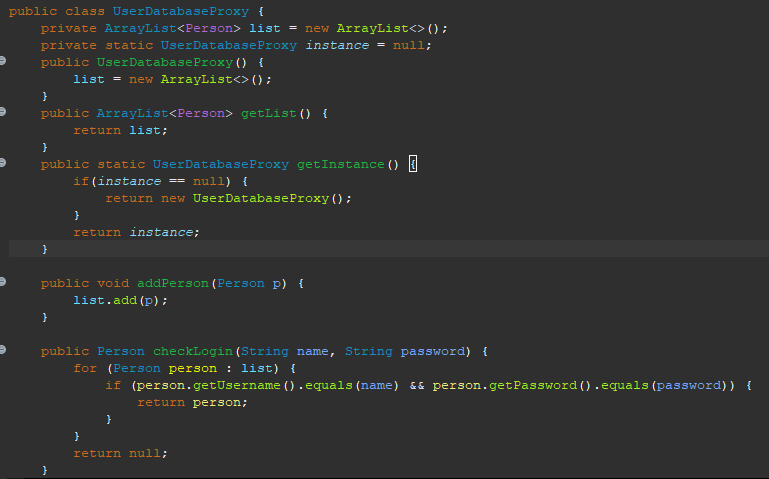






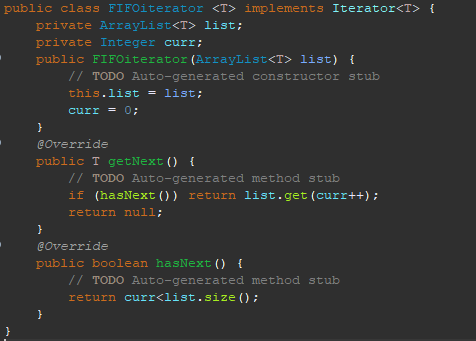
**3.4 Protection Proxy**

Kami menggunakan protection proxy untuk memastikan agar dapat melakukan autentikasi terhadap user login. Protection proxy memastikan agar user dapat melakukan login berdasarkan username dan password yang sudah didaftarkan.



**3.5 Iterator**

Kami menggunakan iterator untuk melakukan looping terhadap data yang tersedia dalam aplikasi Eventpedia. Iterator dapat memudahkan adanya looping, tanpa mengetahui apa tipe data dari koleksi yang tersedia.



**BAB IV**

**IMPLEMENTATION IN JAVA**

**Main.java**

public class Main {

EventDatabaseManager edm = EventDatabaseManager.getInstance();

UserDatabaseProxy udm = UserDatabaseProxy.getInstance();

Scanner sc = new Scanner(System.in);

public void displayEventList() {

ArrayList<Event> list = edm.getEventList();

FIFOiterator<Event> EventIterator = new FIFOiterator<>(list);

int i = 1;

while (EventIterator.hasNext()) {

Event e = EventIterator.getNext();

System.out.println("Event ID : " + i + "\n"

+ "Event Title: " + e.getTitle() + "\n"

+ "Organizer Name: " + e.getOrganizerName() + "\n"

+ "Event Venue: " + e.getLocation() + "\n"

+ "Event Date: " + e.getDate() + "\n");

}

}

public void register() {

String username;

String email;

String password;

String role;

do {

System.out.print("Input your username [5 - 30 (inclusive)]: ");

username = sc.nextLine();

} while (username.length() < 5 || username.length() > 30);

do {

System.out.print("Input your password [At least 8 characters long]: ");

password = sc.nextLine();

} while (password.length() < 8);

System.out.print("Input your email: ");

email = sc.nextLine();

do {

System.out.print("Input your role [Admin | User (Case sensitive)]: ");

role = sc.nextLine();

} while (!role.equals("Admin") && !role.equals("User"));

PersonFactory f;

if (role.equals("Admin")){

f = new AdminFactory();

Person p = f.create(username, password, email);

udm.addPerson(p);

} else {

f = new UserFactory();

Person p = f.create(username, password, email);

udm.addPerson(p);

}

System.out.println("Account succesfully registered");

System.out.println("Press enter to continue...");

sc.nextLine();

}

public void createEvent() {

String inputTitle;

String inputOrganizerName;

String inputLocation;

String inputDate;

do {

System.out.print("Input Event Title [1..32]: ");

inputTitle = sc.nextLine();

} while(inputTitle.length() < 1 || inputTitle.length() > 32);

do {

System.out.print("Input Organizer Name [1..32]: ");

inputOrganizerName = sc.nextLine();

} while(inputOrganizerName.length() < 1 || inputOrganizerName.length() > 32);

do {

System.out.print("Input Event Location [1..16]: ");

inputLocation = sc.nextLine();

} while(inputLocation.length() < 1 || inputLocation.length() > 16);

do {

System.out.print("Input Event Date [dd/mm/yy]: ");

inputDate = sc.nextLine();

} while(!(inputDate.charAt(2) == '/') || !(inputDate.charAt(5) == '/'));

edm.getEventList().add(new EventBuilder().setEventTitle(inputTitle).setOrganizerName(inputOrganizerName)

.setLocation(inputLocation).setDate(inputDate).build());

System.out.println("\nEvent successfuly created!");

System.out.println("Press enter to continue...");

sc.nextLine();

}

public void editEvent() {

int ch = -1;

int inputID;

String inputTitle, inputOrganizerName, inputLocation, inputDate;

if(edm.getEventList().isEmpty()) {

System.out.println("There are no events to modify.");

System.out.println("Press enter to continue...");

sc.nextLine();

}

else {

displayEventList();

do {

System.out.print("Input Event ID to modify [1.." + edm.getEventList().size() + "]: ");

inputID = sc.nextInt(); sc.nextLine();

} while(inputID < 1 || inputID > edm.getEventList().size());

do {

System.out.println("Choose option to modify: ");

System.out.println("1. Title");

System.out.println("2. Organizer Name");

System.out.println("3. Location");

System.out.println("4. Date");

ch = sc.nextInt(); sc.nextLine();

} while(ch < 1 || ch > 4);

switch(ch) {

case 1:

do {

System.out.print("Input Event Title [1..32]: ");

inputTitle = sc.nextLine();

} while(inputTitle.length() < 1 || inputTitle.length() > 32);

edm.getEventList().get(inputID-1).setTitle(inputTitle);

break;

case 2:

do {

System.out.print("Input Organizer Name [1..32]: ");

inputOrganizerName = sc.nextLine();

} while(inputOrganizerName.length() < 1 || inputOrganizerName.length() > 32);

edm.getEventList().get(inputID-1).setOrganizerName(inputOrganizerName);

break;

case 3:

do {

System.out.print("Input Event Location [1..16]: ");

inputLocation = sc.nextLine();

} while(inputLocation.length() < 1 || inputLocation.length() > 16);

edm.getEventList().get(inputID-1).setLocation(inputLocation);

break;

case 4:

do {

System.out.print("Input Event Date [dd/mm/yy]: ");

inputDate = sc.nextLine();

} while(!(inputDate.charAt(2) == '/') && !(inputDate.charAt(5) == '/'));

edm.getEventList().get(inputID-1).setDate(inputDate);

break;

}

System.out.println("Successfully edited event!");

System.out.println("Press enter to continue...");

sc.nextLine();

}

}

public void deleteEvent() {

int inputID;

if(edm.getEventList().isEmpty()) {

System.out.println("There are no events to delete.");

System.out.println("Press enter to continue...");

sc.nextLine();

}

else {

displayEventList();

do {

System.out.print("Input Event ID to delete [1.." + edm.getEventList().size() + "]: ");

inputID = sc.nextInt(); sc.nextLine();

} while(inputID < 1 || inputID > edm.getEventList().size());

edm.getEventList().remove(inputID-1);

System.out.println("\nEvent ID " + inputID + " has been deleted!");

System.out.println("Press enter to continue...");

sc.nextLine();

}

}

*// User Menu*

public void joinEvent() {

int inputID;

if (edm.getEventList().isEmpty()) {

System.out.println("There are no events at the moment.");

System.out.println("Press enter to continue...");

sc.nextLine();

} else {

viewEventList();

do {

System.out.print("Input Event ID to join [1.." + edm.getEventList().size() + "]: ");

inputID = sc.nextInt(); sc.nextLine();

} while(inputID < 1 || inputID > edm.getEventList().size());

}

System.out.println("Successfully joined event!");

}

public void viewEventList() {

if(edm.getEventList().isEmpty()) {

System.out.println("There are no events at the moment.");

System.out.println("Press enter to continue...");

sc.nextLine();

}

else {

displayEventList();

}

}

public Main() {

int opsi = 0;

do {

System.out.println("Welcome to Eventpedia!");

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

System.out.println("1. Register New Account");

System.out.println("2. Login");

System.out.println("3. Exit Application");

System.out.print(">> ");

opsi = sc.nextInt(); sc.nextLine();

switch(opsi) {

case 1:

register();

break;

case 2:

login();

break;

}

} while (opsi != 3);

System.out.println("Thank you for using Eventpedia!");

}

public void login() {

String username;

String password;

String role = null;

int flag = -1;

do {

System.out.print("Input your username: ");

username = sc.nextLine();

System.out.print("Input your password: ");

password = sc.nextLine();

Person p = udm.checkLogin(username, password);

if (p != null){

flag =1;

role = p.getRole();

}

else

{System.out.println("Incorrect username or password!");}

} while(flag!=1);

System.out.println("Login Suceed!!!");

System.out.println();

if (role.equals("Admin")) adminMenu();

else if (role.equals("User")) userMenu();

}

public void adminMenu() {

int ch = -1;

do {

System.out.println("Eventpedia Admin Menu");

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

System.out.println("1. Create New Event");

System.out.println("2. Edit Event");

System.out.println("3. Delete Event");

System.out.println("4. Exit to Main Menu");

System.out.print(">> ");

ch = sc.nextInt(); sc.nextLine();

switch(ch) {

case 1:

createEvent();

break;

case 2:

editEvent();

break;

case 3:

deleteEvent();

break;

}

} while (ch != 4);

}

public void userMenu() {

int ch = -1;

do {

System.out.println("Eventpedia User Menu");

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

System.out.println("1. Join Event");

System.out.println("2. View Event List");

System.out.println("3. Exit to Main Menu");

System.out.print(">> ");

ch = sc.nextInt(); sc.nextLine();

switch(ch) {

case 1:

joinEvent();

break;

case 2:

viewEventList();

break;

default:

break;

}

} while (ch != 3);

}

public void cls() {

for(int i=0;i<15;i++) System.out.println();

}

public static void main(String[] *args*) {

new Main();

}

}

**Event.java**

**public class Event {**

**private String title;**

**private String organizerName;**

**private String location;**

**private String date;**

**public Event() {**

**}**

**public Event(String *title*, String *organizerName*, String *location*, String *date*) {**

**super();**

**this.title = title;**

**this.organizerName = organizerName;**

**this.location = location;**

**this.date = date;**

**}**

**public String getTitle() {**

**return title;**

**}**

**public void setTitle(String *title*) {**

**this.title = title;**

**}**

**public String getOrganizerName() {**

**return organizerName;**

**}**

**public void setOrganizerName(String *organizerName*) {**

**this.organizerName = organizerName;**

**}**

**public String getLocation() {**

**return location;**

**}**

**public void setLocation(String *location*) {**

**this.location = location;**

**}**

**public String getDate() {**

**return date;**

**}**

**public void setDate(String *date*) {**

**this.date = date;**

**}**

**}**

Setiap event memiliki property berupa judul, nama EO, venue, dan tanggal event

**EventBuilder.java**

**public class EventBuilder {**

**private Event event;**

**public EventBuilder() {**

**this.event = new Event();**

**}**

**public EventBuilder setEventTitle(String *title*) {**

**event.setTitle(title);**

**return this;**

**}**

**public EventBuilder setOrganizerName(String *name*) {**

**event.setOrganizerName(name);**

**return this;**

**}**

**public EventBuilder setLocation(String *location*) {**

**event.setLocation(location);**

**return this;**

**}**

**public EventBuilder setDate(String *date*) {**

**event.setDate(date);**

**return this;**

**}**

**public Event build() {**

**return this.event;**

**}**

**}**

Class **EventBuilder** digunakan untuk membuat event dan di sini diterapkan **Builder Pattern**

**EventDatabaseManager.java**

**public class EventDatabaseManager {**

**private static EventDatabaseManager instance = null;**

**private ArrayList<Event> eventList = new ArrayList<>();**

**private EventDatabaseManager() {**

**}**

**public static EventDatabaseManager getInstance() {**

**if(instance == null) {**

**instance = new EventDatabaseManager();**

**}**

**return instance;**

**}**

**public ArrayList<Event> getEventList() {**

**return eventList;**

**}**

**}**

Class **EventDatabaseManager** digunakan untuk menyimpan data dari event dan di sini diterapkan **Singleton Pattern**

**PersonFactory.java**

**public abstract class PersonFactory {**

**public abstract Person create (String *username*, String *password*, String *email*);**

**}**

Class **PersonFactory** memiliki dua anak yaitu **AdminFactory** dan **UserFactory**. Di sini diterapkan **Factory Pattern**

**AdminFactory.java**

**public class AdminFactory extends PersonFactory {**

**@Override**

**public Person create(String *username*, String *password*, String *email*) {**

**return new Admin (username, password, email);**

**}**

**}**

Class **AdminFactory** merupakan anak dari **PersonFactory**

**UserFactory.java**

**public class UserFactory extends PersonFactory {**

**@Override**

**public Person create(String *username*, String *password*, String *email*) {**

**return new User ( username, password, email);**

**}**

**}**

Class **UserFactory** merupakan anak dari **PersonFactory**

**Person.java**

**public class Person {**

**private String username;**

**private String password;**

**private String email;**

**protected String role;**

**public String getUsername() {**

**return username;**

**}**

**public String getRole() {**

**return role;**

**}**

**public void setRole(String *role*) {**

**this.role = role;**

**}**

**public void setUsername(String *username*) {**

**this.username = username;**

**}**

**public String getPassword() {**

**return password;**

**}**

**public void setPassword(String *password*) {**

**this.password = password;**

**}**

**public String getEmail() {**

**return email;**

**}**

**public void setEmail(String *email*) {**

**this.email = email;**

**}**

**public Person(String *username*, String *password*, String *email*) {**

**super();**

**this.username = username;**

**this.password = password;**

**this.email = email;**

**}**

**}**

**Admin.java**

**public class Admin extends Person {**

**public Admin(String *username*, String *password*, String *email*) {**

**super(username, password, email);**

**this.role = "Admin";**

**}**

**}**

**User.java**

**public class User extends Person {**

**public User(String *username*, String *password*, String *email*) {**

**super(username, password, email);**

**this.role = "User";**

**}**

**UserDatabaseProxy.java**

**public class UserDatabaseProxy {**

**private ArrayList<Person> list = new ArrayList<>();**

**private static UserDatabaseProxy instance = null;**

**public UserDatabaseProxy() {**

**list = new ArrayList<>();**

**}**

**public ArrayList<Person> getList() {**

**return list;**

**}**

**public static UserDatabaseProxy getInstance() {**

**if(instance == null) {**

**return new UserDatabaseProxy();**

**}**

**return instance;**

**}**

**public void addPerson(Person *p*) {**

**list.add(p);**

**}**

**public Person checkLogin(String *name*, String *password*) {**

**for (Person person : list) {**

**if (person.getUsername().equals(name) && person.getPassword().equals(password)) {**

**return person;**

**}**

**}**

**return null;**

**}**

**}**

Class **UserDatabaseProxy** digunakan untuk pengecekan autentikasi. Pada bagian ini diterapkan **Protection Proxy Pattern**

**Iterator.java**

**public interface Iterator <T> {**

**public T getNext();**

**public boolean hasNext();**

**}**

Class **Iterator** ini menerapkan **Iterator Pattern** dimana **FIFOIterator** mengimplements interface **Iterator** ini

**FIFOIterator.java**

public class FIFOiterator <T> implements Iterator<T> {

private ArrayList<T> list;

private Integer curr;

public FIFOiterator(ArrayList<T> *list*) {

this.list = list;

curr = 0;

}

@Override

public T getNext() {

if (hasNext()) return list.get(curr++);

return null;

}

@Override

public boolean hasNext() {

return curr<list.size();

}

}

**VI.**

**CONCLUSION**

Demikian merupakan aplikasi Eventpedia yang sudah kami buat. Secara keseluruhan kami menggunakan 5 design pattern, dengan pembagian 3 creational design pattern (Singleton, Factory, Builder), 1 structural design pattern (Proxy) dan 1 behavioral design pattern (iterator). Masing-masing design pattern tersebut memiliki fungsi yang berbeda. Creational design pattern digunakan untuk membuat objek-objek dalam aplikasi kami seperti User, Event, dll. Structural design pattern proxy berguna untuk menambahkan autentikasi user, sementara behavioral design pattern iterator berguna untuk melakukan looping pada data. Dengan penggunaan design pattern tersebut, aplikasi kami dapat memiliki fungsionalitas yang lebih baik, *source code* yang lebih rapi dan mudah dibaca, serta lebih dapat membantu pengguna untuk menemukan event yang ingin diikuti.

**REFERENCES**

<https://socs.binus.ac.id/2018/12/21/waterfall-model/>

<https://socs.binus.ac.id/files/2018/12/son-3.jpg>

<https://refactoring.guru/design-patterns/factory-method>

<https://refactoring.guru/design-patterns/builder>

<https://refactoring.guru/design-patterns/builder>

<https://refactoring.guru/design-patterns/proxy>

<https://refactoring.guru/design-patterns/iterator>