CineMeet

Cinema em
qualquer lugar !

Nossa Equipe

- Adaury Oliveira
- Demetrius Souza
- Israel Erlich
- João Pedro Araújo
- João Pedro Batista
- Lucas Lucena
- Mathews Ivo
- Rafael Menezes
- Théo Moura
- Vinícius Gonçalves

Nossa Ideia

- **Organização de Watch Parties**

O CineMeet permite organizar Watch Parties, conectando amantes do cinema para assistir a filmes em grupo, promovendo interação social e personalização dos eventos

- **Avaliações e interatividade nos eventos**

O CineMeet incentiva avaliações de Watch Parties, promovendo feedback para aprimorar eventos, construir confiança na plataforma e conectar usuários com interesses semelhantes, fortalecendo laços sociais no universo do cinema

Design Patterns

Repository

O padrão Repository centraliza o acesso e manipulação de dados, abstraindo a lógica de persistência. **FriendRepository** e **EventRepository** seguem este padrão.

Singleton

O padrão Singleton garante instância de uma classe de uma classe. No **FriendServiceImpl**, anotado com @service, o Spring cria e gerencia um único bean, assegurando o uso da mesma instância em toda a aplicação.

Strategy

Define algoritmos intercambiáveis, permitindo flexibilidade. **FriendService** e **EventService** usam interfaces para alternar implementações conforme necessário, promovendo reutilização e adaptação do código.

Factory

O padrão Factory simplifica a criação de objetos, promovendo reutilização. Em **UserDTO** e **EventDTO** por exemplo, é usado para converter entidades em objetos transferíveis de forma consistente e eficiente.

Factory

EventDTO

```java
package com.CineMeetServer.dto;

import lombok.Data;

import java.util.Date;

@Data
public class EventDTO {

    private Long id;

    private String title;

    private Date date;

    private String description;
    private String movieName;
    private String movieImgUrl;
    private String userName;
    private Long userId;
}
```

UserDTO

```java
package com.CineMeetServer.dto;

import lombok.Data;

@Data
public class UserDTO {

    private Long id;

    private String email;

    private String name;

    private Double rating;
    private Long eventsHosted;
}
```

Utiliza-se do padrão Factory para simplificar a criação de instâncias de DTOs (EventDTO e UserDTO) e encapsular a lógica de inicialização. Isso promove a reutilização ao facilitar a conversão de entidades em objetos transferíveis entre camadas.

Factory

EVENT

```
public EventDTO getDto(){
        EventDTO dto = new EventDTO();

        dto.setId(id);
        dto.setTitle(title);
        dto.setDescription(description);
        dto.setDate(date);
        dto.setMovieName(movieName);
        dto.setMovieImgUrl(movieImgUrl);
        dto.setUserId(user.getId());
        dto.setUserName(user.getName());

        return dto;
    }
```

codesnap.dev

USER

```
public UserDTO getDto(){
      UserDTO dto = new UserDTO();

      dto.setId(id);
      dto.setEmail(email);
      dto.setName(name);
      dto.setRating(rating);
      dto.setEventsHosted(eventsHosted);

      return dto;
  }
```

codesnap.dev

Repository

EventRepository

```java
package com.CineMeetServer.repo;

import com.CineMeetServer.entities.Event;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Date;
import java.util.List;

@Repository
public interface EventRepository extends JpaRepository<Event, Long> {

    List<Event> findByDateAfter(Date date);


    List<Event> findByDateAfterAndUserIdIn(Date date, List<Long> friends);
}
```

FriendRepository

```java
package com.CineMeetServer.repo;

import com.CineMeetServer.entities.Friend;
import com.CineMeetServer.entities.User;
import com.CineMeetServer.enums.FriendStatus;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface FriendRepository extends JpaRepository<Friend, Long> {

    boolean existsByUserAndFriend(User user, User friend);

    List<Friend> findByFriendIdAndStatus(Long friendId, FriendStatus status);

//    List<Friend> findByFriendIdOrUserIdAndStatusIn(Long friendId, Long userId, List<FriendStatus> statuses);

    @Query("SELECT f FROM Friend f WHERE (f.user.id = :userId OR f.friend.id = :userId) AND f.status IN :statuses")
    List<Friend> findFriendsByUserIdAndStatus(@Param("userId") Long userId, @Param("statuses") List<FriendStatus> statuses);



    List<Friend> findByUserIdAndStatusOrFriendIdAndStatus(Long userId, FriendStatus status1,
                                                          Long friendId, FriendStatus status2);
}
```

Esses repositórios tornam o código mais limpo, desacoplado e fácil de testar, o que é exatamente o objetivo do padrão.

Strategy

FriendService

```java
package com.CineMeetServer.service.friend;

import com.CineMeetServer.dto.FriendDTO;
import com.CineMeetServer.entities.Friend;
import com.CineMeetServer.enums.FriendStatus;

import java.util.List;

public interface FriendService {

    FriendDTO sendFriendRequest(Long userId, Long friendId);

    List<FriendDTO> getFriendRequests(Long userId);

    List<FriendDTO> getFriends(Long userId);

    FriendDTO respondToFriendRequest(Long requestId, FriendStatus status);
}
```

EventService

```java
package com.CineMeetServer.service.event;

import com.CineMeetServer.dto.EventDTO;

import java.util.List;

public interface EventService {

    EventDTO createEvent(EventDTO dto);

    List<EventDTO> getFutureEvents();

    List<EventDTO> getFriendsEvents(Long userId);

    EventDTO getEvent(Long id);
}
```

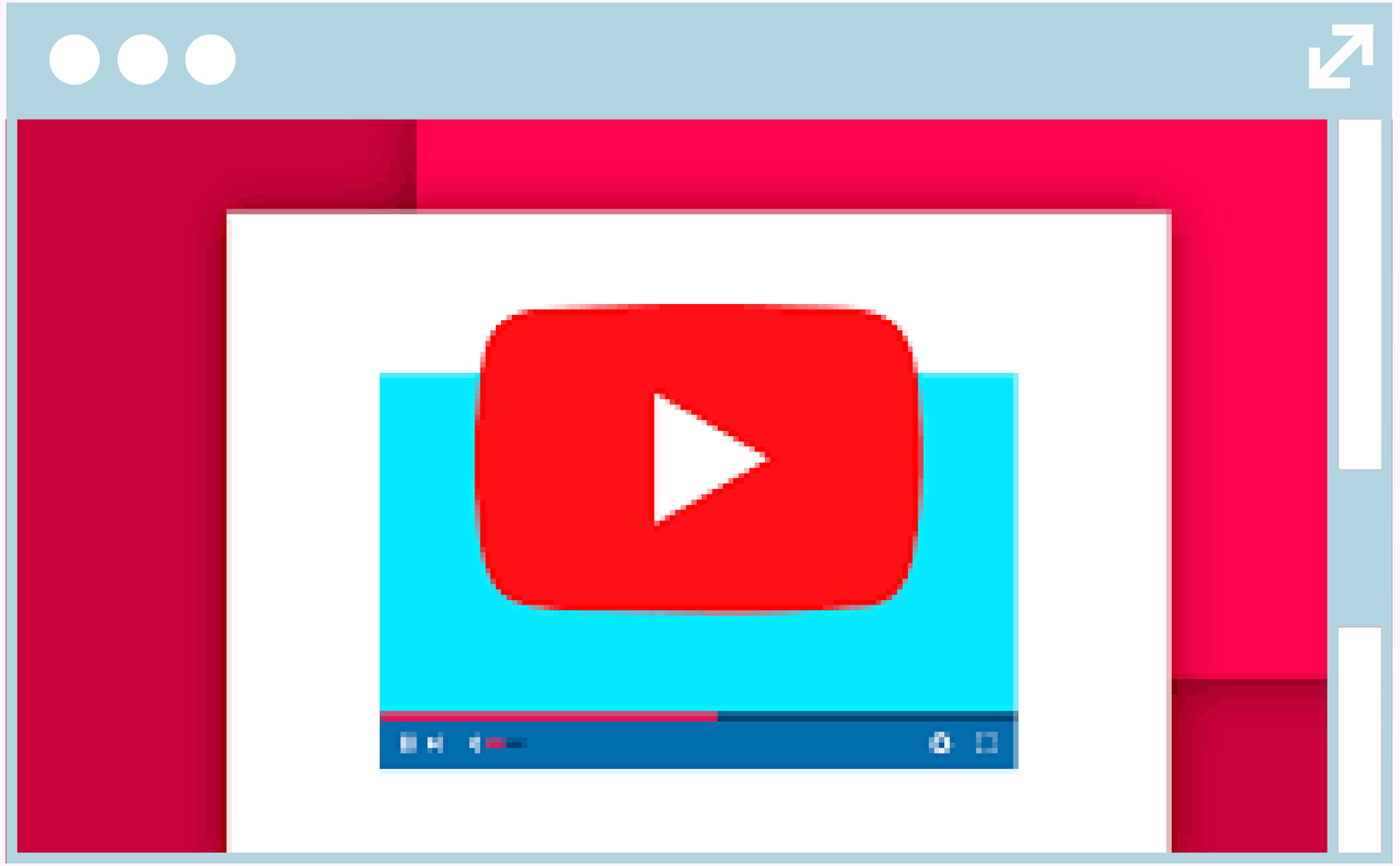
Ao definir as interfaces **FriendService** e **EventService**, é possível realizar múltiplas implementações. Isso proporciona flexibilidade para intercambiar diferentes algoritmos ou lógicas sem modificar o código cliente.

Singleton

```java
@Service
public class FriendServiceImpl implements FriendService{

    @Autowired
    private FriendRepository friendRepository;

    @Autowired
    private UserRepo userRepository;

    public FriendDTO sendFriendRequest(Long userId, Long friendId) {
        if (userId.equals(friendId)) {
            throw new EntityNotFoundException("Cannot send friend request to yourself.");
        }

        Optional<User> user = userRepository.findById(userId);
        Optional<User> friend = userRepository.findById(friendId);

        if (user.isEmpty() || friend.isEmpty()) {
            throw new EntityNotFoundException("User not found.");
        }

        // Check if there's already a friend request or friendship
        if (friendRepository.existsByUserAndFriend(user.get(), friend.get()) || friendRepository.existsByUserAndFriend(friend.get(), user.get())) {
            throw new EntityNotFoundException("Friend request already exists.");
        }

        Friend friendRequest = new Friend();
        friendRequest.setFriend(friend.get());
        friendRequest.setUser(user.get());
        friendRequest.setStatus(FriendStatus.PENDING);
        return friendRepository.save(friendRequest).getDto();
    }

    public List<FriendDTO> getFriendRequests(Long userId) {
```

A classe **FriendServiceImpl** é um **Singleton** porque está anotada com **@Service** no Spring Framework, que por padrão cria apenas uma instância única (bean) para ser reutilizada em toda a aplicação. Isso garante consistência e economia de recursos.

Utilização do site



Tecnologias usadas

Frontend 

Backend 

BDD 

Outros 

Thank You