**Слой репозиториев**

Слой репозиториев — это слой абстракции отвечающий за предоставление данных из различных источников. В данном проекте репозитории используются для получения данных из базы данных. Под каждую сущность системы создан свой интерфейс который расширяет интерфейсы JpaRepository и JpaSpecificationExecutor из фреймворка Spring, передавая в него через обобщение класс сущности, с которым будет работать репозиторий. Каждый интерфейс может содержать методы, реализация которых будет создана во время исполнения программы.Реализация создается или на основе названия метода и его сигнатуры, либо с помощью аннотации @Query. Проект содержит 6 репозиториев:

1. BoardGameRepository,
2. CommentRepository,
3. MeetRepository,
4. UserRepository,
5. RoleRepository,
6. RecommendationRepository,

Листинг 1. Код интерфейса «BoardGameRepository»

import BoardMeet.Backend.Filter.BoardGameFilter;

import BoardMeet.Backend.Model.BoardGame;

import BoardMeet.Backend.Model.Meet;

import org.springframework.data.domain.Page;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.JpaSpecificationExecutor;

import org.springframework.data.jpa.repository.Query;

import javax.persistence.criteria.CriteriaBuilder;

import java.util.Collection;

import java.util.List;

public interface BoardGameRepository extends JpaRepository<BoardGame,Long>, JpaSpecificationExecutor<BoardGame> {

List<BoardGame> findByNameContains(String filter);

List<BoardGame> findByGenre (String genre);

@Query(nativeQuery = true,value = "select board\_game.\* from board\_game join (select board\_game\_id,sum(interest\*cnt) as interestAll from (SELECT \* FROM \n" +

"(select a.board\_game\_id,interest,user\_id from recommendation join (select board\_game\_id from recommendation where user\_id = ?1 ) as a\n" +

"on a.board\_game\_id = recommendation.board\_game\_id)as b) as t1 join\n" +

"(select \* from (select count(user\_id) as cnt ,user\_id from\n" +

"(select recommendation.user\_id,interest,a.board\_game\_id from recommendation join (select user\_id,board\_game\_id from recommendation where user\_id = ?1 ) as a\n" +

"on a.board\_game\_id = recommendation.board\_game\_id)as b group by user\_id) as ass order by cnt DESC limit ?2) as t2 on t1.user\_id = t2.user\_id group by board\_game\_id) as c on c.board\_game\_id = board\_game.id order by c.interestAll DESC ",

countQuery = "select count(board\_game.id) from board\_game join (select board\_game\_id,sum(interest\*cnt) as interestAll from (SELECT \* FROM \n" +

"(select a.board\_game\_id,interest,user\_id from recommendation join (select board\_game\_id from recommendation where user\_id = ?1 ) as a\n" +

"on a.board\_game\_id = recommendation.board\_game\_id)as b) as t1 join\n" +

"(select \* from(select count(user\_id) as cnt ,user\_id from\n" +

"(select recommendation.user\_id,interest,a.board\_game\_id from recommendation join (select user\_id,board\_game\_id from recommendation where user\_id = ?1) as a\n" +

"on a.board\_game\_id = recommendation.board\_game\_id)as b group by user\_id)as ass order by cnt DESC limit ?2) as t2 on t1.user\_id = t2.user\_id group by board\_game\_id) as c on c.board\_game\_id = board\_game.id order by c.interestAll DESC ")

Page<BoardGame> getRecommendation(Long id,Long recommendationPeopleCount, Pageable page);

}

Код остальных репозиториев будет приведен в приложении

Слой сервисов

Слой сервисов это слой абстрации между слоем бизнес-логики и слоем презентации ( в данном проекте выполнен с помощью отдельной подсистемы).В данном проекте слой сервисов содержит бизнес-логику системы, получая данные из репозитория, обрабатывает их и передает контроллеру для предоставления пользователю. В проекте содержится 8 сервисов:

1. AuthenticationServiceImpl,
2. BoardGameServiceImpl,
3. CommentServiceImpl,
4. ControllAccessServiceImpl,
5. FileServiceImpl,
6. MeetServiceImpl,
7. RecommendationServiceImpl,
8. UserServiceImpl,

AuthenticationServiceImpl реализован с использование фреймворка Spring Security.Cервис содержит логику обработки входа в систему

Листин 2. метод login

public AuthenticationResponseDTO login(AuthenticationRequestDTO requestDto) {

try{

String username = requestDto.getUsername();

authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username,requestDto.getPassword()));

User user = userService.findByUsername(username);

if(user == null){

throw new UsernameNotFoundException(username);

}

String token = jwtTokenProvider.createToken(username);

Map<Object,Object> response = new HashMap<>();

response.put("authUser",user);

response.put("token",token);

return new AuthenticationResponseDTO(user,token);

}catch (AuthenticationException e){

throw new BadCredentialsException("invalid username or password");

}

}

FileServiceImpl содержит логику работы с файлами и их загрузкой на статический сервер, который располагается на, машине с запущенным сервисом. Имена загруженных файлов генерируются с использование UUID (всемирно уникальный идентификатор) для того, что бы сохраненный файлы были с уникальными названиями.

ControllAccessServiceImpl содержит логику контроля доступа к данным, то есть логика которая проверяет может ли пользователь удалить, изменить или создать определенную сущность в системе.

Листинг 3. Класс сущности «ControllAccessServiceImpl»

package BoardMeet.Backend.Service.Impl;

import BoardMeet.Backend.Exception.NoAccessException;

import BoardMeet.Backend.Model.Role;

import BoardMeet.Backend.Security.jwt.JwtUser;

import BoardMeet.Backend.Service.ControllAccessService;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.stereotype.Service;

import java.security.Principal;

import java.util.Set;

@Service

public class ControllAccessServiceImpl implements ControllAccessService {

@Override

public void tryAccess(Long userId) throws NoAccessException {

JwtUser principal = (JwtUser) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

Set<Role> roles = principal.getRole();

for (Role role:

roles) {

if( role.getName().equals("ROLE\_ADMIN") )

{

return;

}

}

if(userId.compareTo(principal.getId()) != 0){

throw new NoAccessException("NOT ACCESS user by id : " + userId);

}

}

@Override

public Long getIdUser() {

if(SecurityContextHolder.getContext().getAuthentication().getPrincipal().equals("anonymousUser")){

return 0L;

}

JwtUser user = (JwtUser) SecurityContextHolder.getContext().getAuthentication().getPrincipal();

return user.getId();

}

}

RecommendationServiceImpl содержит логику рекомендательно алгоритма для настольных игр. В нем есть 2 метода которые вызываются при получении пользователем информации о настольной игре и при комментирование настольной игры. Эти методы изменяют интерес игрока к определенной настольной игре

Листинг 4. Класс сущности «RecommendationServiceImpl»

package BoardMeet.Backend.Service.Impl;

import BoardMeet.Backend.Model.Recommendation;

import BoardMeet.Backend.Repository.RecommendationRepository;

import BoardMeet.Backend.Service.ControllAccessService;

import BoardMeet.Backend.Service.RecommendationService;

import BoardMeet.Backend.DTO.CommentCreateDTO;

import org.springframework.stereotype.Service;

@Service

public class RecommendationServiceImpl implements RecommendationService {

private final RecommendationRepository recommendationRepository;

private final ControllAccessService controllAccessService;

public RecommendationServiceImpl(RecommendationRepository recommendationRepository, ControllAccessService controllAccessService) {

this.recommendationRepository = recommendationRepository;

this.controllAccessService = controllAccessService;

}

@Override

public void changeByComment(CommentCreateDTO comment) {

Recommendation r = recommendationRepository.findByUserAndBoardGame(comment.getAuthorId(),comment.getGameId());

if(r == null){

r = new Recommendation(comment);

}

r.setInterest(r.getInterest()+20);

recommendationRepository.save(r);

}

@Override

public void changeByFollowingLink(Long boardGameId) {

Long id = controllAccessService.getIdUser();

if (id <= 0){

return;

}

Recommendation r = recommendationRepository.findByUserAndBoardGame(controllAccessService.getIdUser(),boardGameId);

if(r == null){

r = new Recommendation(boardGameId,id);

}

r.setInterest(r.getInterest()+1);

recommendationRepository.save(r);

}

}

MeetServiceImpl кроме логики взаимодействия с сущностью еще использует метод выполняющийся с определенным периодом. Периодичность выполнения реализована с помощью аннотации @Sheduled, у которой есть параметр fixedDelay, который отвечает за периодичность выполнения метода. Этот метод обновляет состояние мероприятий.

Листинг 4. метод «refreshStateMeet»

@Scheduled(fixedDelay = 10000)

public void refreshStateMeet(){

List<Meet> meets = meetRepository.findAll();

for(Meet meet : meets){

meet.refreshState();

}

meetRepository.saveAll(meets);

}

Остальные сервисы содержат логику работы с сущностями и содержат методы добавления, удаления, изменения и чтения сущностей с различными правилами и проверками.

Код остальных сервисов будет приведен в приложении.

Слой Контроллеров

Слой контроллеров это слой который отвечает за обработку запросов от пользователя и управление бизнес-логикой приложения. В данном проекте в контроллерах мы указываем соответствие пути и логики его обработки, а также доступность путей для пользователей с определенными ролями, и вызываем методы сервисов ответственных за обработку запроса. При создании контроллеров были использованы аннотации из фреймворка Spring. В проекте 5 контроллеров:

1. AuthenticationController,
2. BoardGamesController,
3. CommentsController,
4. MeetsController,
5. UsersController

Листинг 5. Класс «UsersController»

package BoardMeet.Backend.Controller;

import BoardMeet.Backend.Exception.NoAccessException;

import BoardMeet.Backend.Exception.NotAccessExtensionException;

import BoardMeet.Backend.Exception.NotFoundUserException;

import BoardMeet.Backend.Exception.UserExistException;

import BoardMeet.Backend.Service.UserService;

import BoardMeet.Backend.DTO.UserRegisterDTO;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import javax.annotation.security.RolesAllowed;

import javax.validation.Valid;

import java.io.IOException;

@RestController

@RequestMapping(value = "/api/users/")

public class UsersController {

private final UserService userService;

@Autowired

public UsersController(UserService userService) {

this.userService = userService;

}

@PostMapping("register")

public ResponseEntity<?> register(@RequestBody UserRegisterDTO userDTO){

try {

return new ResponseEntity(userService.register(userDTO), HttpStatus.OK);

}catch (UserExistException e) {

return new ResponseEntity(e.getMessage(),HttpStatus.BAD\_REQUEST);

}

}

@PostMapping("{id}/uploadAvatar")

public ResponseEntity<?> upload(@RequestParam("image")MultipartFile avatar,@PathVariable Long id){

try {

return new ResponseEntity<>(userService.uploadAvatar(avatar,id),HttpStatus.OK);

}catch (IOException e){

return new ResponseEntity<>(e.getMessage(),HttpStatus.BAD\_REQUEST);

}catch (NotFoundUserException e ){

return new ResponseEntity<>(e.getMessage(),HttpStatus.NOT\_FOUND);

} catch (NotAccessExtensionException e) {

return new ResponseEntity(e.getMessage(),HttpStatus.BAD\_REQUEST);

} catch (NoAccessException e){

return new ResponseEntity(e.getMessage(),HttpStatus.BAD\_REQUEST);

}

}

@GetMapping

public ResponseEntity<?> getAll(){

return new ResponseEntity<>(userService.getAll(),HttpStatus.OK);

}

@GetMapping("{id}")

public ResponseEntity<?> get(@PathVariable Long id){

try {

return new ResponseEntity<>(userService.get(id),HttpStatus.OK);

}catch (NotFoundUserException e ){

return new ResponseEntity<>(e.getMessage(),HttpStatus.NOT\_FOUND);

}

}

@RolesAllowed("ROLE\_ADMIN")

@DeleteMapping("{id}")

public ResponseEntity<?> delete(@PathVariable Long id){

try {

userService.delete(id);

return new ResponseEntity<>(HttpStatus.OK);

}catch (NotFoundUserException e ){

return new ResponseEntity<>(e.getMessage(),HttpStatus.NOT\_FOUND);

}

}

@GetMapping("{id}/createdMeet")

public ResponseEntity<?> getCreatedMeet(@PathVariable Long id){

try {

return new ResponseEntity<>(userService.getCreatedMeet(id),HttpStatus.OK);

}catch (NotFoundUserException e ){

return new ResponseEntity<>(e.getMessage(),HttpStatus.NOT\_FOUND);

}

}

@GetMapping("{id}/joinedMeet")

public ResponseEntity<?> getJoinedMeet(@PathVariable Long id){

try {

return new ResponseEntity<>(userService.getJoinedMeet(id),HttpStatus.OK);

}catch (NotFoundUserException e ){

return new ResponseEntity<>(e.getMessage(),HttpStatus.NOT\_FOUND);

}

}

@GetMapping("{id}/createdBoardGame")

public ResponseEntity<?> getCreatedBoardGame(@PathVariable Long id){

try {

return new ResponseEntity<>(userService.getCreatedBoardGame(id),HttpStatus.OK);

}catch (NotFoundUserException e ){

return new ResponseEntity<>(e.getMessage(),HttpStatus.NOT\_FOUND);

}

}

}

Код остальных контроллеров будет приведен в приложении.

Фильтрация и Сортировка

В данной системе реализована фильтрация и сортировка настольных игр. Фильтрация и сортировка происходит по 8 различным полям:

1. имя игры,
2. жанр игры,
3. автор игры,
4. художник игры,
5. издатель игры,
6. пользовательский рейтинг,
7. сложность игры,
8. количество игроков,

Логику сортировки и фильтрации содержит класс BoardGameFilter, который реализовывает интерфейс Specification из фреймворка Spring Data. Объект это класса с полями, по которым проводятся операции сортировки и фильтрации мы получаем в контроллере и передаем его в сервис, который в свою очередь передает его в репозиторий, где вызывается метод toPredicate интерфейса Specification реализованный в классе BoardGameFilter, который фильтрует и сортирует сущности из базы данных.

Листинг 6. метод «toPredicate»

@Override

public Predicate toPredicate(Root<BoardGame> root, CriteriaQuery<?> query, CriteriaBuilder criteriaBuilder) {

ArrayList<Predicate> predicates = new ArrayList<>();

if (StringUtils.isNotBlank(name))

{

predicates.add(criteriaBuilder.like(root.get("name"), "%" + name + "%"));

}

if (StringUtils.isNotBlank(genre))

{

predicates.add(criteriaBuilder.equal(root.get("genre"), genre));

}

if (StringUtils.isNotBlank(authorsGame))

{

predicates.add(criteriaBuilder.like(root.get("authorsGame"), "%" + authorsGame + "%"));

}

if (StringUtils.isNotBlank(artists))

{

predicates.add(criteriaBuilder.like(root.get("artists"), "%" + artists + "%"));

}

if (StringUtils.isNotBlank(publishers))

{

predicates.add(criteriaBuilder.like(root.get("publishers"), "%" + publishers + "%"));

}

if(ratingUser!=null){

predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get("ratingUser"),ratingUser));

}

if(weightGameUser!=null){

predicates.add(criteriaBuilder.lessThanOrEqualTo(root.get("weightGameUser"),weightGameUser));

}

if(rangeOfPlayersMax!=null){

predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get("rangeOfPlayersMax"),rangeOfPlayersMax));

}

if(rangeOfPlayersMin!=null){

predicates.add(criteriaBuilder.greaterThanOrEqualTo(root.get("rangeOfPlayersMin"),rangeOfPlayersMin));

}

try {

if (StringUtils.isNotBlank(sortBy)) {

if (StringUtils.isNotBlank(sortDirection)) {

if (sortDirection.equals("DESC")) {

query.orderBy(criteriaBuilder.desc(root.get(sortBy)));

} else if (sortDirection.equals("ASC")) {

query.orderBy(criteriaBuilder.asc(root.get(sortBy)));

}

} else {

query.orderBy(criteriaBuilder.desc(root.get(sortBy)));

}

}

}catch (IllegalArgumentException e){

}

return predicates.size() <= 0 ? null : criteriaBuilder.and(predicates.toArray(new Predicate[predicates.size()]));

}

Рекомендательный алгоритм

В данной системе предусмотрен алгоритм рекомендации настольных игр. Он работает по принципу того, что при каждом комментирование или просмотре настольной игры мы увеличиваем интерес к этой игре пользователя в базе данных. Далее при запросе рекомендации для определенного пользователя мы смотрим какие игры интересны, пользователю, далее мы просматриваем людей которым так же интересны данные игры, отбираем людей с самыми похожими играми и смотрим игры у этих людей, складываем их интерес к данным играм, и выдаем список рекомендованных игр. Рекомендательный алгоритм реализован в классах RecommendationServiceImpl, а также в интерфейсах BoardGameRepository,RecommendationRepository.

В RecommendationServiceImpl реализована логика повышения интереса к игре при комментировании и просмотре настольной игры.

Листинг 7. методы «changeByComment» и «changeByFollowingLink»

@Override

public void changeByComment(CommentCreateDTO comment) {

Recommendation r = recommendationRepository.findByUserAndBoardGame(comment.getAuthorId(),comment.getGameId());

if(r == null){

r = new Recommendation(comment);

}

r.setInterest(r.getInterest()+20);

recommendationRepository.save(r);

}

@Override

public void changeByFollowingLink(Long boardGameId) {

Long id = controllAccessService.getIdUser();

if (id <= 0){

return;

}

Recommendation r = recommendationRepository.findByUserAndBoardGame(controllAccessService.getIdUser(),boardGameId);

if(r == null){

r = new Recommendation(boardGameId,id);

}

r.setInterest(r.getInterest()+1);

recommendationRepository.save(r);

}

В интерфейсе RecommendationRepository реализована логика получения строки рекомендации для определенной игры и пользователя

Листинг 7. метод «findByUserAndBoardGame»

@Query(nativeQuery = true,value = "SELECT \* FROM recommendation WHERE user\_id = ?1 and board\_game\_id = ?2 ")

Recommendation findByUserAndBoardGame(Long userId, Long boardGameId);

В интерфейсе BoardGameRepository с помощью SQL-кода в аннотации @Query реализован алгоритм получения списка рекомендованных игр

Листинг 8. алгоритм получения списка рекомендованных игр на SQL

select board\_game.\* from board\_game join (select board\_game\_id,sum(interest\*cnt) as interestAll from (SELECT \* FROM \n" +

"(select a.board\_game\_id,interest,user\_id from recommendation join (select board\_game\_id from recommendation where user\_id = ?1 ) as a\n" +

"on a.board\_game\_id = recommendation.board\_game\_id)as b) as t1 join\n" +

"(select \* from (select count(user\_id) as cnt ,user\_id from\n" +

"(select recommendation.user\_id,interest,a.board\_game\_id from recommendation join (select user\_id,board\_game\_id from recommendation where user\_id = ?1 ) as a\n" +

"on a.board\_game\_id = recommendation.board\_game\_id)as b group by user\_id) as ass order by cnt DESC limit ?2) as t2 on t1.user\_id = t2.user\_id group by board\_game\_id) as c on c.board\_game\_id = board\_game.id order by c.interestAll DESC ",