**Peer review report**

Fontys University of Applied Science Software

Project Plan Individual project:

Logo

Description automatically generated

Tutor: Tim Kurvers & Maja pesic

Student: Mohammed Al Harbi

4089553

Contents

[Introduction 3](#_Toc75101465)

[Criteria checked for code review 3](#_Toc75101466)

[Henaknowledge code snippet 3](#_Toc75101467)

[Henaknowledge code review received 7](#_Toc75101468)

[Kirill Simonov code snippet 8](#_Toc75101469)

[code review I sent to Kirill Smirnov regarding his code snippet 10](#_Toc75101470)

# Introduction

In this document, some code in henaknowledge project gets a code review as well as showing a code from another project and providing code review as well.

# Criteria checked for code review

 Functional Defects

 Problems with the logic

 Missing Validation (e.g., edge cases)

 Usage of API

 Design Patterns

 Architectural Issues

 Testability

 Readability

 Security

 Naming conventions

 Team Coding Style

 Documentation

 Use of best practices

 Language-specific issues

 Use of deprecated methods

 Performance (e.g., complexity of the solution

# Henaknowledge code snippet

package moee.henaknowledge.repository;

import moee.henaknowledge.dal\_interfaces.IExperienceDAL;

import moee.henaknowledge.dal\_interfaces.IExperienceOpinionDAL;

import moee.henaknowledge.module.Experience;

import moee.henaknowledge.module.ExperienceOpinion;

import moee.henaknowledge.util.points;

import org.hibernate.engine.transaction.jta.platform.internal.SynchronizationRegistryBasedSynchronizationStrategy;

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

import org.springframework.stereotype.Repository;

import java.util.List;

import java.util.Optional;

@Repository

public class ExperienceDalJPA implements IExperienceDAL {

@Autowired

IExperienceRepository repos;

@Autowired

IStudentRepository studentRepos;

@Autowired

ITeacherRepository teacherRepos;

@Autowired

IExperienceOpinionRepository Opinionrepos;

@Override

public Optional<Experience> getExperienceByExperienceID(int ID) {

return repos.findByExperienceID(ID);

}

@Override

public List<Experience> getAllExperiences() {

return repos.findAll();

}

private void increasePointsPerExperience(Experience experience ){

var publisher = experience.getPersonID();

for (var student:

studentRepos.findAll()) {

if(student.getPersonID() == publisher){

studentRepos.updatePoints(publisher,student.getPoints()+points.*pointsPerExperience*);

}

}

for (var teacher:

teacherRepos.findAll()) {

if(teacher.getPersonID() == publisher){

teacherRepos.updatePoints(publisher,teacher.getPoints()+points.*pointsPerExperience*);

}

}

}

private void increasePointsPerLike(Experience experience ){

var publisher = experience.getPersonID();

for (var student:

studentRepos.findAll()) {

if(student.getPersonID() == publisher){

studentRepos.updatePoints(publisher,student.getPoints()+points.*pointsPerLike*);

}

}

for (var teacher:

teacherRepos.findAll()) {

if(teacher.getPersonID() == publisher){

teacherRepos.updatePoints(publisher,teacher.getPoints()+points.*pointsPerLike*);

}

}

}

private void decreasePointPerDislike(Experience experience ){

var publisher = experience.getPersonID();

for (var student:

studentRepos.findAll()) {

if(student.getPersonID() == publisher){

studentRepos.updatePoints(publisher,student.getPoints()-points.*pointsPerDislike*);

}

}

for (var teacher:

teacherRepos.findAll()) {

if(teacher.getPersonID() == publisher){

teacherRepos.updatePoints(publisher,teacher.getPoints()-points.*pointsPerDislike*);

}

}

}

@Override

public void AddExperience(Experience experience) {

repos.save(experience);

increasePointsPerExperience(experience);

}

@Override

public void DeleteExperienceByID(int experienceID) {

repos.deleteById(experienceID);

}

@Override

public void SetExperienceById(int experienceID, String updatedTitle, String updatedDescription) {

repos.updatedExperienceByExperienceID(experienceID,updatedTitle,updatedDescription);

}

private ExperienceOpinion getExperienceOpinionHelpingMethod(int experienceID, int personID) {

//check if experience opinion exists given experienceID and Person ID

var opinion = Opinionrepos.findExperienceOpinionByPersonIDAndExperienceID(

personID,experienceID

);

if(opinion.isPresent()){

return opinion.get();

}

return null;

}

@Override

public void like(int experienceID, int personID) {

var theExperience = repos.findById(experienceID);

var theExperienceOpinion = getExperienceOpinionHelpingMethod(experienceID, personID);

if(theExperienceOpinion == null){

//the person would like to like or dislike for the first time in the specified experienceID

Opinionrepos.save(new ExperienceOpinion(1,0,experienceID,personID));

if(theExperience.isPresent()){

repos.likeTheExperience(experienceID,theExperience.get().getLikes()+1);

increasePointsPerLike(theExperience.get());

}

}

else {

if(theExperienceOpinion.getLikes() == 1) {

// trying to like the experience again

}

else if ( theExperienceOpinion.getDislikes() == 1) {

Opinionrepos.updateOpinion(theExperienceOpinion.getOpinionID(),1,0);

if(theExperience.isPresent()) {

repos.dislikesTheExperience(experienceID, theExperience.get().getDislikes() - 1);

repos.likeTheExperience(experienceID, theExperience.get().getLikes() + 1);

increasePointsPerLike(theExperience.get());

}

}

}

}

@Override

public void dislike(int experienceID, int personID) {

var theExperience = repos.findById(experienceID);

var theExperienceOpinion = getExperienceOpinionHelpingMethod(experienceID, personID);

if(theExperienceOpinion == null){

//the person would like to like or dislike for the first time in the specified experienceID

Opinionrepos.save(new ExperienceOpinion(0,1,experienceID,personID));

if(theExperience.isPresent()) {

repos.dislikesTheExperience(experienceID, theExperience.get().getDislikes()+1);

decreasePointPerDislike(theExperience.get());

}

}

else {

if(theExperienceOpinion.getLikes() == 1) {

Opinionrepos.updateOpinion(theExperienceOpinion.getOpinionID(),0,1);

if(theExperience.isPresent()) {

repos.dislikesTheExperience(experienceID, theExperience.get().getDislikes() + 1);

repos.likeTheExperience(experienceID, theExperience.get().getLikes() - 1);

decreasePointPerDislike(theExperience.get());

}

}

else if ( theExperienceOpinion.getDislikes() == 1) {

// trying to dislike the experience again

}

}

}

}

# Henaknowledge code review received

I contacted Kirill Smirnov to share his thoughts about the code and this is what I received:

1. dislike method (final else does nothing),
2. poor error handling,   
   good naming of variables and methods,
3. easily readable,
4. consistent patterns and style of code,
5. documents present discussing important design choices,
6. everything is functional,
7. code is SOLID,
8. many loops can cause longer loading times.

# Kirill Simonov code snippet

package Project.DataAccess;

import Project.InterfacesDAL.IPostDAL;

import Project.Logic.Model.IPost;

import Project.Logic.Model.Post;

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

import org.springframework.context.annotation.Primary;

import org.springframework.stereotype.Repository;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.List;

@Primary

@Repository

public class PostDAL implements IPostDAL {

@Autowired

IPostRepository repo;

@Override

public void deletePost(int id) {repo.deletePost(id);}

@Override

public void addPost(Post post) {repo.save(post);}

@Override

public List<Post> getAllPosts(){return repo.findAll();}

@Override

public List<IPost> GetPostsFromUser(String username){

return repo.GetPostsFromUser(username);}

public List<IPost> SearchPosts(String title) {

return repo.SearchPosts(title); }

@Override

public int CheckLikes(int postId) {

List<Integer> posts = repo.CheckLikes(postId);

if(posts != null){

return posts.get(0);

}

else{

return 0;

}

}

@Override

public void ChangeLikes(int postId, int changed\_likes) {

repo.ChangeLikes(postId,changed\_likes);

}

public List<Object[]> GetFollowedPosts(String username){

return repo.GetFollowedPosts(username);

}

public List<Object[]> GetLikedPosts(String username){

Object[] o = repo.GetLikedPosts(username).get(0);

System.*out*.println(String.*valueOf*(o[0]));

return repo.GetLikedPosts(username);

}

public List<Object[]> GetPopularPosts(){

long DAY\_IN\_MS = 1000 \* 60 \* 60 \* 24;

Date current\_date = new Date();

Date date = new Date(System.*currentTimeMillis*() - (7 \* DAY\_IN\_MS));

SimpleDateFormat formatter = new SimpleDateFormat("yyy-MM-dd");

return repo.GetPopularPosts(formatter.format(current\_date),formatter.format(date));

}

public List<Object[]> ShowPosts(String username,String type){

if(type.equals("Recent")){

return repo.GetFollowedPosts(username);

}

if(type.equals("Most Liked")){

long DAY\_IN\_MS = 1000 \* 60 \* 60 \* 24;

Date date = new Date(System.*currentTimeMillis*() - (7 \* DAY\_IN\_MS));

SimpleDateFormat formatter = new SimpleDateFormat("yyy-MM-dd");

return repo.GetMostLikedFollowedPosts(username,formatter.format(date));

}

if(type.equals("Top")){

long DAY\_IN\_MS = 1000 \* 60 \* 60 \* 24;

Date current\_date = new Date();

Date date = new Date(System.*currentTimeMillis*() - (7 \* DAY\_IN\_MS));

SimpleDateFormat formatter = new SimpleDateFormat("yyy-MM-dd");

return repo.GetTopPosts(formatter.format(current\_date),formatter.format(date));

}

else{

List<Object[]> posts = new ArrayList<>();

System.*out*.println("The type is incorrect for the timeline request");

return posts;

}

}

}

# code review I sent to Kirill Smirnov regarding his code snippet

1. Object[] should be replaced with the concrete object type,
2. usage of auto-wiring is correct,
3. ShowPosts function could be improved by using switch case or adding else if(s) instead of using 3 separate if statements with one else,
4. easily readable,
5. consistent patterns and style of code,
6. everything is functional, code is SOLID,
7. simple logic thus high performance,
8. deletePost function can be using DeleteByID which is built in function the repository offers by default instead of creating your own deletePost method.