**Додаток 2 Лістинг**

# Package com.folva.moderneastculture

## Main

package com.folva.moderneastculture;

import com.folva.moderneastculture.model.Repository;

import static com.folva.moderneastculture.model.Repository.instance;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Group;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.layout.Region;

import javafx.stage.Stage;

import javafx.util.Pair;

import org.apache.logging.log4j.Level;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileNotFoundException;

import java.io.InputStream;

import java.net.URL;

import java.util.\*;

/\*\*

\* Головний клас з точкою входу в додаток, головними вспливаючими

\* вікнами та деякими методами-утилітами

\*/

public class Main extends Application {

private static final Logger logger = LogManager.getLogger(Main.class.getName());

private static final int WINDOW\_WIDTH = 800;

private static final int WINDOW\_HEIGHT = 490;

/\*\*

\* Вспливаюче вікно "підтвердження" або ж якоїсь інформації

\*/

public static final Alert confirmationAlert;

/\*\*

\* Вспливаюче вікно попереджень або якихось малих помилок

\*/

public static final Alert warningAlert;

/\*\*

\* Вспливаюче вікно середніх та великих помилок

\*/

public static final Alert errorAlert;

/\*\*

\* Головне (і єдине) вікно, на якому і висить весь інтерфейс програми

\*/

public static Stage stage;

/\*\*

\* Колекція пар "назва меню-пара{об'єкт Parent меню-контроллер меню}"

\*/

public static Map<String, Pair<Parent, Object>> formMap = new HashMap<>();

static {

try {

Class.forName("com.folva.moderneastculture.model.Repository");

} catch (ClassNotFoundException e) {

logger.fatal("Repository class hasn't been found", e);

}

confirmationAlert = new Alert(Alert.AlertType.CONFIRMATION);

confirmationAlert.getButtonTypes().remove(1);

confirmationAlert.getDialogPane().setMinHeight(Region.USE\_PREF\_SIZE);

warningAlert = new Alert(Alert.AlertType.WARNING);

warningAlert.getDialogPane().setMinHeight(Region.USE\_PREF\_SIZE);

errorAlert = new Alert(Alert.AlertType.ERROR);

errorAlert.getDialogPane().setMinHeight(Region.USE\_PREF\_SIZE);

}

/\*\*

\* Головна точка входу програми

\* @param args аргументи командного рядку

\*/

public static void main(String[] args) {

launch(args);

}

@Override

public void start(Stage primaryStage) {

Repository.instance.connectToDatabase();

getParameters().getRaw().forEach(System.out::println);

stage = primaryStage;

runMenuForm();

}

@Override

public void stop() throws Exception {

Repository.instance.disconnectFromDatabase();

super.stop();

System.exit(0);

}

/\*\*

\* @param formName Назва меню, яке потрібно отримати

\* @return об'єкт Parent, який містить в собі все меню, за яким буде закріплений

\* його відповідний контроллер, якщо він є, і null, якщо нема

\*/

public static Parent getForm(String formName) {

Parent root = new Group();

try {

if (formMap.containsKey(formName)) {

root = formMap.get(formName).getKey();

} else {

FXMLLoader loader = new FXMLLoader(getResource("/res/fxml\_views/" + formName + ".fxml"), Repository.namesBundle);

root = loader.load();

formMap.put(formName, new Pair<>(root, loader.getController()));

}

} catch (Exception e) {

logger.error("Error while opening " + formName + ": ", e);

Main.errorAlert.setContentText(instance.getNamesBundleValue("problemOccurred"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

return root;

}

/\*\*

\* @param formName Назва меню, контроллер якого потрібно отримати

\* @return об'єкт контроллеру меню, якщо він є, і null, якщо нема

\*/

public static Object getControllerForForm(String formName) {

Object controller = new Object();

if (formMap.containsKey(formName)) {

controller = formMap.get(formName).getValue();

}

return controller;

}

/\*\*

\* @param resourceName Ім'я ресурсу

\* @return шлях до ресурсу у вигляді URL-об'єкту, якщо він є, і null, якщо нема

\*/

public static URL getResource(String resourceName) {

return Main.class.getResource(resourceName);

}

/\*\*

\* @param file Файл, входячий поток якого потрібно отримати

\* @return входячий поток відповідного файла, якщо він є, і null, якщо нема.

\* При помилці додаток закривається, перед цим виводячи повідомлення про помилку.

\*/

public static InputStream getFileStream(File file) {

try {

return new FileInputStream(file);

} catch (FileNotFoundException e) {

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("problemOccurred"));

Main.errorAlert.showAndWait();

System.exit(-1);

return null;

}

}

private void runMenuForm() {

try {

FXMLLoader loader = new FXMLLoader(getResource("/res/fxml\_views/MenuForm.fxml"), Repository.namesBundle);

Parent menuRoot = loader.load();

Scene menuScene = new Scene(menuRoot, WINDOW\_WIDTH, WINDOW\_HEIGHT);

formMap.put("MenuForm", new Pair<>(menuRoot, loader.getController()));

stage.setTitle(instance.getNamesBundleValue("title"));

stage.setResizable(false);

stage.setScene(menuScene);

stage.show();

} catch (Exception e) {

logger.log(Level.ERROR, "Error while loading menu window", e);

Main.errorAlert.setContentText(instance.getNamesBundleValue("problemOccurred"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

}

}

# Package com.folva.moderneastculture.model

## Repository

package com.folva.moderneastculture.model;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.dto.\*;

import javafx.application.Application;

import javafx.beans.property.SimpleBooleanProperty;

import javafx.scene.image.Image;

import javafx.util.Pair;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import java.io.File;

import java.io.IOException;

import java.io.InputStream;

import java.io.InputStreamReader;

import java.net.URL;

import java.net.URLConnection;

import java.nio.charset.StandardCharsets;

import java.nio.file.\*;

import java.sql.\*;

import java.time.Duration;

import java.util.\*;

import java.util.prefs.Preferences;

import java.util.stream.Collectors;

import java.util.stream.Stream;

/\*\*

\* Сховище, яке містить всю головну інформацію та можливості роботи з базою даних. Також містить деякі

\* статичні методи-утиліти

\*/

@SuppressWarnings({"RedundantIfStatement", "DuplicatedCode"})

public class Repository {

private static final Logger logger = LogManager.getLogger(Repository.class);

private static final Preferences prefs = Preferences.userRoot().node("com/folva/moderneastculture/config");

private static final String PREFS\_ADMIN\_LOGIN = "superadmin\_login";

private static final String PREFS\_ADMIN\_PASSWORD = "superadmin\_password";

private static final String DEFAULT\_ADMIN\_LOGIN = "somebody@oncetold.me";

private static final String DEFAULT\_ADMIN\_PASSWORD = "simplePassword";

private static final long MAX\_CACHED\_TIME;

/\*\*

\* Екземпляр "сховища", захищений від породження копій

\*/

public static final Repository instance;

/\*\*

\* Рік виходу першого аніме

\*/

public static final int FIRST\_ANIME\_PREMIERE\_YEAR = 1958;

/\*\*

\* Рік виходу першого коміксу-манги

\*/

public static final int FIRST\_MANGA\_PREMIERE\_YEAR = 1900;

/\*\*

\* Об'єкт з ресурсами до магатомовного інтерфейсу - в залежності від працюючої локалі або вказаних

\* параметрів (в командній строці) щодо локалі буде використовуватися та чи інша мова в інтерфейсі.

\* Наразі доступні - українська, російська (по замовчуванню) та англійська

\*/

public static final ResourceBundle namesBundle;

/\*\*

\* Властивість щодо того, чи авторизувався адміністратор зараз.

\* Є можливість підписатися на нього

\*/

public static final SimpleBooleanProperty adminIsAuthorizedProperty;

/\*\*

\* Шлях до каталогу з зображеннями бази даних

\*/

public static final String DB\_IMAGES\_FOLDER = "db\_img";

private static String ADMIN\_LOGIN;

private static String ADMIN\_PASSWORD;

private final DbRepository dbRepository;

private Pair<ArrayList<Genre>, Long> cachedGenres;

private Pair<ArrayList<Author>, Long> cachedAuthors;

private Pair<ArrayList<Anime>, Long> cachedAnime;

private Pair<ArrayList<Comics>, Long> cachedComics;

private Pair<ArrayList<Pair<Anime, Genre>>, Long> cachedAnimeGenreMap;

private Pair<ArrayList<Pair<Integer, String>>, Long> cachedAnimeImagePaths;

private Pair<ArrayList<Pair<Comics, Genre>>, Long> cachedComicsGenreMap;

private Pair<ArrayList<Pair<Integer, String>>, Long> cachedComicsImagePaths;

static {

instance = new Repository();

ADMIN\_LOGIN = prefs.get(PREFS\_ADMIN\_LOGIN, DEFAULT\_ADMIN\_LOGIN);

ADMIN\_PASSWORD = prefs.get(PREFS\_ADMIN\_PASSWORD, DEFAULT\_ADMIN\_PASSWORD);

if (ADMIN\_LOGIN.equalsIgnoreCase(DEFAULT\_ADMIN\_LOGIN))

prefs.put(PREFS\_ADMIN\_LOGIN, DEFAULT\_ADMIN\_LOGIN);

if (ADMIN\_PASSWORD.equalsIgnoreCase(DEFAULT\_ADMIN\_PASSWORD))

prefs.put(PREFS\_ADMIN\_PASSWORD, DEFAULT\_ADMIN\_PASSWORD);

namesBundle = ResourceBundle.getBundle("ui\_names", new ResourceBundle.Control() {

@Override

public ResourceBundle newBundle

(String baseName, Locale locale, String format, ClassLoader loader, boolean reload)

throws IOException {

String bundleName = toBundleName(baseName, locale);

String resourceName = toResourceName(bundleName, "properties");

ResourceBundle bundle = null;

InputStream stream = null;

if (reload) {

URL url = loader.getResource(resourceName);

if (url != null) {

URLConnection connection = url.openConnection();

if (connection != null) {

connection.setUseCaches(false);

stream = connection.getInputStream();

}

}

} else {

stream = loader.getResourceAsStream(resourceName);

}

if (stream != null) {

try {

// Only this line is changed to make it to read properties files as UTF-8.

bundle = new PropertyResourceBundle(new InputStreamReader(stream, StandardCharsets.UTF\_8));

} finally {

stream.close();

}

}

return bundle;

}

});

adminIsAuthorizedProperty = new SimpleBooleanProperty(false);

MAX\_CACHED\_TIME = Duration.ofMinutes(15).toMillis();

}

private Repository() {

dbRepository = new DbRepository();

}

/\*\*

\* Метод під'єднання до бази даних. Потрібно виконувати його перед

\* роботою з базою даних, інакше робота з базою даних буде

\* неможлива

\*/

public void connectToDatabase() {

dbRepository.connect();

}

/\*\*

\* Метод від'єднання від бази даних. Потрібно виконувати після роботи

\* з базою даних

\*/

public void disconnectFromDatabase() {

dbRepository.disconnect();

}

/\*\*

\* @param login строка логіну для перевірки

\* @return true, якщо логін корректний (враховуючи регістр). Інакше false

\*/

public boolean loginIsCorrect(String login) {

if (ADMIN\_LOGIN.equals(login)) return true;

return false;

}

/\*\*

\* @param password строка паролю для перевірки

\* @return true, якщо пароль корректний (враховуючи регістр). Інакше false

\*/

public boolean passwordIsCorrect(String password) {

if (ADMIN\_PASSWORD.equals(password)) return true;

return false;

}

/\*\*

\* Оновлює логін адміністратору

\* @param newLogin новий логін. Перед оновленням проходить валідацію та в разі

\* її непроходження НЕ встановлюється

\*/

public void updateAdminLogin(String newLogin) {

if (loginIsValid(newLogin)) {

ADMIN\_LOGIN = newLogin;

prefs.put(PREFS\_ADMIN\_LOGIN, ADMIN\_LOGIN);

}

}

/\*\*

\* Оновлює пароль адміністратору

\* @param newPassword новий пароль. Перед оновленням проходить валідацію та в разі

\* її непроходження НЕ встановлюється

\*/

public void updateAdminPassword(String newPassword) {

if (passwordIsValid(newPassword)) {

ADMIN\_PASSWORD = newPassword;

prefs.put(PREFS\_ADMIN\_PASSWORD, ADMIN\_PASSWORD);

}

}

/\*\*

\* @param login строка логіну для валідації

\* @return true, якщо логін можна встановити як новий. Інакше false

\*/

public boolean loginIsValid(String login) {

if (login == null || !login.matches("[a-zA-Z0-9@.]{3,20}")) return false;

return true;

}

/\*\*

\* @param password строка паролю для валідації

\* @return true, якщо пароль можна встановити як новий. Інакше false

\*/

public boolean passwordIsValid(String password) {

if (password == null || !password.matches("[a-zA-Z0-9]{3,20}")) return false;

return true;

}

/\*\*

\* @param key "ключ" до значення, яке потрібно отримати в поточній локалі

\* ресурсів. Наприклад, якщо значення <code>key</code> буде

\* <code>title</code>, то в залежності від локалі повернеться

\* титулка або на російській, або на українській, або на

\* англійській мові (або більше, якщо буде добавлено більше

\* файлів ресурсів)

\* @return значення ключа в залежності від локалізації

\*/

public String getNamesBundleValue(String key) {

if (key == null) return "";

return namesBundle.getString(key);

}

/\*\*

\* @param sourceFile Файл, який потрібно скопіювати у каталог зображень

\* бази даних.

\* @return об'єкт шляху до нового файлу, якщо він був скопійований, або

\* null, якщо ні

\*/

public Path imageWasCopiedToDbImgFolder(File sourceFile) {

Path destinationFilePath = null;

String extension = getFileExtension(sourceFile);

String randomName = String.valueOf(System.currentTimeMillis())

.substring(0, 10).concat(".").concat(extension);

File destinationFile = Paths.get(DB\_IMAGES\_FOLDER, randomName).toFile();

try {

handleDirectoryExistence();

destinationFilePath = Files.copy(sourceFile.toPath(), destinationFile.toPath(),

StandardCopyOption.REPLACE\_EXISTING);

} catch (IOException e) {

logger.error("Error while copying image file: ", e);

Main.warningAlert.setContentText(Repository.instance.getNamesBundleValue("imageCopyError"));

Main.warningAlert.show();

}

return destinationFilePath;

}

/\*\*

\* Забезпечує існування папки {@value #DB\_IMAGES\_FOLDER} - якщо її нема,

\* то створює її. Викликається перед копіюванням файлу до папки

\*/

public void handleDirectoryExistence() {

File directory = Paths.get(DB\_IMAGES\_FOLDER).toFile();

if (!directory.exists()) {

try {

Files.createDirectory(directory.toPath());

} catch (IOException e) {

logger.fatal("Error while creating db\_img folder: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("imageCopyError"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

}

}

/\*\*

\* @param updateCacheAndGetNew true, якщо потрібно отримати найновіші

\* дані, та false, якщо можливо отримати

\* дані, які були актуальні максимум 15

\* хвилин тому

\* @return колекцію жанрів

\*/

public ArrayList<Genre> getGenres(boolean updateCacheAndGetNew) {

long currentTimeMillis = System.currentTimeMillis();

if (cachedGenres == null || updateCacheAndGetNew ||

currentTimeMillis - cachedGenres.getValue() > MAX\_CACHED\_TIME) {

cachedGenres = new Pair<>(dbRepository.getGenres(), currentTimeMillis);

}

return cachedGenres.getKey();

}

/\*\*

\* @param updateCacheAndGetNew true, якщо потрібно отримати найновіші

\* дані, та false, якщо можливо отримати

\* дані, які були актуальні максимум 15

\* хвилин тому

\* @return колекцію авторів

\*/

public ArrayList<Author> getAuthors(boolean updateCacheAndGetNew) {

long currentTimeMillis = System.currentTimeMillis();

if (cachedAuthors == null || updateCacheAndGetNew ||

currentTimeMillis - cachedAuthors.getValue() > MAX\_CACHED\_TIME) {

cachedAuthors = new Pair<>(dbRepository.getAuthors(), currentTimeMillis);

}

return cachedAuthors.getKey();

}

/\*\*

\* @param updateCacheAndGetNew true, якщо потрібно отримати найновіші

\* дані, та false, якщо можливо отримати

\* дані, які були актуальні максимум 15

\* хвилин тому

\* @return колекцію аніме

\*/

public ArrayList<Anime> getAnimes(boolean updateCacheAndGetNew) {

long currentTimeMillis = System.currentTimeMillis();

if (cachedAnime == null || updateCacheAndGetNew ||

currentTimeMillis - cachedAnime.getValue() > MAX\_CACHED\_TIME) {

cachedAnime = new Pair<>(dbRepository.getAnimes(), currentTimeMillis);

}

return cachedAnime.getKey();

}

/\*\*

\* @param updateCacheAndGetNew true, якщо потрібно отримати найновіші

\* дані, та false, якщо можливо отримати

\* дані, які були актуальні максимум 15

\* хвилин тому

\* @return колекцію коміксів

\*/

public ArrayList<Comics> getComics(boolean updateCacheAndGetNew) {

long currentTimeMillis = System.currentTimeMillis();

if (cachedComics == null || updateCacheAndGetNew ||

currentTimeMillis - cachedComics.getValue() > MAX\_CACHED\_TIME) {

cachedComics = new Pair<>(dbRepository.getComics(), currentTimeMillis);

}

return cachedComics.getKey();

}

/\*\*

\* @param updateCacheAndGetNew true, якщо потрібно отримати найновіші

\* дані, та false, якщо можливо отримати

\* дані, які були актуальні максимум 15

\* хвилин тому

\* @return колекцію пар "аніме-жанр". Зв'язок від багатьох до багатьох

\*/

public ArrayList<Pair<Anime, Genre>> getAnimeGenreMap(boolean updateCacheAndGetNew) {

long currentTimeMillis = System.currentTimeMillis();

if (cachedAnimeGenreMap == null || updateCacheAndGetNew ||

currentTimeMillis - cachedAnimeGenreMap.getValue() > MAX\_CACHED\_TIME) {

cachedAnimeGenreMap = new Pair<>(dbRepository.getAnimeGenreMap(), currentTimeMillis);

}

return cachedAnimeGenreMap.getKey();

}

/\*\*

\* @param updateCacheAndGetNew true, якщо потрібно отримати найновіші

\* дані, та false, якщо можливо отримати

\* дані, які були актуальні максимум 15

\* хвилин тому

\* @return колекцію пар "комікс-жанр". Зв'язок від багатьох до багатьох

\*/

public ArrayList<Pair<Comics, Genre>> getComicsGenreMap(boolean updateCacheAndGetNew) {

long currentTimeMillis = System.currentTimeMillis();

if (cachedComicsGenreMap == null || updateCacheAndGetNew ||

currentTimeMillis - cachedComicsGenreMap.getValue() > MAX\_CACHED\_TIME) {

cachedComicsGenreMap = new Pair<>(dbRepository.getComicsGenreMap(), currentTimeMillis);

}

return cachedComicsGenreMap.getKey();

}

/\*\*

\* @param updateCacheAndGetNew true, якщо потрібно отримати найновіші

\* дані, та false, якщо можливо отримати

\* дані, які були актуальні максимум 15

\* хвилин тому

\* @return колекцію пар "індекс аніме-назва зображення". Зв'язок від

\* багатьох до багатьох

\*/

public ArrayList<Pair<Integer, String>> getAnimeImagePaths(boolean updateCacheAndGetNew) {

long currentTimeMillis = System.currentTimeMillis();

if (cachedAnimeImagePaths == null || updateCacheAndGetNew ||

currentTimeMillis - cachedAnimeImagePaths.getValue() > MAX\_CACHED\_TIME) {

cachedAnimeImagePaths = new Pair<>(dbRepository.getAnimeImagePaths(), currentTimeMillis);

}

return cachedAnimeImagePaths.getKey();

}

/\*\*

\* @param updateCacheAndGetNew true, якщо потрібно отримати найновіші

\* дані, та false, якщо можливо отримати

\* дані, які були актуальні максимум 15

\* хвилин тому

\* @return колекцію пар "індекс коміксу-назва зображення". Зв'язок від

\* багатьох до багатьох

\*/

public ArrayList<Pair<Integer, String>> getComicsImagePaths(boolean updateCacheAndGetNew) {

long currentTimeMillis = System.currentTimeMillis();

if (cachedComicsImagePaths == null || updateCacheAndGetNew ||

currentTimeMillis - cachedComicsImagePaths.getValue() > MAX\_CACHED\_TIME) {

cachedComicsImagePaths = new Pair<>(dbRepository.getComicsImagePaths(), currentTimeMillis);

}

return cachedComicsImagePaths.getKey();

}

/\*\*

\* Метод-утиліта для оновнення кешу всього аніме підрозділу

\*/

public void updateAnimeCache() {

getGenres(true);

getAnimes(true);

getAnimeGenreMap(true);

getAnimeImagePaths(true);

}

/\*\*

\* Добавляє нове аніме до бази даних або, в разі невдачі, виводить помилку

\* користувачу

\* @param newAnime нове аніме для добавлення

\*/

public void insertNewAnime(Anime newAnime) {

dbRepository.insertNewAnime(newAnime);

}

/\*\*

\* Оновлює аніме в базі даних або, в разі невдачі, виводить помилку користувачу

\* @param anime комікс для оновлення

\*/

public void updateAnime(Anime anime) {

dbRepository.updateAnime(anime);

}

/\*\*

\* Видаляє старе аніме з бази даних або, в разі невдачі, виводить помилку

\* користувачу

\* @param oldAnime старе аніме для видалення

\*/

public void deleteAnimeFromDb(Anime oldAnime) {

dbRepository.deleteAnime(oldAnime);

}

/\*\*

\* Оновлює жанри щодо вказанного аніме

\* @param animeGenres нові жанри до вказаного аніме

\*/

public void updateAnimeGenres(Pair<Anime, ArrayList<Genre>> animeGenres) {

dbRepository.updateAnimeGenres(animeGenres);

}

/\*\*

\* Оновлює шляхи зображень щодо вказанного аніме

\* @param animeImages нові імена зображень до вказаного аніме

\*/

public void updateAnimeImages(Pair<Anime, ArrayList<String>> animeImages) {

dbRepository.updateAnimeImages(animeImages);

}

/\*\*

\* Метод-утиліта для оновнення кешу всього коміксового підрозділу

\*/

public void updateComicsCache() {

getGenres(true);

getComics(true);

getComicsGenreMap(true);

getComicsImagePaths(true);

}

/\*\*

\* Добавляє новий комікс до бази даних або, в разі невдачі, виводить помилку

\* користувачу

\* @param newComics новий комікс для добавлення

\*/

public void insertNewComics(Comics newComics) {

dbRepository.insertNewComics(newComics);

}

/\*\*

\* Оновлює комікс в базі даних або, в разі невдачі, виводить помилку користувачу

\* @param comics комікс для оновлення

\*/

public void updateComics(Comics comics) {

dbRepository.updateComics(comics);

}

/\*\*

\* Видаляє старий комікс з бази даних або, в разі невдачі, виводить помилку

\* користувачу

\* @param oldComics старий комікс для видалення

\*/

public void deleteComicsFromDb(Comics oldComics) {

dbRepository.deleteComics(oldComics);

}

/\*\*

\* Оновлює жанри щодо вказанного коміксу

\* @param comicsGenres нові жанри до вказаного коміксу

\*/

public void updateComicsGenres(Pair<Comics, ArrayList<Genre>> comicsGenres) {

dbRepository.updateComicsGenres(comicsGenres);

}

/\*\*

\* Оновлює шляхи зображень щодо вказанного коміксу

\* @param comicsImages нові імена зображень до вказаного коміксу

\*/

public void updateComicsImages(Pair<Comics, ArrayList<String>> comicsImages) {

dbRepository.updateComicsImages(comicsImages);

}

/\*\*

\* @param file файл, розширення якого потрібно отримати

\* @return розширення файлу. Наприклад, з файлу "somebody.txt"

\* поверне строку "txt", а з файлу "heh" вилучить пусту строку

\*/

public static String getFileExtension(File file) {

String fileName = file.getName();

if (fileName.lastIndexOf(".") != -1 && fileName.lastIndexOf(".") != 0)

return fileName.substring(fileName.lastIndexOf(".") + 1);

else return "";

}

/\*\*

\* @param imageName назва зображення каталогу бази даних, яке

\* потрібно завантажати в об'єкт Image

\* @return об'єкт Image вказаного зображення або null, якщо вказаного

\* зображення нема або не вдалося вилучити

\*/

public static Image loadImage(String imageName) {

Path imagePath = Paths.get(DB\_IMAGES\_FOLDER, imageName);

if (Files.notExists(imagePath)) return null;

InputStream fileStream = Main.getFileStream(imagePath.toFile());

Image image = new Image(fileStream);

try {

fileStream.close();

} catch (Exception e) {

logger.error("Error closing image file stream: ", e);

}

return image;

}

/\*\*

\* @return зображення з текстом "NO IMAGE FOUND". Потрібне

\* зазвичай для випадків, коли у аніме/коміксу нема навіть

\* одного зображення, щоб поставити на "титульне" віконце

\* зображення

\*/

public static Image getNoImageImage() {

URL notFoundImageResource = Main.getResource("/res/img/no\_image.jpg");

return new Image(notFoundImageResource.toString());

}

}

## Repisotory.DbRepository

private static final class DbRepository {

private static final String CONNECTION\_URL = "jdbc:sqlite:modern\_east\_culture.db3";

private enum GenreFields {

ID("id", 1),

NAME("name", 2);

public final String columnName;

public final int columnIndex;

GenreFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum AuthorFields {

ID("id", 1),

TYPE("type", 2),

NAME("name", 3);

public final String columnName;

public final int columnIndex;

AuthorFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum AgeRatingFields {

ID("id", 1),

NAME("name", 2),

DESCRIPTION("description", 3);

public final String columnName;

public final int columnIndex;

AgeRatingFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum AnimeFields {

ID("id", 1),

ID\_AUTHOR("id\_author", 2),

TYPE("type", 3),

NAME("name", 4),

DESCRIPTION("description", 5),

EPISODE\_COUNT("episode\_count", 6),

SOURCE("source", 7),

ID\_RATING("id\_rating", 8),

PREMIERE\_YEAR("premiere\_year", 9),

PREMIERE\_SEASON("premiere\_season", 10),

STATUS("status", 11);

public final String columnName;

public final int columnIndex;

AnimeFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum ComicsFields {

ID("id", 1),

ID\_AUTHOR("id\_author", 2),

TYPE("type", 3),

NAME("name", 4),

DESCRIPTION("description", 5),

CHAPTER\_COUNT("chapter\_count", 6),

SOURCE("source", 7),

PREMIERE\_YEAR("premiere\_year", 8),

STATUS("status", 9);

public final String columnName;

public final int columnIndex;

ComicsFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum AnimeGenreFields {

ID\_ANIME("id\_anime", 1),

ID\_GENRE("id\_genre", 2);

public final String columnName;

public final int columnIndex;

AnimeGenreFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum AnimeAltNameFields {

ID\_ANIME("id\_anime", 1),

ALTERNATIVE\_NAME("alternative\_name", 2);

public final String columnName;

public final int columnIndex;

AnimeAltNameFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum AnimeGalleryFields {

ID\_ANIME("id\_anime", 1),

IMG\_PATH("img\_path", 2);

public final String columnName;

public final int columnIndex;

AnimeGalleryFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum ComicsGenreFields {

ID\_COMICS("id\_comics", 1),

ID\_GENRE("id\_genre", 2);

public final String columnName;

public final int columnIndex;

ComicsGenreFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum ComicsAltNameFields {

ID\_COMICS("id\_comics", 1),

ALTERNATIVE\_NAME("alternative\_name", 2);

public final String columnName;

public final int columnIndex;

ComicsAltNameFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

private enum ComicsGalleryFields {

ID\_COMICS("id\_comics", 1),

IMG\_PATH("img\_path", 2);

public final String columnName;

public final int columnIndex;

ComicsGalleryFields(String columnName, int columnIndex) {

this.columnName = columnName;

this.columnIndex = columnIndex;

}

@Override

public String toString() {

return columnName;

}

}

public Connection connection;

/\*

\* Connection/disconnection

\*/

public void connect() {

try {

connection = DriverManager.getConnection(CONNECTION\_URL);

logger.info("Database connection is successful.");

} catch (SQLException e) {

logger.error("Database connection was refused: ", e);

}

}

public void disconnect() {

try {

if (connection != null)

connection.close();

logger.info("Database was successfully disconnected.");

} catch (SQLException e) {

logger.error("Database wasn't disconnected: ", e);

}

}

/\*

\* Methods which return the output of "select" statements

\*/

public ArrayList<Anime> getAnimes() {

ArrayList<Anime> animeList = new ArrayList<>();

ArrayList<Author> authors = getAuthors();

ArrayList<AgeRating> ageRatings = getAgeRatings();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from anime");

while (resultSet.next()) {

int animeId = resultSet.getInt(AnimeFields.ID.columnName);

int animeAuthorId = resultSet.getInt(AnimeFields.ID\_AUTHOR.columnName);

Stream<Author> authorStream = authors.stream().filter(

author -> author.getId() == animeAuthorId);

Optional<Author> optionalAuthor = authorStream.findFirst();

Author animeAuthor = optionalAuthor.orElse(null);

String animeTypeString = resultSet.getString(AnimeFields.TYPE.columnName);

Anime.Type animeType = Anime.Type.valueOf(animeTypeString.toUpperCase());

String animeName = resultSet.getString(AnimeFields.NAME.columnName);

String animeDescription = resultSet.getString(AnimeFields.DESCRIPTION.columnName);

int animeEpisodeCount = resultSet.getInt(AnimeFields.EPISODE\_COUNT.columnName);

String animeSourceString = resultSet.getString(AnimeFields.SOURCE.columnName);

Anime.Source animeSource = Anime.Source.valueOf(animeSourceString.toUpperCase());

int animeAgeRatingId = resultSet.getInt(AnimeFields.ID\_RATING.columnName);

Stream<AgeRating> ageRatingStream = ageRatings.stream().filter(

ageRating -> ageRating.id == animeAgeRatingId);

Optional<AgeRating> optionalAgeRating = ageRatingStream.findFirst();

AgeRating animeAgeRating = optionalAgeRating.orElse(null);

int animePremiereYear = resultSet.getInt(AnimeFields.PREMIERE\_YEAR.columnName);

String animePremiereSeasonString = resultSet.getString(AnimeFields.PREMIERE\_SEASON.columnName);

YearSeason animePremiereSeason = YearSeason.valueOf(animePremiereSeasonString.toUpperCase());

String animeStatusString = resultSet.getString(AnimeFields.STATUS.columnName);

Status animeStatus = Status.valueOf(animeStatusString.toUpperCase());

Anime newAnime = Anime.Builder.newBuilder()

.setId(animeId)

.setAuthor(animeAuthor)

.setType(animeType)

.setName(animeName)

.setDescription(animeDescription)

.setEpisodeCount(animeEpisodeCount)

.setSource(animeSource)

.setAgeRating(animeAgeRating)

.setPremiereYear(animePremiereYear)

.setPremiereSeason(animePremiereSeason)

.setStatus(animeStatus)

.build();

animeList.add(newAnime);

}

resultSet.close();

} catch (SQLException | IllegalArgumentException e) {

logger.error("Error while getting anime from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

setAnimeAltNames(animeList);

return animeList;

}

private void setAnimeAltNames(ArrayList<Anime> animeList) {

ArrayList<Pair<Integer, String>> allAltNames = new ArrayList<>();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from anime\_alt\_names");

while (resultSet.next()) {

int idAnime = resultSet.getInt(AnimeAltNameFields.ID\_ANIME.columnName);

String altName = resultSet.getString(AnimeAltNameFields.ALTERNATIVE\_NAME.columnName);

Pair<Integer, String> newAltNameObj = new Pair<>(idAnime, altName);

allAltNames.add(newAltNameObj);

}

resultSet.close();

} catch (SQLException e) {

logger.error("Error while getting anime alt names from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

animeList.forEach(anime -> allAltNames.forEach(altName -> {

if (anime.getId() == altName.getKey())

anime.getAltNames().add(altName.getValue());

}));

}

public ArrayList<Comics> getComics() {

ArrayList<Comics> comicsList = new ArrayList<>();

ArrayList<Author> authors = getAuthors();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from comics");

while (resultSet.next()) {

int comicsId = resultSet.getInt(ComicsFields.ID.columnName);

int comicsAuthorId = resultSet.getInt(ComicsFields.ID\_AUTHOR.columnName);

Stream<Author> authorStream = authors.stream().filter(

author -> author.getId() == comicsAuthorId);

Optional<Author> optionalAuthor = authorStream.findFirst();

Author comicsAuthor = optionalAuthor.orElse(null);

String comicsTypeString = resultSet.getString(ComicsFields.TYPE.columnName);

Comics.Type comicsType = Comics.Type.valueOf(comicsTypeString.toUpperCase());

String comicsName = resultSet.getString(ComicsFields.NAME.columnName);

String comicsDescription = resultSet.getString(ComicsFields.DESCRIPTION.columnName);

int comicsChapterCount = resultSet.getInt(ComicsFields.CHAPTER\_COUNT.columnName);

String comicsSourceString = resultSet.getString(ComicsFields.SOURCE.columnName);

Comics.Source comicsSource = Comics.Source.valueOf(comicsSourceString.toUpperCase());

int comicsPremiereYear = resultSet.getInt(ComicsFields.PREMIERE\_YEAR.columnName);

String comicsStatusString = resultSet.getString(ComicsFields.STATUS.columnName);

Status comicsStatus = Status.valueOf(comicsStatusString.toUpperCase());

Comics newAnime = Comics.Builder.newBuilder()

.setId(comicsId)

.setAuthor(comicsAuthor)

.setType(comicsType)

.setName(comicsName)

.setDescription(comicsDescription)

.setChapterCount(comicsChapterCount)

.setSource(comicsSource)

.setPremiereYear(comicsPremiereYear)

.setStatus(comicsStatus)

.build();

comicsList.add(newAnime);

}

resultSet.close();

} catch (SQLException | IllegalArgumentException e) {

logger.error("Error while getting comics from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

setComicsAltNames(comicsList);

return comicsList;

}

private void setComicsAltNames(ArrayList<Comics> comicsList) {

ArrayList<Pair<Integer, String>> allAltNames = new ArrayList<>();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from comics\_alt\_names");

while (resultSet.next()) {

int idComics = resultSet.getInt(ComicsAltNameFields.ID\_COMICS.columnName);

String altName = resultSet.getString(ComicsAltNameFields.ALTERNATIVE\_NAME.columnName);

Pair<Integer, String> newAltNameObj = new Pair<>(idComics, altName);

allAltNames.add(newAltNameObj);

}

resultSet.close();

} catch (SQLException e) {

logger.error("Error while getting comics alt names from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

comicsList.forEach(comics -> allAltNames.forEach(altName -> {

if (comics.getId() == altName.getKey())

comics.getAltNames().add(altName.getValue());

}));

}

public ArrayList<Author> getAuthors() {

ArrayList<Author> authors = new ArrayList<>();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from authors");

while (resultSet.next()) {

int authorId = resultSet.getInt(AuthorFields.ID.columnName);

String authorType = resultSet.getString(AuthorFields.TYPE.columnName);

String authorName = resultSet.getString(AuthorFields.NAME.columnName);

Author newAuthor = Author.Builder.newBuilder()

.setId(authorId)

.setType(Author.Type.valueOf(authorType.toUpperCase()))

.setName(authorName)

.build();

authors.add(newAuthor);

}

resultSet.close();

} catch (SQLException | IllegalArgumentException e) {

logger.error("Error while getting authors from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

return authors;

}

public ArrayList<AgeRating> getAgeRatings() {

ArrayList<AgeRating> ageRatings = new ArrayList<>();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from age\_ratings");

while (resultSet.next()) {

int ageRatingId = resultSet.getInt(AgeRatingFields.ID.columnName);

AgeRating ageRating = AgeRating.valueOfId(ageRatingId);

ageRatings.add(ageRating);

}

resultSet.close();

} catch (SQLException e) {

logger.error("Error while getting age ratings from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

return ageRatings;

}

public ArrayList<Genre> getGenres() {

ArrayList<Genre> genres = new ArrayList<>();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from genres");

while (resultSet.next()) {

int genreId = resultSet.getInt(GenreFields.ID.columnName);

String genreName = resultSet.getString(GenreFields.NAME.columnName);

Genre newGenre = new Genre(genreId, genreName);

genres.add(newGenre);

}

resultSet.close();

} catch (SQLException e) {

logger.error("Error while getting genres from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

return genres;

}

public ArrayList<Pair<Anime, Genre>> getAnimeGenreMap() {

ArrayList<Pair<Anime, Genre>> animeGenres = new ArrayList<>();

ArrayList<Anime> allAnimes = getAnimes();

ArrayList<Genre> allGenres = getGenres();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from anime\_genres");

while (resultSet.next()) {

int idAnime = resultSet.getInt(AnimeGenreFields.ID\_ANIME.columnName);

int idGenre = resultSet.getInt(AnimeGenreFields.ID\_GENRE.columnName);

Optional<Anime> optionalAnime = allAnimes.stream().filter(anime ->

anime.getId() == idAnime).findFirst();

Optional<Genre> optionalGenre = allGenres.stream().filter(genre ->

genre.id == idGenre).findFirst();

if (!optionalAnime.isPresent() || !optionalGenre.isPresent()) continue;

Anime anime = optionalAnime.get();

Genre genre = optionalGenre.get();

Pair<Anime, Genre> pair = new Pair<>(anime, genre);

animeGenres.add(pair);

}

resultSet.close();

} catch (SQLException e) {

logger.error("Error while getting anime genre connections: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

return animeGenres;

}

public ArrayList<Pair<Integer, String>> getAnimeImagePaths() {

ArrayList<Pair<Integer, String>> animeImagePaths = new ArrayList<>();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from anime\_gallery");

while (resultSet.next()) {

int animeId = resultSet.getInt(AnimeGalleryFields.ID\_ANIME.columnName);

String imgPath = resultSet.getString(AnimeGalleryFields.IMG\_PATH.columnName);

Pair<Integer, String> newPair = new Pair<>(animeId, imgPath);

animeImagePaths.add(newPair);

}

resultSet.close();

} catch (SQLException e) {

logger.error("Error while getting anime image paths from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

return animeImagePaths;

}

public ArrayList<Pair<Comics, Genre>> getComicsGenreMap() {

ArrayList<Pair<Comics, Genre>> comicsGenres = new ArrayList<>();

ArrayList<Comics> allComics = getComics();

ArrayList<Genre> allGenres = getGenres();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from comics\_genres");

while (resultSet.next()) {

int idComics = resultSet.getInt(ComicsGenreFields.ID\_COMICS.columnName);

int idGenre = resultSet.getInt(ComicsGenreFields.ID\_GENRE.columnName);

Optional<Comics> optionalComics = allComics.stream().filter(comics ->

comics.getId() == idComics).findFirst();

Optional<Genre> optionalGenre = allGenres.stream().filter(genre ->

genre.id == idGenre).findFirst();

if (!optionalComics.isPresent() || !optionalGenre.isPresent()) continue;

Comics comics = optionalComics.get();

Genre genre = optionalGenre.get();

Pair<Comics, Genre> pair = new Pair<>(comics, genre);

comicsGenres.add(pair);

}

resultSet.close();

} catch (SQLException e) {

logger.error("Error while getting comics genre connections: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

return comicsGenres;

}

public ArrayList<Pair<Integer, String>> getComicsImagePaths() {

ArrayList<Pair<Integer, String>> comicsImagePaths = new ArrayList<>();

try {

ResultSet resultSet = connection.createStatement().executeQuery("select \* from comics\_gallery");

while (resultSet.next()) {

int comicsId = resultSet.getInt(ComicsGalleryFields.ID\_COMICS.columnName);

String imgPath = resultSet.getString(ComicsGalleryFields.IMG\_PATH.columnName);

Pair<Integer, String> newPair = new Pair<>(comicsId, imgPath);

comicsImagePaths.add(newPair);

}

resultSet.close();

} catch (SQLException e) {

logger.error("Error while getting comics image paths from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseImportError"));

Main.errorAlert.show();

}

return comicsImagePaths;

}

/\*

\* Methods which drop data in the database

\*/

public void deleteAnime(Anime anime) {

try {

int animeId = anime.getId();

PreparedStatement preparedStatement = connection.prepareStatement("delete from anime where id = ?");

preparedStatement.setInt(AnimeFields.ID.columnIndex, animeId);

preparedStatement.execute();

} catch (SQLException e) {

logger.error("Error while deleting anime from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseDeleteError"));

Main.errorAlert.show();

}

}

public void deleteComics(Comics comics) {

try {

int comicsId = comics.getId();

PreparedStatement preparedStatement = connection.prepareStatement("delete from comics where id = ?");

preparedStatement.setInt(ComicsFields.ID.columnIndex, comicsId);

preparedStatement.execute();

} catch (SQLException e) {

logger.error("Error while deleting comics from db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseDeleteError"));

Main.errorAlert.show();

}

}

/\*

\* Methods which add data in the database

\*/

@SuppressWarnings("DuplicatedCode")

public void insertNewAnime(Anime newAnime) {

try {

int animeId = newAnime.getId();

int animeAuthorId = newAnime.getAuthor().getId();

String animeType = newAnime.getType().name;

String animeName = newAnime.getName();

String animeDescription = newAnime.getDescription();

int animeEpisodeCount = newAnime.getEpisodeCount();

String animeSource = newAnime.getSource().name;

int animeAgeRatingId = newAnime.getAgeRating().id;

int animePremiereYear = newAnime.getPremiereYear();

String animePremiereSeason = newAnime.getPremiereSeason().name;

String animeStatus = newAnime.getStatus().name;

String insertString = "insert into anime values (?,?,?,?,?,?,?,?,?,?,?)";

PreparedStatement insertStatement = connection.prepareStatement(insertString);

insertStatement.setInt(1, animeId);

insertStatement.setInt(2, animeAuthorId);

insertStatement.setString(3, animeType);

insertStatement.setString(4, animeName);

insertStatement.setString(5, animeDescription);

insertStatement.setInt(6, animeEpisodeCount);

insertStatement.setString(7, animeSource);

insertStatement.setInt(8, animeAgeRatingId);

insertStatement.setInt(9, animePremiereYear);

insertStatement.setString(10, animePremiereSeason);

insertStatement.setString(11, animeStatus);

insertStatement.execute();

} catch (SQLException e) {

logger.error("Error while inserting anime in db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseInsertError"));

Main.errorAlert.show();

}

}

@SuppressWarnings("DuplicatedCode")

public void insertNewComics(Comics newComics) {

try {

int comicsId = newComics.getId();

int comicsAuthorId = newComics.getAuthor().getId();

String comicsType = newComics.getType().name;

String comicsName = newComics.getName();

String comicsDescription = newComics.getDescription();

int comicsChapterCount = newComics.getChapterCount();

String comicsSource = newComics.getSource().name;

int comicsPremiereYear = newComics.getPremiereYear();

String comicsStatus = newComics.getStatus().name;

String insertString = "insert into comics values (?,?,?,?,?,?,?,?,?)";

PreparedStatement insertStatement = connection.prepareStatement(insertString);

insertStatement.setInt(1, comicsId);

insertStatement.setInt(2, comicsAuthorId);

insertStatement.setString(3, comicsType);

insertStatement.setString(4, comicsName);

insertStatement.setString(5, comicsDescription);

insertStatement.setInt(6, comicsChapterCount);

insertStatement.setString(7, comicsSource);

insertStatement.setInt(8, comicsPremiereYear);

insertStatement.setString(9, comicsStatus);

insertStatement.execute();

} catch (SQLException e) {

logger.error("Error while inserting comics in db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseInsertError"));

Main.errorAlert.show();

}

}

/\*

\* Methods which update (specifically "update" or drop & add) data in the database

\*/

public void updateAnime(Anime anime) {

try {

int animeId = anime.getId();

int animeAuthorId = anime.getAuthor().getId();

String animeType = anime.getType().name;

String animeName = anime.getName();

String animeDescription = anime.getDescription();

int animeEpisodeCount = anime.getEpisodeCount();

String animeSource = anime.getSource().name;

int animeAgeRatingId = anime.getAgeRating().id;

int animePremiereYear = anime.getPremiereYear();

String animePremiereSeason = anime.getPremiereSeason().name;

String animeStatus = anime.getStatus().name;

String updateString = "update anime set id\_author=?, type=?, name=?, description=?, " +

"episode\_count=?, source=?, id\_rating=?, premiere\_year=?, premiere\_season=?, " +

"status=? where id=?";

PreparedStatement updateStatement = connection.prepareStatement(updateString);

updateStatement.setInt(1, animeAuthorId);

updateStatement.setString(2, animeType);

updateStatement.setString(3, animeName);

updateStatement.setString(4, animeDescription);

updateStatement.setInt(5, animeEpisodeCount);

updateStatement.setString(6, animeSource);

updateStatement.setInt(7, animeAgeRatingId);

updateStatement.setInt(8, animePremiereYear);

updateStatement.setString(9, animePremiereSeason);

updateStatement.setString(10, animeStatus);

updateStatement.setInt(11, animeId);

updateStatement.execute();

} catch (SQLException e) {

logger.error("Error while updating anime in db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseUpdateError"));

Main.errorAlert.show();

}

}

public void updateComics(Comics comics) {

try {

int animeId = comics.getId();

int animeAuthorId = comics.getAuthor().getId();

String animeType = comics.getType().name;

String animeName = comics.getName();

String animeDescription = comics.getDescription();

int animeChapterCount = comics.getChapterCount();

String animeSource = comics.getSource().name;

int animePremiereYear = comics.getPremiereYear();

String animeStatus = comics.getStatus().name;

String updateString = "update comics set id\_author=?, type=?, name=?, description=?, " +

"episode\_count=?, source=?, premiere\_year=?, status=? where id=?";

PreparedStatement updateStatement = connection.prepareStatement(updateString);

updateStatement.setInt(1, animeAuthorId);

updateStatement.setString(2, animeType);

updateStatement.setString(3, animeName);

updateStatement.setString(4, animeDescription);

updateStatement.setInt(5, animeChapterCount);

updateStatement.setString(6, animeSource);

updateStatement.setInt(7, animePremiereYear);

updateStatement.setString(8, animeStatus);

updateStatement.setInt(9, animeId);

updateStatement.execute();

} catch (SQLException e) {

logger.error("Error while updating comics in db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseUpdateError"));

Main.errorAlert.show();

}

}

@SuppressWarnings("unchecked")

public void updateAnimeGenres(Pair<Anime, ArrayList<Genre>> animeGenres) {

try {

ArrayList<Genre> currentAnimeGenres = getAnimeGenreMap().stream().filter(

animeGenrePair -> animeGenrePair.getKey().getId() == animeGenres.getKey().getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

ArrayList<Genre> genresToAdd = (ArrayList<Genre>) animeGenres.getValue().clone();

genresToAdd.removeIf(genre -> currentAnimeGenres.stream().anyMatch(localGenre ->

localGenre.name.equalsIgnoreCase(genre.name)));

ArrayList<Genre> genresToDrop = (ArrayList<Genre>) currentAnimeGenres.clone();

genresToDrop.removeIf(genre -> animeGenres.getValue().stream().anyMatch(localGenre ->

localGenre.name.equalsIgnoreCase(genre.name)));

if (genresToDrop.size() > 0) {

StringBuilder dropGenresBuilder = new StringBuilder("delete from anime\_genres where id\_anime = ? " +

"and id\_genre in (?");

for (int i = 1; i < genresToDrop.size(); i++) {

dropGenresBuilder.append(",?");

}

dropGenresBuilder.append(")");

PreparedStatement preparedDropStatement = connection.prepareStatement(dropGenresBuilder.toString());

preparedDropStatement.setInt(1, animeGenres.getKey().getId());

for (int i = 1; i < genresToDrop.size() + 1; i++) {

preparedDropStatement.setInt(i + 1, genresToDrop.get(i - 1).id);

}

preparedDropStatement.execute();

}

if (genresToAdd.size() > 0) {

StringBuilder addGenresBuilder = new StringBuilder("insert into anime\_genres values (?,?)");

for (int i = 1; i < genresToAdd.size(); i++) {

addGenresBuilder.append(",(?,?)");

}

PreparedStatement preparedAddStatement = connection.prepareStatement(addGenresBuilder.toString());

for (int i = 1, j = 1; i < genresToAdd.size() + 1; i++, j += 2) {

preparedAddStatement.setInt(j, animeGenres.getKey().getId());

preparedAddStatement.setInt(j + 1, genresToAdd.get(i - 1).id);

}

preparedAddStatement.execute();

}

} catch (Exception e) {

logger.error("Error while updating anime genres in db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseUpdateError"));

Main.errorAlert.show();

}

}

@SuppressWarnings("unchecked")

public void updateAnimeImages(Pair<Anime, ArrayList<String>> animeImages) {

try {

ArrayList<String> currentAnimeImages = getAnimeImagePaths().stream().filter(

animeImagePair -> animeImagePair.getKey() == animeImages.getKey().getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

ArrayList<String> imagesToAdd = (ArrayList<String>) animeImages.getValue().clone();

imagesToAdd.removeIf(image -> currentAnimeImages.stream().anyMatch(localImage ->

localImage.equalsIgnoreCase(image)));

ArrayList<String> imagesToDrop = (ArrayList<String>) currentAnimeImages.clone();

imagesToDrop.removeIf(image -> animeImages.getValue().stream().anyMatch(localImage ->

localImage.equalsIgnoreCase(image)));

if (imagesToDrop.size() > 0) {

StringBuilder dropImagesBuilder = new StringBuilder("delete from anime\_gallery where id\_anime = ? " +

"and img\_path in (?");

for (int i = 1; i < imagesToDrop.size(); i++) {

dropImagesBuilder.append(",?");

}

dropImagesBuilder.append(")");

PreparedStatement preparedDropStatement = connection.prepareStatement(dropImagesBuilder.toString());

preparedDropStatement.setInt(1, animeImages.getKey().getId());

for (int i = 1; i < imagesToDrop.size() + 1; i++) {

preparedDropStatement.setString(i + 1, imagesToDrop.get(i - 1));

}

preparedDropStatement.execute();

}

if (imagesToAdd.size() > 0) {

StringBuilder addImagesBuilder = new StringBuilder("insert into anime\_gallery values (?,?)");

for (int i = 1; i < imagesToAdd.size(); i++) {

addImagesBuilder.append(",(?,?)");

}

PreparedStatement preparedAddStatement = connection.prepareStatement(addImagesBuilder.toString());

for (int i = 1, j = 1; i < imagesToAdd.size() + 1; i++, j += 2) {

preparedAddStatement.setInt(j, animeImages.getKey().getId());

preparedAddStatement.setString(j + 1, imagesToAdd.get(i - 1));

}

preparedAddStatement.execute();

}

} catch (Exception e) {

logger.error("Error while updating anime images in db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseUpdateError"));

Main.errorAlert.show();

}

}

@SuppressWarnings({"unchecked", "DuplicatedCode"})

public void updateComicsGenres(Pair<Comics, ArrayList<Genre>> comicsGenres) {

try {

ArrayList<Genre> currentComicsGenres = getComicsGenreMap().stream().filter(

comicsGenrePair -> comicsGenrePair.getKey().getId() == comicsGenres.getKey().getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

ArrayList<Genre> genresToAdd = (ArrayList<Genre>) comicsGenres.getValue().clone();

genresToAdd.removeIf(genre -> currentComicsGenres.stream().anyMatch(localGenre ->

localGenre.name.equalsIgnoreCase(genre.name)));

ArrayList<Genre> genresToDrop = (ArrayList<Genre>) currentComicsGenres.clone();

genresToDrop.removeIf(genre -> comicsGenres.getValue().stream().anyMatch(localGenre ->

localGenre.name.equalsIgnoreCase(genre.name)));

if (genresToDrop.size() > 0) {

StringBuilder dropGenresBuilder = new StringBuilder("delete from comics\_genres where id\_comics = ? " +

"and id\_genre in (?");

for (int i = 1; i < genresToDrop.size(); i++) {

dropGenresBuilder.append(",?");

}

dropGenresBuilder.append(")");

PreparedStatement preparedDropStatement = connection.prepareStatement(dropGenresBuilder.toString());

preparedDropStatement.setInt(1, comicsGenres.getKey().getId());

for (int i = 1; i < genresToDrop.size() + 1; i++) {

preparedDropStatement.setInt(i + 1, genresToDrop.get(i - 1).id);

}

preparedDropStatement.execute();

}

if (genresToAdd.size() > 0) {

StringBuilder addGenresBuilder = new StringBuilder("insert into comics\_genres values (?,?)");

for (int i = 1; i < genresToAdd.size(); i++) {

addGenresBuilder.append(",(?,?)");

}

PreparedStatement preparedAddStatement = connection.prepareStatement(addGenresBuilder.toString());

for (int i = 1, j = 1; i < genresToAdd.size() + 1; i++, j += 2) {

preparedAddStatement.setInt(j, comicsGenres.getKey().getId());

preparedAddStatement.setInt(j + 1, genresToAdd.get(i - 1).id);

}

preparedAddStatement.execute();

}

} catch (Exception e) {

logger.error("Error while updating anime genres in db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseUpdateError"));

Main.errorAlert.show();

}

}

@SuppressWarnings({"unchecked", "DuplicatedCode"})

public void updateComicsImages(Pair<Comics, ArrayList<String>> comicsImages) {

try {

ArrayList<String> currentComicsImages = getComicsImagePaths().stream().filter(

comicsImagePair -> comicsImagePair.getKey() == comicsImages.getKey().getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

ArrayList<String> imagesToAdd = (ArrayList<String>) comicsImages.getValue().clone();

imagesToAdd.removeIf(image -> currentComicsImages.stream().anyMatch(localImage ->

localImage.equalsIgnoreCase(image)));

ArrayList<String> imagesToDrop = (ArrayList<String>) currentComicsImages.clone();

imagesToDrop.removeIf(image -> comicsImages.getValue().stream().anyMatch(localImage ->

localImage.equalsIgnoreCase(image)));

if (imagesToDrop.size() > 0) {

StringBuilder dropImagesBuilder = new StringBuilder("delete from comics\_gallery where id\_comics = ? " +

"and img\_path in (?");

for (int i = 1; i < imagesToDrop.size(); i++) {

dropImagesBuilder.append(",?");

}

dropImagesBuilder.append(")");

PreparedStatement preparedDropStatement = connection.prepareStatement(dropImagesBuilder.toString());

preparedDropStatement.setInt(1, comicsImages.getKey().getId());

for (int i = 1; i < imagesToDrop.size() + 1; i++) {

preparedDropStatement.setString(i + 1, imagesToDrop.get(i - 1));

}

preparedDropStatement.execute();

}

if (imagesToAdd.size() > 0) {

StringBuilder addImagesBuilder = new StringBuilder("insert into comics\_gallery values (?,?)");

for (int i = 1; i < imagesToAdd.size(); i++) {

addImagesBuilder.append(",(?,?)");

}

PreparedStatement preparedAddStatement = connection.prepareStatement(addImagesBuilder.toString());

for (int i = 1, j = 1; i < imagesToAdd.size() + 1; i++, j += 2) {

preparedAddStatement.setInt(j, comicsImages.getKey().getId());

preparedAddStatement.setString(j + 1, imagesToAdd.get(i - 1));

}

preparedAddStatement.execute();

}

} catch (Exception e) {

logger.error("Error while updating comics images in db: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("databaseUpdateError"));

Main.errorAlert.show();

}

}

}

# Package com.folva.moderneastculture.model.dto

## AgeRating

package com.folva.moderneastculture.model.dto;

/\*\*

\* Віковий рейтинг аніме/коміксу. Перераховані значення признані

\* світом, тому можуть використовуватися й для інших творів

\*/

public enum AgeRating {

G("G", 1), PG("PG", 2), PG13("PG-13", 3),

R("R", 4), NC17("NC-17", 5), UN("UN", 6);

/\*\*

\* Більш коректна назва

\*/

public final String name;

/\*\*

\* Ідентифікатор в таблиці бази данних

\*/

public final int id;

AgeRating(String name, int id) {

this.name = name;

this.id = id;

}

/\*\*

\* @param id ідентифікатор шуканого об'єкту рейтингу

\* @return шуканий об'єкт рейтингу, якщо знайдений, або null, якщо ны

\*/

public static AgeRating valueOfId(int id) {

AgeRating ageRating = null;

for (AgeRating rating : AgeRating.values()) {

if (rating.id == id) {

ageRating = rating;

break;

}

}

return ageRating;

}

}

## Anime

package com.folva.moderneastculture.model.dto;

import com.folva.moderneastculture.model.Repository;

import javafx.beans.InvalidationListener;

import javafx.beans.property.SimpleObjectProperty;

import java.util.ArrayList;

import java.util.Objects;

/\*\*

\* Звичайний DTO-об'єкт аніме

\*/

public class Anime {

/\*\*

\* Типи аніме

\*/

public enum Type {

SERIES("series"),

MOVIE("movie");

public final String name;

Type(String name) {

this.name = name;

}

}

/\*\*

\* Типи джерел аніме

\*/

public enum Source {

ORIGINAL("original"),

MANGA("manga"),

RANOBE("ranobe"),

OTHER("other");

public final String name;

Source(String name) {

this.name = name;

}

}

private int id;

private Author author;

private Type type;

private String name;

private String description;

private int episodeCount;

private Source source;

private AgeRating ageRating;

private int premiereYear;

private YearSeason premiereSeason;

private Status status;

private ArrayList<String> altNames;

private final ArrayList<InvalidationListener> changeListeners;

private Anime() {

altNames = new ArrayList<>();

changeListeners = new ArrayList<>();

}

/\*\*

\* Метод повідомлення слухачів змін стану

\*/

public void invalidate() {

changeListeners.forEach(invalidationListener ->

invalidationListener.invalidated(new SimpleObjectProperty<>(this)));

}

/\*\*

\* Метод добавлення нових слухачів змін

\* @param listener новий слухач змін

\*/

public void addInvalidationListener(InvalidationListener listener) {

changeListeners.add(listener);

}

/\*\*

\* @return ідентифікатор аніме

\*/

public int getId() {

return id;

}

/\*\*

\* Встановлює новий ідентифікатор аніме

\* @param id новий ідентифікатор аніме

\*/

public void setId(int id) {

this.id = id;

}

/\*\*

\* @return об'єкт автору аніме

\*/

public Author getAuthor() {

return author;

}

/\*\*

\* Встановлює новий об'єкт автору аніме

\* @param author новий об'єкт автору аніме

\*/

public void setAuthor(Author author) {

this.author = author;

}

/\*\*

\* @return тип аніме

\*/

public Type getType() {

return type;

}

/\*\*

\* Встановлює новий тип аніме

\* @param type новий тип аніме

\*/

public void setType(Type type) {

this.type = type;

}

/\*\*

\* @return назва аніме

\*/

public String getName() {

return name;

}

/\*\*

\* Встановлює нову назву аніме

\* @param name нова назва аніме

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* @return опис аніме

\*/

public String getDescription() {

return description;

}

/\*\*

\* Встановлює новий опис аніме

\* @param description новий опис аніме

\*/

public void setDescription(String description) {

this.description = description;

}

/\*\*

\* @return кількість епізодів

\*/

public int getEpisodeCount() {

return episodeCount;

}

/\*\*

\* Встановлює нову кількість епізодів

\* @param episodeCount нова кількість епізодів

\*/

public void setEpisodeCount(int episodeCount) {

this.episodeCount = episodeCount;

}

/\*\*

\* @return тип джерела аніме

\*/

public Source getSource() {

return source;

}

/\*\*

\* Встановлює новий тип джерела аніме

\* @param source новий тип джерела аніме

\*/

public void setSource(Source source) {

this.source = source;

}

/\*\*

\* @return віковий рейтинг аніме

\*/

public AgeRating getAgeRating() {

return ageRating;

}

/\*\*

\* Встановлює новий віковий рейтинг аніме

\* @param rating новий віковий рейтинг аніме

\*/

public void setAgeRating(AgeRating rating) {

this.ageRating = rating;

}

/\*\*

\* @return рік початку випуску аніме

\*/

public int getPremiereYear() {

return premiereYear;

}

/\*\*

\* Встановлює новий рік початку випуску аніме

\* @param premiereYear новий рік початку випуску аніме

\*/

public void setPremiereYear(int premiereYear) {

this.premiereYear = premiereYear;

}

/\*\*

\* @return сезон випуску аніме

\*/

public YearSeason getPremiereSeason() {

return premiereSeason;

}

/\*\*

\* Встановлює новий сезон випуску аніме

\* @param premiereSeason новий сезон випуску аніме

\*/

public void setPremiereSeason(YearSeason premiereSeason) {

this.premiereSeason = premiereSeason;

}

/\*\*

\* @return статус аніме

\*/

public Status getStatus() {

return status;

}

/\*\*

\* Встановлює новий статус аніме

\* @param status новий статус аніме

\*/

public void setStatus(Status status) {

this.status = status;

}

/\*\*

\* @return колекція альтернативних імен аніме

\*/

public ArrayList<String> getAltNames() {

return altNames;

}

/\*\*

\* Встановлює нову колекцію альтернативних імен аніме

\* @param altNames нова колекція альтернативних імен аніме

\*/

public void setAltNames(ArrayList<String> altNames) {

this.altNames = altNames;

}

/\*\*

\* Будувач аніме для більш елегантного та зрозумілого конструювання об'єкту аніме

\*/

@Override

public boolean equals(Object obj) {

boolean result = false;

if (obj instanceof Anime) {

Anime anime = (Anime) obj;

boolean sameId = (id == anime.id);

boolean sameAuthor = (Objects.equals(author, anime.author));

boolean sameType = (Objects.equals(type, anime.type));

boolean sameName = (Objects.equals(name, anime.name));

boolean sameDescription = (Objects.equals(description, anime.description));

boolean sameEpisodeCount = (episodeCount == anime.episodeCount);

boolean sameSource = (Objects.equals(source, anime.source));

boolean sameRating = (Objects.equals(ageRating, anime.ageRating));

boolean samePremiereYear = (premiereYear == anime.premiereYear);

boolean samePremiereSeason = (Objects.equals(premiereSeason, anime.premiereSeason));

boolean sameStatus = (Objects.equals(status, anime.status));

boolean sameAltNames = (Objects.equals(altNames, anime.altNames));

result = sameId && sameAuthor && sameType && sameName && sameDescription

&& sameEpisodeCount && sameSource && sameRating && samePremiereYear

&& samePremiereSeason && sameStatus && sameAltNames;

}

return result;

}

}

## Anime.Builder

public static class Builder {

private Anime anime;

private Builder() {

anime = new Anime();

}

/\*\*

\* @return новий об'єкт будівника

\*/

public static Builder newBuilder() {

return new Builder();

}

/\*\*

\* @param id ідентифікатор аніме

\* @return цей будівник

\*/

public Builder setId(int id) {

anime.id = id;

return this;

}

/\*\*

\* @param author автор аніме

\* @return цей будівник

\*/

public Builder setAuthor(Author author) {

anime.author = author;

return this;

}

/\*\*

\* @param type тип аніме

\* @return цей будівник

\*/

public Builder setType(Type type) {

anime.type = type;

return this;

}

/\*\*

\* @param name назва аніме

\* @return цей будівник

\*/

public Builder setName(String name) {

anime.name = name;

return this;

}

/\*\*

\* @param description опис аніме

\* @return цей будівник

\*/

public Builder setDescription(String description) {

anime.description = description;

return this;

}

/\*\*

\* @param episodeCount кількість епізодів аніме

\* @return цей будівник

\*/

public Builder setEpisodeCount(int episodeCount) {

anime.episodeCount = episodeCount;

return this;

}

/\*\*

\* @param source тип джерела аніме

\* @return цей будівник

\*/

public Builder setSource(Source source) {

anime.source = source;

return this;

}

/\*\*

\* @param rating віковий рейтинг аніме

\* @return цей будівник

\*/

public Builder setAgeRating(AgeRating rating) {

anime.ageRating = rating;

return this;

}

/\*\*

\* @param premiereYear рік початку випуску аніме

\* @return цей будівник

\*/

public Builder setPremiereYear(int premiereYear) {

anime.premiereYear = Math.max(premiereYear, Repository.FIRST\_ANIME\_PREMIERE\_YEAR);

return this;

}

/\*\*

\* @param season сезон початку випуску аніме

\* @return цей будівник

\*/

public Builder setPremiereSeason(YearSeason season) {

anime.premiereSeason = season;

return this;

}

/\*\*

\* @param status статус аніме

\* @return цей будівник

\*/

public Builder setStatus(Status status) {

anime.status = status;

return this;

}

/\*\*

\* @param altNames альтернативні назви аніме

\* @return цей будівник

\*/

public Builder setAltNames(ArrayList<String> altNames) {

anime.altNames = altNames;

return this;

}

/\*\*

\* @return збудований об'єкт аніме

\*/

public Anime build() {

Anime newAnime = anime;

anime = null;

return newAnime;

}

}

## Author

package com.folva.moderneastculture.model.dto;

/\*\*

\* Звичайний DTO-об'єкт автору

\*/

public class Author {

/\*\*

\* Типи авторів

\*/

public enum Type {

STUDIO,

HUMAN

}

private int id;

private Type type;

private String name;

private Author() {}

/\*\*

\* @return ідентифікатор автора

\*/

public int getId() {

return id;

}

/\*\*

\* Встановлює новий ідентифікатор автора

\* @param id новий ідентифікатор автора

\*/

public void setId(int id) {

this.id = id;

}

/\*\*

\* @return тип автору

\*/

public Type getType() {

return type;

}

/\*\*

\* Встановлює новий тип автора

\* @param type новий тип автору

\*/

public void setType(Type type) {

this.type = type;

}

/\*\*

\* @return ім'я автору

\*/

public String getName() {

return name;

}

/\*\*

\* Встановлює нове ім'я автору

\* @param name нове ім'я автору

\*/

public void setName(String name) {

this.name = name;

}

@Override

public boolean equals(Object obj) {

boolean result = false;

if (obj instanceof Author) {

Author author = (Author) obj;

boolean sameId = (id == author.id);

boolean sameType = (type.equals(author.type));

boolean sameName = (name.equals(author.name));

result = sameId && sameType && sameName;

}

return result;

}

}

## Author.Builder

/\*\*

\* "Будівник" класу з більш комфортними іменами встановлення

\* значень полів

\*/

public static class Builder {

private final Author author;

private Builder() {

author = new Author();

}

/\*\*

\* Фабричний метод створення нового будівника

\* @return новий будівник

\*/

public static Builder newBuilder() {

return new Builder();

}

/\*\*

\* @param id ідентифікатор автору

\* @return цей будівник

\*/

public Builder setId(int id) {

author.id = id;

return this;

}

/\*\*

\* @param type тип автору

\* @return цей будівник

\*/

public Builder setType(Type type) {

author.type = type;

return this;

}

/\*\*

\* @param name ім'я автору

\* @return цей будівник

\*/

public Builder setName(String name) {

author.name = name;

return this;

}

/\*\*

\* @return "побудований" автор

\*/

public Author build() {

return author;

}

}

## Comics

package com.folva.moderneastculture.model.dto;

import com.folva.moderneastculture.model.Repository;

import javafx.beans.InvalidationListener;

import javafx.beans.property.SimpleObjectProperty;

import java.util.ArrayList;

import java.util.Objects;

/\*\*

\* Звичайний DTO-об'єкт коміксу

\*/

public class Comics {

/\*\*

\* Типи коміксів

\*/

public enum Type {

MANGA("manga"),

RANOBE("ranobe");

public final String name;

Type(String name) {

this.name = name;

}

}

/\*\*

\* Типи джерел коміксів

\*/

public enum Source {

ORIGINAL("original"),

RANOBE("ranobe"),

OTHER("other");

public final String name;

Source(String name) {

this.name = name;

}

}

private int id;

private Author author;

private Type type;

private String name;

private String description;

private int chapterCount;

private Source source;

private int premiereYear;

private Status status;

private ArrayList<String> altNames;

private final ArrayList<InvalidationListener> changeListeners;

private Comics() {

altNames = new ArrayList<>();

changeListeners = new ArrayList<>();

}

/\*\*

\* Надіслати всім слухачам повідомлення

\*/

public void invalidate() {

changeListeners.forEach(invalidationListener ->

invalidationListener.invalidated(new SimpleObjectProperty<>(this)));

}

/\*\*

\* Додати нового слухача змін

\* @param listener новий слухач змін об'єкту

\*/

public void addInvalidationListener(InvalidationListener listener) {

changeListeners.add(listener);

}

/\*\*

\* @return ідентифікатор коміксу

\*/

public int getId() {

return id;

}

/\*\*

\* Встановлює новий ідентифікатор коміксу

\* @param id новий ідентифікатор коміксу

\*/

public void setId(int id) {

this.id = id;

}

/\*\*

\* @return об'єкт автору

\*/

public Author getAuthor() {

return author;

}

/\*\*

\* Встановлює новий об'єкт автора

\* @param author новий об'єкт автора

\*/

public void setAuthor(Author author) {

this.author = author;

}

/\*\*

\* @return тип коміксу

\*/

public Type getType() {

return type;

}

/\*\*

\* Встановлює новий тип коміксу

\* @param type новий тип коміксу

\*/

public void setType(Type type) {

this.type = type;

}

/\*\*

\* @return назва коміксу

\*/

public String getName() {

return name;

}

/\*\*

\* Встановлює нову назву коміксу

\* @param name нова назва коміксу

\*/

public void setName(String name) {

this.name = name;

}

/\*\*

\* @return опис коміксу

\*/

public String getDescription() {

return description;

}

/\*\*

\* Встановлює новий опис коміксу

\* @param description новий опис коміксу

\*/

public void setDescription(String description) {

this.description = description;

}

/\*\*

\* @return кількість глав коміксу

\*/

public int getChapterCount() {

return chapterCount;

}

/\*\*

\* Встановлює нову кількість глав коміксу

\* @param chapterCount нова кількість глав коміксу

\*/

public void setChapterCount(int chapterCount) {

this.chapterCount = chapterCount;

}

/\*\*

\* @return тип джерела коміксу

\*/

public Source getSource() {

return source;

}

/\*\*

\* Встановлює новий тип джерела коміксу

\* @param source новий тип джерела коміксу

\*/

public void setSource(Source source) {

this.source = source;

}

/\*\*

\* @return рік початку випуску

\*/

public int getPremiereYear() {

return premiereYear;

}

/\*\*

\* Повертає рік початку випуску коміксу

\* @param premiereYear новий рік початку випуску

\*/

public void setPremiereYear(int premiereYear) {

this.premiereYear = premiereYear;

}

/\*\*

\* @return статус коміксу

\*/

public Status getStatus() {

return status;

}

/\*\*

\* Встановлює новий статус коміксу

\* @param status новий статус коміксу

\*/

public void setStatus(Status status) {

this.status = status;

}

/\*\*

\* @return колекція альтернативних імен коміксу

\*/

public ArrayList<String> getAltNames() {

return altNames;

}

/\*\*

\* Встановлює нову колекцію альтернативних імен коміксу

\* @param altNames нова колекція альтернативних імен коміксу

\*/

public void setAltNames(ArrayList<String> altNames) {

this.altNames = altNames;

}

@Override

public boolean equals(Object obj) {

boolean result = false;

if (obj instanceof Comics) {

Comics comics = (Comics) obj;

boolean sameId = (id == comics.id);

boolean sameAuthor = (Objects.equals(author, comics.author));

boolean sameType = (Objects.equals(type, comics.type));

boolean sameName = (Objects.equals(name, comics.name));

boolean sameDescription = (Objects.equals(description, comics.description));

boolean sameChapterCount = (chapterCount == comics.chapterCount);

boolean sameSource = (Objects.equals(source, comics.source));

boolean samePremiereYear = (premiereYear == comics.premiereYear);

boolean sameStatus = (Objects.equals(status, comics.status));

boolean sameAltNames = (Objects.equals(altNames, comics.altNames));

result = sameId && sameAuthor && sameType && sameName

&& sameDescription && sameChapterCount

&& sameSource && samePremiereYear

&& sameStatus && sameAltNames;

}

return result;

}

}

## Comics.Builder

/\*\*

\* "Будувач" коміксу для більш елегантного конструювання об'єкту

\*/

public static class Builder {

private Comics comics;

private Builder() {

comics = new Comics();

}

/\*\*

\* Фабричний метод будівника

\* @return новий будувач

\*/

public static Builder newBuilder() {

return new Builder();

}

/\*\*

\* @param id ідентифікатор коміксу

\* @return цей будувач

\*/

public Builder setId(int id) {

comics.id = id;

return this;

}

/\*\*

\* @param author автор коміксу

\* @return цей будувач

\*/

public Builder setAuthor(Author author) {

comics.author = author;

return this;

}

/\*\*

\* @param type тип коміксу

\* @return цей будувач

\*/

public Builder setType(Type type) {

comics.type = type;

return this;

}

/\*\*

\* @param name назва коміксу

\* @return цей будувач

\*/

public Builder setName(String name) {

comics.name = name;

return this;

}

/\*\*

\* @param description опис коміксу

\* @return цей будувач

\*/

public Builder setDescription(String description) {

comics.description = description;

return this;

}

/\*\*

\* @param chapterCount кількість глав коміксу

\* @return цей будувач

\*/

public Builder setChapterCount(int chapterCount) {

comics.chapterCount = chapterCount;

return this;

}

/\*\*

\* @param source тип джерела коміксу

\* @return цей будувач

\*/

public Builder setSource(Source source) {

comics.source = source;

return this;

}

/\*\*

\* @param premiereYear рік початку випуску коміксу

\* @return цей будувач

\*/

public Builder setPremiereYear(int premiereYear) {

comics.premiereYear = Math.max(premiereYear, Repository.FIRST\_MANGA\_PREMIERE\_YEAR);

return this;

}

/\*\*

\* @param status статус коміксу

\* @return цей будувач

\*/

public Builder setStatus(Status status) {

comics.status = status;

return this;

}

/\*\*

\* @param altNames альтернативні назви коміксу

\* @return цей будувач

\*/

public Builder setAltNames(ArrayList<String> altNames) {

comics.altNames = altNames;

return this;

}

/\*\*

\* @return побудований об'єкт коміксу

\*/

public Comics build() {

Comics newComics = comics;

comics = null;

return newComics;

}

}

## Genre

package com.folva.moderneastculture.model.dto;

/\*\*

\* Звичайний DTO-об'єкт жанру

\*/

public class Genre {

public final int id;

public final String name;

public Genre(int id, String name) {

this.id = id;

this.name = name;

}

}

## OpenPair

package com.folva.moderneastculture.model.dto;

/\*\*

\* "Відкрита" версія класу {@link javafx.util.Pair}. Дає

\* можливість змінювати ключ та значення, проте типи, що логічно,

\* залишаються тими же

\* @param <K> Тип ключа

\* @param <V> Тип значення

\*/

public final class OpenPair<K, V> {

private K key;

/\*\*

\* @return ключ

\*/

public K getKey() { return key; }

/\*\*

\* Встановлює новий ключ з тим же типом або розширяющим

\* @param key ключ

\*/

public void setKey(K key) {

this.key = key;

}

private V value;

/\*\*

\* @return значення

\*/

public V getValue() { return value; }

/\*\*

\* Встановлює нове значення з тим же типом або розширяющим

\* @param value нове значення

\*/

public void setValue(V value) {

this.value = value;

}

/\*\*

\* Відкрита пара "Ключ-значення" по типу тих, що зберігаються в

\* {@link java.util.Map}

\* @param key ключ

\* @param value значення

\*/

public OpenPair(K key, V value) {

this.key = key;

this.value = value;

}

}

## Status

package com.folva.moderneastculture.model.dto;

/\*\*

\* Статус аніме/коміксу. Зазвичай він або анонсований,

\* або виходить наразі, або вже закінчений

\*/

public enum Status {

ANNOUNCED("announced"),

ONGOING("ongoing"),

FINISHED("finished");

/\*\*

\* Назва статусу в малому регістрі. Заміняє ланцюг методів

\* name().toLowerCase()

\*/

public final String name;

Status(String name) {

this.name = name;

}

}

## YearSeason

package com.folva.moderneastculture.model.dto;

/\*\*

\* Сезони року. Включається "неідентифікований" сезон року для

\* різних випадків

\*/

public enum YearSeason {

WINTER("winter"), SPRING("spring"),

SUMMER("summer"), AUTUMN("autumn"),

UNDEFINED("undefined");

/\*\*

\* Назва сезону в малому регістрі. Заміняє ланцюг методів

\* name().toLowerCase()

\*/

public final String name;

YearSeason(String name) {

this.name = name;

}

@Override

public String toString() {

return name;

}

}

# Package com.folva.moderneastculture.controller

## AnimeController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import com.folva.moderneastculture.model.dto.\*;

import com.sun.javafx.collections.ObservableListWrapper;

import javafx.beans.InvalidationListener;

import javafx.beans.binding.DoubleBinding;

import javafx.beans.property.SimpleObjectProperty;

import javafx.beans.property.StringProperty;

import javafx.beans.value.ChangeListener;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.control.\*;

import javafx.scene.layout.VBox;

import javafx.util.Pair;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import java.net.URL;

import java.util.ArrayList;

import java.util.ResourceBundle;

import java.util.stream.Collectors;

/\*\*

\* Контроллер відповідаючий за головне підменю аніме - де можна продивитися увесь

\* список наявних коміксів та відфільтрувати, викликати меню редагування, додавання,

\* оглядання повної інформації якогось коміксу або видалити взагалі аніме

\*/

public class AnimeController implements Initializable {

private static final Logger logger = LogManager.getLogger(AnimeController.class.getName());

private static final int ANIME\_CONTROLS\_PER\_PAGE = 5;

private static final int ANIME\_CONTROL\_HEIGHT = 100;

@FXML

private ScrollPane spAnimeList;

@FXML

private ChoiceBox<String> cbAnimeStatuses;

@FXML

private ChoiceBox<String> cbAnimeTypes;

@FXML

private ChoiceBox<String> cbAgeRatings;

@FXML

private ListView<String> lvGenres;

@FXML

private TextField tbYearFrom;

@FXML

private TextField tbYearTo;

@FXML

private Pagination animePagination;

@FXML

private TextField tbAnimeName;

@FXML

private Button bAddNewAnime;

private final ArrayList<OpenPair<Anime, AnimePresentationControl>> animeList = new ArrayList<>();

private final SimpleObjectProperty<OpenPair<Boolean, Anime>> editingObject = new SimpleObjectProperty<>();

private ObservableListWrapper<OpenPair<Anime, AnimePresentationControl>> filteredAnimeList;

private Tab tabAnime;

@Override

public void initialize(URL location, ResourceBundle resources) {

importAnime();

setUpFiltrationData();

setPagination();

bindProperties();

logger.info("AnimeController has been loaded!");

}

/\*\*

\* Встановлює посилання на вкладку аніме для переключення між підменю

\* @param tabAnime посилання на вкладку аніме

\*/

public void setTabAnime(Tab tabAnime) {

this.tabAnime = tabAnime;

}

private void importAnime() {

ArrayList<Anime> animes = Repository.instance.getAnimes(true);

animeList.clear();

animeList.addAll(animes.stream().map(anime ->

new OpenPair<Anime, AnimePresentationControl>(anime, null))

.collect(Collectors.toCollection(ArrayList::new)));

filteredAnimeList = new ObservableListWrapper<>(new ArrayList<>());

filteredAnimeList.addAll(animeList);

setAnimeDisplay();

}

@SuppressWarnings("DuplicatedCode")

private void setAnimeDisplay() {

filteredAnimeList.addListener((InvalidationListener) c -> {

int newPageCount = (int) Math.ceil(filteredAnimeList.size() / (double) ANIME\_CONTROLS\_PER\_PAGE);

newPageCount = (newPageCount > 0) ? newPageCount : 1;

animePagination.setPageCount(newPageCount + 2);

animePagination.setPageCount(newPageCount);

});

}

private void bindProperties() {

editingObject.addListener((observable, oldValue, newValue) -> {

boolean startEditing = newValue.getKey();

if (startEditing) {

Parent editPane = Main.getForm("EditAnimeForm");

try {

EditAnimeController editController = (EditAnimeController) Main

.getControllerForForm("EditAnimeForm");

editController.setEditingObjectReference(editingObject);

} catch (Exception e) {

logger.error("Error while setting called anime control for edit controller: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("problemOccurred"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

tabAnime.setContent(editPane);

} else {

Parent animeContentPane = Main.getForm("AnimeForm");

importAnime();

tabAnime.setContent(animeContentPane);

}

});

bAddNewAnime.visibleProperty().bind(Repository.adminIsAuthorizedProperty);

tbAnimeName.prefWidthProperty().bind(new DoubleBinding() {

{

bind(Repository.adminIsAuthorizedProperty);

}

@Override

protected double computeValue() {

if (Repository.adminIsAuthorizedProperty.get()) {

tbAnimeName.setLayoutX(88);

return 700;

} else {

tbAnimeName.setLayoutX(22);

return 766;

}

}

});

}

private void setPagination() {

int newPageCount = (int) Math.ceil(animeList.size() / (double) ANIME\_CONTROLS\_PER\_PAGE);

animePagination.setPageCount(newPageCount);

animePagination.currentPageIndexProperty().addListener(observable -> spAnimeList.setVvalue(0));

animePagination.setPageFactory(pageIndex -> {

final int controlCountOnTheLastPage = filteredAnimeList.size() - pageIndex \* ANIME\_CONTROLS\_PER\_PAGE;

final int startAnimeIndex = pageIndex \* ANIME\_CONTROLS\_PER\_PAGE;

final int endAnimeIndex = startAnimeIndex + Math.min(ANIME\_CONTROLS\_PER\_PAGE, controlCountOnTheLastPage);

ArrayList<OpenPair<Anime, AnimePresentationControl>> animeListToPresent = new ArrayList<>(

filteredAnimeList.subList(startAnimeIndex, endAnimeIndex));

ArrayList<AnimePresentationControl> animeControls = new ArrayList<>();

for (OpenPair<Anime, AnimePresentationControl> animePair : animeListToPresent) {

if (animePair.getValue() != null) {

animeControls.add(animePair.getValue());

continue;

}

Anime anime = animePair.getKey();

anime.addInvalidationListener(observable -> {

AnimePresentationControl presentationControl = animePair.getValue();

presentationControl.setAnime(anime);

presentationControl.loadImage();

});

AnimePresentationControl animeControl = new AnimePresentationControl();

animeControl.logger = logger;

animeControl.load();

animeControl.setAnime(anime);

animeControl.loadImage();

animeControl.isDeleted.addListener(observable -> {

logger.info("Anime: " + anime.getName() + " was deleted");

animeList.remove(animePair);

filteredAnimeList.remove(animePair);

});

animeControl.infoCallHandled.addListener((observable, oldValue, newValue) -> {

boolean startLooking = !newValue;

if (startLooking) {

Parent editPane = Main.getForm("AnimeInfoForm");

try {

AnimeInfoController infoController = (AnimeInfoController) Main

.getControllerForForm("AnimeInfoForm");

infoController.setAnimeControllerReference(animeControl);

} catch (Exception e) {

logger.error("Error while setting called anime control for info controller: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("problemOccurred"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

tabAnime.setContent(editPane);

} else {

Parent animeContentPane = Main.getForm("AnimeForm");

importAnime();

tabAnime.setContent(animeContentPane);

}

});

animeControl.setEditingObjectReference(editingObject);

animeControls.add(animeControl);

animePair.setValue(animeControl);

}

VBox pane = new VBox();

pane.setPrefHeight(ANIME\_CONTROL\_HEIGHT \* ANIME\_CONTROLS\_PER\_PAGE);

pane.getChildren().addAll(animeControls);

return pane;

});

}

private void setUpFiltrationData() {

setUpGenres();

setAnimeTypes();

setAnimeStatuses();

setAnimeYearBoxes();

setAgeRatings();

}

private void setAgeRatings() {

ArrayList<String> ageRatings = new ArrayList<>();

for (AgeRating ageRating : AgeRating.values()) {

ageRatings.add(ageRating.toString());

}

ageRatings.add(Repository.instance.getNamesBundleValue("ignoreValue"));

cbAgeRatings.getItems().addAll(ageRatings);

}

private void setAnimeYearBoxes() {

ChangeListener<String> tbChangeListener = (observable, oldValue, newValue) -> {

if (!newValue.matches("[0-9]{0,4}")) {

((StringProperty) observable).setValue(oldValue);

}

};

tbYearFrom.textProperty().addListener(tbChangeListener);

tbYearTo.textProperty().addListener(tbChangeListener);

}

@SuppressWarnings("DuplicatedCode")

private void setAnimeStatuses() {

ArrayList<String> animeStatuses = new ArrayList<>();

for (Status status : Status.values()) {

animeStatuses.add(Repository.instance.getNamesBundleValue(status.name));

}

animeStatuses.add(Repository.instance.getNamesBundleValue("ignoreValue"));

cbAnimeStatuses.getItems().addAll(animeStatuses);

}

private void setAnimeTypes() {

ArrayList<String> animeTypes = new ArrayList<>();

for (Anime.Type value : Anime.Type.values()) {

animeTypes.add(Repository.instance.getNamesBundleValue(value.name));

}

animeTypes.add(Repository.instance.getNamesBundleValue("ignoreValue"));

cbAnimeTypes.getItems().addAll(animeTypes);

}

private void setUpGenres() {

lvGenres.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

lvGenres.getItems().addAll(Repository.instance.getGenres(false).stream()

.map(genre -> genre.name)

.collect(Collectors.toCollection(ArrayList::new)));

}

@SuppressWarnings("DuplicatedCode")

@FXML

private void onBFilter() {

if (!filterFieldsAreValidated()) return;

filteredAnimeList.clear();

ArrayList<Pair<Anime, Genre>> animeGenreMap = Repository.instance.getAnimeGenreMap(true);

ArrayList<String> selectedGenres = new ArrayList<>(lvGenres.getSelectionModel().getSelectedItems());

String selectedType = cbAnimeTypes.getSelectionModel().getSelectedItem();

String selectedStatus = cbAnimeStatuses.getSelectionModel().getSelectedItem();

int selectedYearFrom = tbYearFrom.getText().isEmpty() ? 0 : Integer.parseInt(tbYearFrom.getText());

int selectedYearTo = tbYearTo.getText().isEmpty() ? 9999 : Integer.parseInt(tbYearTo.getText());

String selectedAgeRating = cbAgeRatings.getSelectionModel().getSelectedItem();

filteredAnimeList.addAll(animeList.stream().filter(animePair -> {

Anime anime = animePair.getKey();

boolean containGenre = false;

boolean correctType = false;

boolean correctStatus = false;

boolean correctYear = false;

boolean correctAgeRating = false;

if (selectedGenres.isEmpty() ||

animeGenreMap.stream().anyMatch(animeGenrePair -> {

boolean currentAnime = (animeGenrePair.getKey().equals(anime));

boolean containsAtLeastOneSelectedGenre = selectedGenres.contains(animeGenrePair.getValue().name);

return currentAnime && containsAtLeastOneSelectedGenre;

}))

containGenre = true;

if (selectedType == null ||

selectedType.equals(Repository.instance.getNamesBundleValue("ignoreValue")) ||

selectedType.equals(Repository.instance.getNamesBundleValue(anime.getType().name)))

correctType = true;

if (selectedStatus == null ||

selectedStatus.equals(Repository.instance.getNamesBundleValue("ignoreValue")) ||

selectedStatus.equals(Repository.instance.getNamesBundleValue(anime.getStatus().name)))

correctStatus = true;

if (anime.getPremiereYear() >= selectedYearFrom && anime.getPremiereYear() <= selectedYearTo)

correctYear = true;

if (selectedAgeRating == null ||

selectedAgeRating.equals(Repository.instance.getNamesBundleValue("ignoreValue")) ||

selectedAgeRating.equals(anime.getAgeRating().name))

correctAgeRating = true;

return containGenre && correctType && correctStatus && correctYear && correctAgeRating;

})

.collect(Collectors.toCollection(ArrayList::new)));

}

@SuppressWarnings("DuplicatedCode")

private boolean filterFieldsAreValidated() {

int selectedYearFrom = tbYearFrom.getText().isEmpty() ? 0 : Integer.parseInt(tbYearFrom.getText());

int selectedYearTo = tbYearTo.getText().isEmpty() ? 9999 : Integer.parseInt(tbYearTo.getText());

if (selectedYearFrom > selectedYearTo) {

Main.warningAlert.setContentText(Repository.instance.getNamesBundleValue("yearValidationIncorrect"));

Main.warningAlert.show();

return false;

}

return true;

}

@SuppressWarnings("DuplicatedCode")

@FXML

private void onBSearchByNameClick() {

String animeName = tbAnimeName.getText();

if (animeName.matches("[ ]+")) return;

filteredAnimeList.clear();

filteredAnimeList.addAll(animeList.stream().filter(anime -> {

boolean nameContainsSearchedSymbols = anime.getKey().getName().contains(animeName);

boolean anyAltNameContainsSearchedSymbols = anime.getKey().getAltNames().stream()

.anyMatch(altName -> altName.contains(animeName));

return nameContainsSearchedSymbols || anyAltNameContainsSearchedSymbols;

})

.collect(Collectors.toCollection(ArrayList::new)));

}

@FXML

private void onBAddNewAnimeClick() {

editingObject.set(new OpenPair<>(true, null));

}

}

## AnimeInfoController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import com.folva.moderneastculture.model.dto.Anime;

import com.folva.moderneastculture.model.dto.Genre;

import javafx.beans.binding.DoubleBinding;

import javafx.event.ActionEvent;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Label;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.layout.VBox;

import javafx.util.Pair;

import java.net.URL;

import java.util.ArrayList;

import java.util.Optional;

import java.util.ResourceBundle;

import java.util.stream.Collectors;

/\*\*

\* Контроллер, який відповідає за меню відображення повної інформації про аніме

\*/

@SuppressWarnings("DuplicatedCode")

public class AnimeInfoController implements Initializable {

@FXML

private Label lAnimeName;

@FXML

private Label lAnimeAltNames;

@FXML

private Label lStatus;

@FXML

private Label lYear;

@FXML

private Label lPremiereSeason;

@FXML

private Label lAgeRating;

@FXML

private Label lGenresTitle;

@FXML

private Label lGenres;

@FXML

private Label lSource;

@FXML

private Label lAuthorTitle;

@FXML

private Label lAuthor;

@FXML

private Label lType;

@FXML

private Label lEpisodeCount;

@FXML

private Label lDescription;

@FXML

private ImageView ivFirstImage;

private AnimePresentationControl animePresentationControl;

@Override

public void initialize(URL location, ResourceBundle resources) {

lGenres.maxWidthProperty().bind(new DoubleBinding() {

private static final int MAX\_LEFT\_PART\_WIDTH = 517;

private static final int COLON\_WIDTH = 10;

{ bind(lGenresTitle.widthProperty()); }

@SuppressWarnings("UnnecessaryLocalVariable")

@Override

protected double computeValue() {

double newWidth = MAX\_LEFT\_PART\_WIDTH - COLON\_WIDTH - lGenresTitle.getWidth();

return newWidth;

}

});

}

/\*\*

\* @param animePresentationControl посилання на об'єкт елементу керування, який викликав це меню.

\* Коли дане меню буде закриватися, у властивості цього елементу

\* керування змінять стан на true. Також дозволяє вилучити аніме

\* для відображення

\*/

public void setAnimeControllerReference(AnimePresentationControl animePresentationControl) {

this.animePresentationControl = animePresentationControl;

Anime anime = animePresentationControl.getAnime();

lAnimeName.setText(anime.getName());

Optional<String> collectedAltNames = anime.getAltNames()

.stream()

.reduce((s, s2) -> s.concat("\n").concat(s2));

if (!collectedAltNames.isPresent()) {

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("fatalError"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

lAnimeAltNames.setText(collectedAltNames.get());

lStatus.setText(Repository.instance.getNamesBundleValue(anime.getStatus().name));

lYear.setText(String.valueOf(anime.getPremiereYear()));

lPremiereSeason.setText(Repository.instance.getNamesBundleValue(anime.getPremiereSeason().name));

lAgeRating.setText(anime.getAgeRating().name);

ArrayList<Pair<Anime, Genre>> animeGenreMap = Repository.instance.getAnimeGenreMap(true);

ArrayList<String> currentAnimeGenreNames = animeGenreMap.stream()

.filter(animeGenrePair -> animeGenrePair.getKey().getId() == anime.getId())

.map(Pair::getValue)

.map(genre -> genre.name)

.collect(Collectors.toCollection(ArrayList::new));

String genresString = currentAnimeGenreNames

.stream()

.reduce("", (s, s2) -> s.concat(", ".concat(s2)))

.substring(2);

lGenres.setText(genresString);

lSource.setText(Repository.instance.getNamesBundleValue(anime.getSource().name));

lAuthorTitle.setText(Repository.instance.getNamesBundleValue(anime

.getAuthor().getType().name().toLowerCase()));

lAuthor.setText(anime.getAuthor().getName());

lType.setText(Repository.instance.getNamesBundleValue(anime.getType().name));

lEpisodeCount.setText(String.valueOf(anime.getEpisodeCount()));

lDescription.setText(anime.getDescription());

ArrayList<String> imagePaths = Repository.instance.getAnimeImagePaths(true)

.stream()

.filter(animeImagePathPair -> animeImagePathPair.getKey() == anime.getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

ivFirstImage.setImage(Repository.getNoImageImage());

if (imagePaths.size() != 0) {

String firstImageName = imagePaths.get(0);

Image firstImage = Repository.loadImage(firstImageName);

if (firstImage != null) ivFirstImage.setImage(firstImage);

}

}

@FXML

private void onBExitEditWindowClick() {

animePresentationControl.infoCallHandled.setValue(true);

}

}

## AnimePresentationControl

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import com.folva.moderneastculture.model.dto.Anime;

import com.folva.moderneastculture.model.dto.OpenPair;

import javafx.beans.property.BooleanProperty;

import javafx.beans.property.SimpleBooleanProperty;

import javafx.beans.property.SimpleObjectProperty;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.layout.Pane;

import javafx.util.Pair;

import org.apache.logging.log4j.Logger;

import java.io.File;

import java.io.IOException;

import java.io.InputStream;

import java.net.URL;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.Optional;

import java.util.ResourceBundle;

import static com.folva.moderneastculture.model.Repository.DB\_IMAGES\_FOLDER;

/\*\*

\* Контроллер елементу керування, який має в собі коротку інформацію про аніме

\*/

public class AnimePresentationControl extends Pane implements Initializable {

@FXML

private Label lAnimePremiereYear;

@FXML

private Label lAnimeName;

@FXML

private Button bDeleteAnime;

@FXML

private Button bEditAnime;

@FXML

private ImageView ivAnimeImage;

/\*\*

\* Об'єкт-властивість, відповідаючий за стан видалення елементу керування

\*/

public final BooleanProperty isDeleted = new SimpleBooleanProperty(false);

/\*\*

\* Об'єкт-властивість, відповідаючий за те, чи оброблений виклик меню інформації про аніме,

\* яке носить в собі даний елемент керування

\*/

public final BooleanProperty infoCallHandled = new SimpleBooleanProperty(true);

/\*\*

\* Об'єкт (логер), який виводить різного роду інформацію відносно цього елементу керування

\*/

public Logger logger;

private SimpleObjectProperty<OpenPair<Boolean, Anime>> editingObjectReference;

private boolean isLoaded = false;

private Anime currentAnime = null;

/\*\*

\* Завантажує даний елемент керування та встановлює залежності від об'єкту-властивості

\* стану авторизації адміністратора

\*/

public void load() {

if (!isLoaded) {

FXMLLoader loader = new FXMLLoader(Main.getResource("/res/fxml\_views/AnimePresentation\_Control.fxml"));

loader.setController(this);

try {

getChildren().add(loader.load());

} catch (IOException e) {

logger.error("Error while loading one of the anime presentation controls", e);

}

bDeleteAnime.visibleProperty().bind(Repository.adminIsAuthorizedProperty);

bEditAnime.visibleProperty().bind(Repository.adminIsAuthorizedProperty);

isLoaded = true;

}

}

/\*\*

\* Встановлює аніме, яке потрібно відображати в цьому елементі керування

\* @param anime аніме для відображення

\*/

public void setAnime(Anime anime) {

currentAnime = anime;

lAnimeName.setText(anime.getName());

lAnimePremiereYear.setText(String.valueOf(anime.getPremiereYear()));

}

/\*\*

\* @return аніме для відображення

\*/

public Anime getAnime() {

return currentAnime;

}

/\*\*

\* Встановлює посилання на об'єкт-властивість стану змінення об'єкту та самого об'єкту аніме

\* @param reference посилання на об'єкт-властивість стану змінення об'єкту та самого об'єкту аніме

\*/

public void setEditingObjectReference(SimpleObjectProperty<OpenPair<Boolean, Anime>> reference) {

editingObjectReference = reference;

}

@Override

public void initialize(URL location, ResourceBundle resources) {

bDeleteAnime.setVisible(false);

}

@FXML

private void onBDeleteAnimeClick() {

try {

Repository.instance.deleteAnimeFromDb(currentAnime);

isDeleted.setValue(true);

} catch (Exception e) {

logger.error("Error while deleting anime '" + lAnimeName.getText() + "': ", e);

}

}

@FXML

private void onBEditAnimeClick() {

editingObjectReference.setValue(new OpenPair<>(true, currentAnime));

}

@FXML

private void onBShowAnimeInfoClick() {

infoCallHandled.setValue(false);

}

@SuppressWarnings("DuplicatedCode")

private void centerImage() {

Image img = ivAnimeImage.getImage();

if (img != null) {

double ratioX = ivAnimeImage.getFitWidth() / img.getWidth();

double ratioY = ivAnimeImage.getFitHeight() / img.getHeight();

double reducCoeff = Math.min(ratioX, ratioY);

double width = img.getWidth() \* reducCoeff;

double height = img.getHeight() \* reducCoeff;

ivAnimeImage.setX((ivAnimeImage.getFitWidth() - width) / 2);

ivAnimeImage.setY((ivAnimeImage.getFitHeight() - height) / 2);

}

}

/\*\*

\* Встановлює зображення, якщо воно є в базі даних та якщо вдалося завантажити в об'єкт Image

\*/

@SuppressWarnings("DuplicatedCode")

public void loadImage() {

ArrayList<Pair<Integer, String>> animeImages = Repository.instance.getAnimeImagePaths(true);

Optional<Pair<Integer, String>> optionalAnimeImagePair = animeImages.stream()

.filter(animeImagePair -> animeImagePair.getKey() == currentAnime.getId()).findFirst();

if (optionalAnimeImagePair.isPresent() &&

Files.exists(Paths.get(DB\_IMAGES\_FOLDER, optionalAnimeImagePair.get().getValue()))) {

Path imagePath = Paths.get(DB\_IMAGES\_FOLDER, optionalAnimeImagePair.get().getValue());

File file = imagePath.toFile();

InputStream fileStream = Main.getFileStream(file);

ivAnimeImage.setImage(new Image(fileStream));

centerImage();

try {

fileStream.close();

} catch (IOException e) {

logger.error("Error while closing the file stream: ", e);

}

}

}

}

## ComicsController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import com.folva.moderneastculture.model.dto.\*;

import com.sun.javafx.collections.ObservableListWrapper;

import javafx.beans.InvalidationListener;

import javafx.beans.binding.DoubleBinding;

import javafx.beans.property.SimpleObjectProperty;

import javafx.beans.property.StringProperty;

import javafx.beans.value.ChangeListener;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.control.\*;

import javafx.scene.layout.VBox;

import javafx.util.Pair;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import java.net.URL;

import java.util.ArrayList;

import java.util.ResourceBundle;

import java.util.stream.Collectors;

/\*\*

\* Контроллер відповідаючий за головне підменю коміксів - де можна продивитися увесь

\* список наявних коміксів та відфільтрувати, викликати меню редагування, додавання,

\* оглядання повної інформації якогось коміксу або видалити взагалі комікс

\*/

public class ComicsController implements Initializable {

private static final Logger logger = LogManager.getLogger(ComicsController.class.getName());

private static final int COMICS\_CONTROLS\_PER\_PAGE = 5;

private static final int COMICS\_CONTROL\_HEIGHT = 100;

@FXML

private TextField tbComicsName;

@FXML

private ScrollPane spComicsList;

@FXML

private Pagination comicsPagination;

@FXML

private ListView<String> lvGenres;

@FXML

private ChoiceBox<String> cbComicsTypes;

@FXML

private ChoiceBox<String> cbComicsStatuses;

@FXML

private TextField tbYearFrom;

@FXML

private TextField tbYearTo;

@FXML

private Button bAddNewComics;

private final ArrayList<OpenPair<Comics, ComicsPresentationControl>> comicsList = new ArrayList<>();

private final SimpleObjectProperty<OpenPair<Boolean, Comics>> editingObject = new SimpleObjectProperty<>();

private Tab tabComics;

private ObservableListWrapper<OpenPair<Comics, ComicsPresentationControl>> filteredComicsList;

@Override

public void initialize(URL location, ResourceBundle resources) {

importComics();

setUpFiltrationData();

setPagination();

bindProperties();

logger.info("ComicsController has been loaded!");

}

/\*\*

\* Встановлює посилання на вкладку коміксів для переключення між підменю

\* @param tabComics посилання на вкладку коміксів

\*/

public void setTabComics(Tab tabComics) {

this.tabComics = tabComics;

}

private void importComics() {

ArrayList<Comics> comics = Repository.instance.getComics(true);

comicsList.clear();

comicsList.addAll(comics.stream().map(localComics ->

new OpenPair<Comics, ComicsPresentationControl>(localComics, null))

.collect(Collectors.toCollection(ArrayList::new)));

filteredComicsList = new ObservableListWrapper<>(new ArrayList<>());

filteredComicsList.addAll(comicsList);

setComicsDisplay();

}

@SuppressWarnings("DuplicatedCode")

private void setComicsDisplay() {

filteredComicsList.addListener((InvalidationListener) c -> {

int newPageCount = (int) Math.ceil(filteredComicsList.size() / (double) COMICS\_CONTROLS\_PER\_PAGE);

newPageCount = (newPageCount > 0) ? newPageCount : 1;

comicsPagination.setPageCount(newPageCount + 2);

comicsPagination.setPageCount(newPageCount);

});

}

private void bindProperties() {

editingObject.addListener((observable, oldValue, newValue) -> {

boolean startEditing = newValue.getKey();

if (startEditing) {

Parent editPane = Main.getForm("EditComicsForm");

try {

EditComicsController editController = (EditComicsController) Main

.getControllerForForm("EditComicsForm");

editController.setEditingObjectReference(editingObject);

} catch (Exception e) {

logger.error("Error while setting called comics control for edit controller: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("problemOccurred"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

tabComics.setContent(editPane);

} else {

Parent comicsContentPane = Main.getForm("ComicsForm");

importComics();

tabComics.setContent(comicsContentPane);

}

});

bAddNewComics.visibleProperty().bind(Repository.adminIsAuthorizedProperty);

tbComicsName.prefWidthProperty().bind(new DoubleBinding() {

{

bind(Repository.adminIsAuthorizedProperty);

}

@Override

protected double computeValue() {

if (Repository.adminIsAuthorizedProperty.get()) {

tbComicsName.setLayoutX(88);

return 700;

} else {

tbComicsName.setLayoutX(22);

return 766;

}

}

});

}

private void setPagination() {

int newPageCount = (int) Math.ceil(comicsList.size() / (double) COMICS\_CONTROLS\_PER\_PAGE);

comicsPagination.setPageCount(newPageCount);

comicsPagination.currentPageIndexProperty().addListener(observable -> spComicsList.setVvalue(0));

comicsPagination.setPageFactory(pageIndex -> {

final int controlCountOnTheLastPage = filteredComicsList.size() - pageIndex \* COMICS\_CONTROLS\_PER\_PAGE;

final int startComicsIndex = pageIndex \* COMICS\_CONTROLS\_PER\_PAGE;

final int endComicsIndex = startComicsIndex + Math.min(COMICS\_CONTROLS\_PER\_PAGE, controlCountOnTheLastPage);

ArrayList<OpenPair<Comics, ComicsPresentationControl>> animeListToPresent = new ArrayList<>(

filteredComicsList.subList(startComicsIndex, endComicsIndex));

ArrayList<ComicsPresentationControl> comicsControls = new ArrayList<>();

for (OpenPair<Comics, ComicsPresentationControl> comicsPair : animeListToPresent) {

if (comicsPair.getValue() != null) {

comicsControls.add(comicsPair.getValue());

continue;

}

Comics comics = comicsPair.getKey();

comics.addInvalidationListener(observable -> {

ComicsPresentationControl presentationControl = comicsPair.getValue();

presentationControl.setComics(comics);

presentationControl.loadImage();

});

ComicsPresentationControl comicsControl = new ComicsPresentationControl();

comicsControl.logger = logger;

comicsControl.load();

comicsControl.setComics(comics);

comicsControl.loadImage();

comicsControl.isDeleted.addListener(observable -> {

logger.info("Comics: " + comics.getName() + " was deleted");

comicsList.remove(comicsPair);

filteredComicsList.remove(comicsPair);

});

comicsControl.infoCallHandled.addListener((observable, oldValue, newValue) -> {

boolean startLooking = !newValue;

if (startLooking) {

Parent editPane = Main.getForm("ComicsInfoForm");

try {

ComicsInfoController infoController = (ComicsInfoController) Main

.getControllerForForm("ComicsInfoForm");

infoController.setComicsControllerReference(comicsControl);

} catch (Exception e) {

logger.error("Error while setting called comics control for info controller: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("problemOccurred"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

tabComics.setContent(editPane);

} else {

Parent comicsContentPane = Main.getForm("ComicsForm");

importComics();

tabComics.setContent(comicsContentPane);

}

});

comicsControl.setEditingObjectReference(editingObject);

comicsControls.add(comicsControl);

comicsPair.setValue(comicsControl);

}

VBox pane = new VBox();

pane.setPrefHeight(COMICS\_CONTROL\_HEIGHT \* COMICS\_CONTROLS\_PER\_PAGE);

pane.getChildren().addAll(comicsControls);

return pane;

});

}

private void setUpFiltrationData() {

setUpGenres();

setComicsTypes();

setComicsStatuses();

setComicsYearBoxes();

}

private void setComicsYearBoxes() {

ChangeListener<String> tbChangeListener = (observable, oldValue, newValue) -> {

if (!newValue.matches("[0-9]{0,4}")) {

((StringProperty) observable).setValue(oldValue);

}

};

tbYearFrom.textProperty().addListener(tbChangeListener);

tbYearTo.textProperty().addListener(tbChangeListener);

}

@SuppressWarnings("DuplicatedCode")

private void setComicsStatuses() {

ArrayList<String> comicsStatuses = new ArrayList<>();

for (Status status : Status.values()) {

comicsStatuses.add(Repository.instance.getNamesBundleValue(status.name));

}

comicsStatuses.add(Repository.instance.getNamesBundleValue("ignoreValue"));

cbComicsStatuses.getItems().addAll(comicsStatuses);

}

private void setComicsTypes() {

ArrayList<String> comicsTypes = new ArrayList<>();

for (Comics.Type value : Comics.Type.values()) {

comicsTypes.add(Repository.instance.getNamesBundleValue(value.name));

}

comicsTypes.add(Repository.instance.getNamesBundleValue("ignoreValue"));

cbComicsTypes.getItems().addAll(comicsTypes);

}

private void setUpGenres() {

lvGenres.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

lvGenres.getItems().addAll(Repository.instance.getGenres(false).stream()

.map(genre -> genre.name)

.collect(Collectors.toCollection(ArrayList::new)));

}

@SuppressWarnings("DuplicatedCode")

@FXML

private void onBFilter() {

if (!filterFieldsAreValidated()) return;

filteredComicsList.clear();

ArrayList<Pair<Comics, Genre>> comicsGenreMap = Repository.instance.getComicsGenreMap(true);

ArrayList<String> selectedGenres = new ArrayList<>(lvGenres.getSelectionModel().getSelectedItems());

String selectedType = cbComicsTypes.getSelectionModel().getSelectedItem();

String selectedStatus = cbComicsStatuses.getSelectionModel().getSelectedItem();

int selectedYearFrom = tbYearFrom.getText().isEmpty() ? 0 : Integer.parseInt(tbYearFrom.getText());

int selectedYearTo = tbYearTo.getText().isEmpty() ? 9999 : Integer.parseInt(tbYearTo.getText());

filteredComicsList.addAll(

comicsList

.stream()

.filter(comicsPair -> {

Comics comics = comicsPair.getKey();

boolean containGenre = false;

boolean correctType = false;

boolean correctStatus = false;

boolean correctYear = false;

if (selectedGenres.isEmpty() ||

comicsGenreMap.stream().anyMatch(comicsGenrePair -> {

boolean currentComics = (comicsGenrePair.getKey().equals(comics));

boolean containsAtLeastOneSelectedGenre = selectedGenres.contains(comicsGenrePair.getValue().name);

return currentComics && containsAtLeastOneSelectedGenre;

}))

containGenre = true;

if (selectedType == null ||

selectedType.equals(Repository.instance.getNamesBundleValue("ignoreValue")) ||

selectedType.equals(Repository.instance.getNamesBundleValue(comics.getType().name)))

correctType = true;

if (selectedStatus == null ||

selectedStatus.equals(Repository.instance.getNamesBundleValue("ignoreValue")) ||

selectedStatus.equals(Repository.instance.getNamesBundleValue(comics.getStatus().name)))

correctStatus = true;

if (comics.getPremiereYear() >= selectedYearFrom && comics.getPremiereYear() <= selectedYearTo)

correctYear = true;

return containGenre && correctType && correctStatus && correctYear;

})

.collect(Collectors.toCollection(ArrayList::new)));

}

@SuppressWarnings("DuplicatedCode")

private boolean filterFieldsAreValidated() {

int selectedYearFrom = tbYearFrom.getText().isEmpty() ? 0 : Integer.parseInt(tbYearFrom.getText());

int selectedYearTo = tbYearTo.getText().isEmpty() ? 9999 : Integer.parseInt(tbYearTo.getText());

if (selectedYearFrom > selectedYearTo) {

Main.warningAlert.setContentText(Repository.instance.getNamesBundleValue("yearValidationIncorrect"));

Main.warningAlert.show();

return false;

}

return true;

}

@SuppressWarnings("DuplicatedCode")

@FXML

private void onBSearchByNameClick() {

String comicsName = tbComicsName.getText();

if (comicsName.matches("[ ]+")) return;

filteredComicsList.clear();

filteredComicsList.addAll(comicsList.stream().filter(comics -> {

boolean nameContainsSearchedSymbols = comics.getKey().getName().contains(comicsName);

boolean anyAltNameContainsSearchedSymbols = comics.getKey().getAltNames().stream()

.anyMatch(altName -> altName.contains(comicsName));

return nameContainsSearchedSymbols || anyAltNameContainsSearchedSymbols;

})

.collect(Collectors.toCollection(ArrayList::new)));

}

@FXML

private void onBAddNewComicsClick() {

editingObject.set(new OpenPair<>(true, null));

}

}

## ComicsInfoController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import com.folva.moderneastculture.model.dto.Anime;

import com.folva.moderneastculture.model.dto.Comics;

import com.folva.moderneastculture.model.dto.Genre;

import javafx.beans.binding.DoubleBinding;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Label;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.util.Pair;

import java.net.URL;

import java.util.ArrayList;

import java.util.Optional;

import java.util.ResourceBundle;

import java.util.stream.Collectors;

/\*\*

\* Контроллер, який відповідає за меню відображення повної інформації про комікс

\*/

@SuppressWarnings("DuplicatedCode")

public class ComicsInfoController implements Initializable {

@FXML

private Label lComicsName;

@FXML

private Label lComicsAltNames;

@FXML

private Label lStatus;

@FXML

private Label lYear;

@FXML

private Label lGenresTitle;

@FXML

private Label lGenres;

@FXML

private Label lSource;

@FXML

private Label lAuthorTitle;

@FXML

private Label lAuthor;

@FXML

private Label lType;

@FXML

private Label lChapterCount;

@FXML

private Label lDescription;

@FXML

private ImageView ivFirstImage;

private ComicsPresentationControl comicsPresentationControl;

@Override

public void initialize(URL location, ResourceBundle resources) {

lGenres.maxWidthProperty().bind(new DoubleBinding() {

private static final int MAX\_LEFT\_PART\_WIDTH = 517;

private static final int COLON\_WIDTH = 10;

{ bind(lGenresTitle.widthProperty()); }

@SuppressWarnings("UnnecessaryLocalVariable")

@Override

protected double computeValue() {

double newWidth = MAX\_LEFT\_PART\_WIDTH - COLON\_WIDTH - lGenresTitle.getWidth();

return newWidth;

}

});

}

/\*\*

\* @param comicsPresentationControl посилання на об'єкт елементу керування, який викликав це меню.

\* Коли дане меню буде закриватися, у властивості цього елементу

\* керування змінять стан на true. Також дозволяє вилучити комікс

\* для відображення

\*/

public void setComicsControllerReference(ComicsPresentationControl comicsPresentationControl) {

this.comicsPresentationControl = comicsPresentationControl;

Comics comics = comicsPresentationControl.getComics();

lComicsName.setText(comics.getName());

Optional<String> collectedAltNames = comics.getAltNames()

.stream()

.reduce((s, s2) -> s.concat("\n").concat(s2));

if (!collectedAltNames.isPresent()) {

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("fatalError"));

Main.errorAlert.showAndWait();

System.exit(-1);

}

lComicsAltNames.setText(collectedAltNames.get());

lStatus.setText(Repository.instance.getNamesBundleValue(comics.getStatus().name));

lYear.setText(String.valueOf(comics.getPremiereYear()));

ArrayList<Pair<Comics, Genre>> comicsGenreMap = Repository.instance.getComicsGenreMap(true);

ArrayList<String> currentComicsGenreNames = comicsGenreMap

.stream()

.filter(comicsGenrePair -> comicsGenrePair.getKey().getId() == comics.getId())

.map(Pair::getValue)

.map(genre -> genre.name)

.collect(Collectors.toCollection(ArrayList::new));

String genresString = currentComicsGenreNames.stream()

.reduce("", (s, s2) -> s.concat(", ".concat(s2)))

.substring(2);

lGenres.setText(genresString);

lSource.setText(Repository.instance.getNamesBundleValue(comics.getSource().name));

lAuthorTitle.setText(Repository.instance.getNamesBundleValue(comics

.getAuthor().getType().name().toLowerCase()));

lAuthor.setText(comics.getAuthor().getName());

lType.setText(Repository.instance.getNamesBundleValue(comics.getType().name));

lChapterCount.setText(String.valueOf(comics.getChapterCount()));

lDescription.setText(comics.getDescription());

ArrayList<String> imagePaths = Repository.instance.getComicsImagePaths(true)

.stream()

.filter(comicsImagePathPair -> comicsImagePathPair.getKey() == comics.getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

ivFirstImage.setImage(Repository.getNoImageImage());

if (imagePaths.size() != 0) {

String firstImageName = imagePaths.get(0);

Image firstImage = Repository.loadImage(firstImageName);

if (firstImage != null) ivFirstImage.setImage(firstImage);

}

}

@FXML

private void onBExitEditWindowClick() {

comicsPresentationControl.infoCallHandled.setValue(true);

}

}

## ComicsPresentationControl

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import com.folva.moderneastculture.model.dto.Comics;

import com.folva.moderneastculture.model.dto.OpenPair;

import javafx.beans.property.BooleanProperty;

import javafx.beans.property.SimpleBooleanProperty;

import javafx.beans.property.SimpleObjectProperty;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.scene.layout.Pane;

import javafx.util.Pair;

import org.apache.logging.log4j.Logger;

import java.io.File;

import java.io.IOException;

import java.io.InputStream;

import java.net.URL;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.ArrayList;

import java.util.Optional;

import java.util.ResourceBundle;

import static com.folva.moderneastculture.model.Repository.DB\_IMAGES\_FOLDER;

/\*\*

\* Контроллер елементу керування, який має в собі коротку інформацію про комікс

\*/

public class ComicsPresentationControl extends Pane implements Initializable {

@FXML

private Label lComicsPremiereYear;

@FXML

private Label lComicsName;

@FXML

private Button bDeleteComics;

@FXML

private Button bEditComics;

@FXML

private ImageView ivComicsImage;

/\*\*

\* Об'єкт-властивість, відповідаючий за стан видалення елементу керування

\*/

public final BooleanProperty isDeleted = new SimpleBooleanProperty(false);

/\*\*

\* Об'єкт-властивість, відповідаючий за те, чи оброблений виклик меню інформації про комікс,

\* яке носить в собі даний елемент керування

\*/

public final BooleanProperty infoCallHandled = new SimpleBooleanProperty(true);

/\*\*

\* Об'єкт (логер), який виводить різного роду інформацію відносно цього елементу керування

\*/

public Logger logger;

private SimpleObjectProperty<OpenPair<Boolean, Comics>> editingObjectReference;

private boolean isLoaded = false;

private Comics currentComics = null;

/\*\*

\* Завантажує даний елемент керування та встановлює залежності від об'єкту-властивості

\* стану авторизації адміністратора

\*/

public void load() {

if (!isLoaded) {

FXMLLoader loader = new FXMLLoader(Main.getResource("/res/fxml\_views/ComicsPresentation\_Control.fxml"));

loader.setController(this);

try {

getChildren().add(loader.load());

} catch (IOException e) {

logger.error("Error while loading one of the comics presentation controls", e);

}

bDeleteComics.visibleProperty().bind(Repository.adminIsAuthorizedProperty);

bEditComics.visibleProperty().bind(Repository.adminIsAuthorizedProperty);

isLoaded = true;

}

}

/\*\*

\* Встановлює комікс, яке потрібно відображати в цьому елементі керування

\* @param comics комік для відображення

\*/

public void setComics(Comics comics) {

currentComics = comics;

lComicsName.setText(comics.getName());

lComicsPremiereYear.setText(String.valueOf(comics.getPremiereYear()));

}

/\*\*

\* @return комікс для відображення

\*/

public Comics getComics() {

return currentComics;

}

/\*\*

\* Встановлює посилання на об'єкт-властивість стану змінення об'єкту та самого об'єкту коміксу

\* @param reference посилання на об'єкт-властивість стану змінення об'єкту та самого об'єкту коміксу

\*/

public void setEditingObjectReference(SimpleObjectProperty<OpenPair<Boolean, Comics>> reference) {

editingObjectReference = reference;

}

@Override

public void initialize(URL location, ResourceBundle resources) {

bDeleteComics.setVisible(false);

}

@FXML

private void onBDeleteComicsClick() {

try {

Repository.instance.deleteComicsFromDb(currentComics);

isDeleted.setValue(true);

} catch (Exception e) {

logger.error("Error while deleting comics '" + lComicsName.getText() + "': ", e);

}

}

@FXML

private void onBEditComicsClick() {

editingObjectReference.setValue(new OpenPair<>(true, currentComics));

}

@FXML

private void onBShowComicsInfoClick() {

infoCallHandled.setValue(false);

}

@SuppressWarnings("DuplicatedCode")

private void centerImage() {

Image img = ivComicsImage.getImage();

if (img != null) {

double ratioX = ivComicsImage.getFitWidth() / img.getWidth();

double ratioY = ivComicsImage.getFitHeight() / img.getHeight();

double reducCoeff = Math.min(ratioX, ratioY);

double width = img.getWidth() \* reducCoeff;

double height = img.getHeight() \* reducCoeff;

ivComicsImage.setX((ivComicsImage.getFitWidth() - width) / 2);

ivComicsImage.setY((ivComicsImage.getFitHeight() - height) / 2);

}

}

/\*\*

\* Встановлює зображення, якщо воно є в базі даних та якщо вдалося завантажити в об'єкт Image

\*/

@SuppressWarnings("DuplicatedCode")

public void loadImage() {

ArrayList<Pair<Integer, String>> comicsImages = Repository.instance.getComicsImagePaths(true);

Optional<Pair<Integer, String>> optionalComicsImagePair = comicsImages.stream()

.filter(comicsImagePair -> comicsImagePair.getKey() == currentComics.getId()).findFirst();

if (optionalComicsImagePair.isPresent() &&

Files.exists(Paths.get(DB\_IMAGES\_FOLDER, optionalComicsImagePair.get().getValue()))) {

Path imagePath = Paths.get(DB\_IMAGES\_FOLDER, optionalComicsImagePair.get().getValue());

File file = imagePath.toFile();

Image image = Repository.loadImage(file.getName());

if (image != null) {

ivComicsImage.setImage(image);

centerImage();

}

}

}

}

## EditAnimeController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import com.folva.moderneastculture.model.dto.\*;

import javafx.beans.property.ObjectProperty;

import javafx.beans.property.SimpleObjectProperty;

import javafx.beans.property.StringProperty;

import javafx.beans.value.ChangeListener;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.stage.FileChooser;

import javafx.util.Pair;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import java.io.File;

import java.io.IOException;

import java.io.InputStream;

import java.net.URL;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.\*;

import java.util.stream.Collectors;

import static com.folva.moderneastculture.model.Repository.DB\_IMAGES\_FOLDER;

import static com.folva.moderneastculture.model.Repository.loadImage;

/\*\*

\* Підменю редагування аніме. Відповідає за зміну та добавлення нових

\* аніме

\*/

@SuppressWarnings("DuplicatedCode")

public class EditAnimeController implements Initializable {

private static final Logger logger = LogManager.getLogger(EditAnimeController.class.getName());

@FXML

private TextArea taName;

@FXML

private TextArea taDescription;

@FXML

private TextArea taAltNames;

@FXML

private TextField tbEpisodeCount;

@FXML

private TextField tbPremiereYear;

@FXML

private ImageView ivFirstImage;

@FXML

private ListView<String> lvAgeRatings;

@FXML

private ListView<String> lvSources;

@FXML

private ListView<String> lvTypes;

@FXML

private ListView<String> lvPremiereSeasons;

@FXML

private ListView<String> lvGenres;

@FXML

private ListView<String> lvStatus;

@FXML

private ListView<String> lvAuthors;

@FXML

private ImageView ivCurrentGalleryImage;

@FXML

private Button bDeleteImage;

@FXML

private Button bPreviousImage;

@FXML

private Button bNextImage;

private final ObjectProperty<ObjectProperty<OpenPair<Boolean, Anime>>> editingObjectReference =

new SimpleObjectProperty<>();

private final ArrayList<String> currentAnimeImagePaths = new ArrayList<>();

private final Map<String, Image> cachedGalleryImages = new HashMap<>();

private int currentImageIndex = 0;

private boolean isNew = false;

/\*\*

\* Встановлює посилання на об'єкт-властивість стану змінення об'єкту та самого об'єкту аніме

\* @param reference посилання на об'єкт-властивість стану змінення об'єкту та самого об'єкту аніме

\*/

public void setEditingObjectReference(SimpleObjectProperty<OpenPair<Boolean, Anime>> reference) {

editingObjectReference.set(reference);

currentAnimeImagePaths.clear();

setAnimeDetails();

}

private void setAnimeDetails() {

Anime currentAnime = editingObjectReference.get().get().getValue();

if (currentAnime == null) {

isNew = true;

ivFirstImage.setImage(Repository.getNoImageImage());

return;

}

ArrayList<Pair<Integer, String>> animeImagePaths = Repository.instance.getAnimeImagePaths(true);

ArrayList<String> currentAnimeImagePaths = animeImagePaths.stream().filter(animeImage ->

animeImage.getKey() == currentAnime.getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

this.currentAnimeImagePaths.addAll(currentAnimeImagePaths);

if (currentAnimeImagePaths.isEmpty()) {

ivFirstImage.setImage(Repository.getNoImageImage());

} else {

String firstImageName = currentAnimeImagePaths.get(0);

Image image = loadImage(firstImageName);

if (image == null) {

image = Repository.getNoImageImage();

}

ivFirstImage.setImage(image);

}

centerImage(ivFirstImage);

taName.setText(currentAnime.getName());

taDescription.setText(currentAnime.getDescription());

String concatenatedString = currentAnime.getAltNames().stream()

.reduce("", (s, s2) -> s.concat(";").concat(s2));

String altNamesString = (concatenatedString.length() > 0) ? concatenatedString.substring(1) : "";

taAltNames.setText(altNamesString); // remove 1 symbol because it will be ";"

tbEpisodeCount.setText(String.valueOf(currentAnime.getEpisodeCount()));

tbPremiereYear.setText(String.valueOf(currentAnime.getPremiereYear()));

lvAgeRatings.getSelectionModel().select(currentAnime.getAgeRating().name);

lvSources.getSelectionModel().select(Repository.instance.getNamesBundleValue(currentAnime.getSource().name));

lvTypes.getSelectionModel().select(Repository.instance.getNamesBundleValue(currentAnime.getType().name));

lvPremiereSeasons.getSelectionModel().select(

Repository.instance.getNamesBundleValue(currentAnime.getPremiereSeason().name));

ArrayList<Genre> currentAnimeGenres = Repository.instance.getAnimeGenreMap(true).stream()

.filter(animeGenrePair -> animeGenrePair.getKey().getId() == currentAnime.getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

ArrayList<Integer> genresIdsList = currentAnimeGenres.stream().map(genre ->

genre.id - 1).collect(Collectors.toCollection(ArrayList::new));

int[] genresIds = new int[genresIdsList.size()];

for (int i = 0; i < genresIdsList.size(); i++) {

genresIds[i] = genresIdsList.get(i);

}

lvGenres.getSelectionModel().clearSelection();

lvGenres.getSelectionModel().selectIndices(-1, genresIds);

lvStatus.getSelectionModel().select(Repository.instance.getNamesBundleValue(currentAnime.getStatus().name));

lvAuthors.getSelectionModel().select(currentAnime.getAuthor().getName());

bNextImage.setDisable(false);

bPreviousImage.setDisable(false);

bDeleteImage.setDisable(false);

if (currentAnimeImagePaths.size() <= 1) {

bNextImage.setDisable(true);

bPreviousImage.setDisable(true);

}

if (currentAnimeImagePaths.size() == 0) {

bDeleteImage.setDisable(true);

ivCurrentGalleryImage.setImage(Repository.getNoImageImage());

return;

}

String firstImageName = currentAnimeImagePaths.get(0);

Image galleryImage = Repository.loadImage(firstImageName);

if (galleryImage != null) {

ivCurrentGalleryImage.setImage(galleryImage);

centerImage(ivCurrentGalleryImage);

} else {

ivCurrentGalleryImage.setImage(Repository.getNoImageImage());

}

}

@FXML

private void onBExitEditWindowClick() {

removeRedundantImages();

ObjectProperty<OpenPair<Boolean, Anime>> reference = editingObjectReference.get();

reference.set(new OpenPair<>(false, null));

}

@SuppressWarnings("ConstantConditions")

private void removeRedundantImages() {

ArrayList<String> dbAnimeImagePaths = Repository.instance.getAnimeImagePaths(true).stream()

.map(Pair::getValue).collect(Collectors.toCollection(ArrayList::new));

Repository.instance.handleDirectoryExistence();

String[] list = Paths.get(DB\_IMAGES\_FOLDER).toFile().list();

ArrayList<String> folderAnimeImagePaths = new ArrayList<>(Arrays.asList(list));

folderAnimeImagePaths.forEach(folderAnimeImagePath -> {

if (!dbAnimeImagePaths.contains(folderAnimeImagePath)) {

try {

Files.delete(Paths.get(DB\_IMAGES\_FOLDER, folderAnimeImagePath));

} catch (IOException e) {

logger.error("Error while deleting file " + folderAnimeImagePath + ": ", e);

}

}

});

}

@SuppressWarnings("OptionalGetWithoutIsPresent")

@FXML

private void onBSaveClick() {

if (!fieldsAreValidated()) return;

String newName = taName.getText();

String newDescription = taDescription.getText();

ArrayList<String> newAltNames = Arrays.stream(taAltNames.getText().split(";"))

.filter(s -> !s.matches("[ ]\*")).collect(Collectors.toCollection(ArrayList::new));

int newEpisodeCount = Integer.parseInt(tbEpisodeCount.getText());

int newPremiereYear = Integer.parseInt(tbPremiereYear.getText());

String newAgeRatingString = lvAgeRatings.getSelectionModel().getSelectedItem();

Optional<AgeRating> optionalAgeRating = Arrays.stream(AgeRating.values()).filter(ageRating ->

ageRating.name.equalsIgnoreCase(newAgeRatingString)).findFirst();

AgeRating newAgeRating = optionalAgeRating.orElse(AgeRating.UN);

String newSourceString = lvSources.getSelectionModel().getSelectedItem();

Optional<Anime.Source> optionalSource = Arrays.stream(Anime.Source.values()).filter(source ->

Repository.instance.getNamesBundleValue(source.name).equalsIgnoreCase(newSourceString)).findFirst();

Anime.Source newSource = optionalSource.orElse(Anime.Source.OTHER);

String newTypeString = lvTypes.getSelectionModel().getSelectedItem();

Optional<Anime.Type> optionalType = Arrays.stream(Anime.Type.values()).filter(type ->

Repository.instance.getNamesBundleValue(type.name).equalsIgnoreCase(newTypeString)).findFirst();

Anime.Type newType = optionalType.orElse(Anime.Type.SERIES);

String newPremiereSeasonString = lvPremiereSeasons.getSelectionModel().getSelectedItem();

Optional<YearSeason> optionalYearSeason = Arrays.stream(YearSeason.values()).filter(yearSeason ->

Repository.instance.getNamesBundleValue(yearSeason.name)

.equalsIgnoreCase(newPremiereSeasonString)).findFirst();

YearSeason newPremiereSeason = optionalYearSeason.orElse(YearSeason.UNDEFINED);

String newAuthorString = lvAuthors.getSelectionModel().getSelectedItem();

Optional<Author> optionalAuthor = Repository.instance.getAuthors(false).stream().filter(author ->

author.getName().equalsIgnoreCase(newAuthorString)).findFirst();

Author newAuthor;

try {

newAuthor = optionalAuthor.get();

} catch (Exception e) {

logger.fatal("Error while finding author from repository: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("fatalError"));

Main.errorAlert.showAndWait();

System.exit(-1);

return;

}

ArrayList<String> newGenresStrings = new ArrayList<>(lvGenres.getSelectionModel().getSelectedItems());

ArrayList<String> newGenresStringsLeftToFind = new ArrayList<>(lvGenres.getSelectionModel().getSelectedItems());

ArrayList<Genre> genres = Repository.instance.getGenres(false);

ArrayList<Genre> newGenres = new ArrayList<>();

for (String newGenreString : newGenresStrings) {

Optional<Genre> optionalGenre = genres.stream().filter(genre -> {

Optional<String> anyGenreWithTheSameName = newGenresStringsLeftToFind.stream()

.filter(localNewGenreStringLeft -> localNewGenreStringLeft.equalsIgnoreCase(genre.name))

.findFirst();

return anyGenreWithTheSameName.isPresent();

}).findFirst();

newGenres.add(optionalGenre.get());

newGenresStringsLeftToFind.remove(newGenreString);

}

String newStatusString = lvStatus.getSelectionModel().getSelectedItem();

Optional<Status> optionalStatus = Arrays.stream(Status.values()).filter(status ->

Repository.instance.getNamesBundleValue(status.name).equalsIgnoreCase(newStatusString)).findFirst();

Status newStatus = optionalStatus.orElse(Status.ONGOING);

if (isNew) {

Optional<Anime> animeWithMaxId = Repository.instance.getAnimes(true).stream()

.max(Comparator.comparingInt(Anime::getId));

int nextId = animeWithMaxId.get().getId() + 1;

logger.trace(nextId);

Anime newAnime = Anime.Builder.newBuilder()

.setId(nextId)

.setAuthor(newAuthor)

.setName(newName)

.setDescription(newDescription)

.setEpisodeCount(newEpisodeCount)

.setPremiereYear(newPremiereYear)

.setAgeRating(newAgeRating)

.setSource(newSource)

.setType(newType)

.setPremiereSeason(newPremiereSeason)

.setAltNames(newAltNames)

.setStatus(newStatus)

.build();

Repository.instance.insertNewAnime(newAnime);

editingObjectReference.get().get().setValue(newAnime);

} else {

Anime currentAnime = editingObjectReference.get().get().getValue();

currentAnime.setName(newName);

currentAnime.setAuthor(newAuthor);

currentAnime.setDescription(newDescription);

currentAnime.setAltNames(newAltNames);

currentAnime.setEpisodeCount(newEpisodeCount);

currentAnime.setPremiereYear(newPremiereYear);

currentAnime.setAgeRating(newAgeRating);

currentAnime.setSource(newSource);

currentAnime.setType(newType);

currentAnime.setPremiereSeason(newPremiereSeason);

currentAnime.setStatus(newStatus);

Repository.instance.updateAnime(currentAnime);

currentAnime.invalidate();

}

int currentAnimeId = editingObjectReference.get().get().getValue().getId();

Pair<Anime, ArrayList<Genre>> animeGenres = new Pair<>(

editingObjectReference.get().get().getValue(), newGenres);

Pair<Anime, ArrayList<String>> animeImages = new Pair<>(

editingObjectReference.get().get().getValue(), currentAnimeImagePaths);

Repository.instance.updateAnimeImages(animeImages);

Repository.instance.updateAnimeGenres(animeGenres);

Repository.instance.updateAnimeCache();

logger.info(String.format("Anime with id %d was altered", currentAnimeId));

}

private boolean fieldsAreValidated() {

String newName = taName.getText();

int newPremiereYear = Integer.parseInt(tbPremiereYear.getText());

String selectedAgeRating = lvAgeRatings.getSelectionModel().getSelectedItem();

String selectedSource = lvSources.getSelectionModel().getSelectedItem();

String selectedType = lvTypes.getSelectionModel().getSelectedItem();

String selectedPremiereSeason = lvPremiereSeasons.getSelectionModel().getSelectedItem();

String selectedStatus = lvStatus.getSelectionModel().getSelectedItem();

String selectedAuthor = lvAuthors.getSelectionModel().getSelectedItem();

boolean nameIsNotEmpty = !newName.matches("[ ]\*");

boolean correctPremiereYear = newPremiereYear >= Repository.FIRST\_ANIME\_PREMIERE\_YEAR;

boolean correctAgeSelected = (selectedAgeRating != null);

boolean correctSourceSelected = (selectedSource != null);

boolean correctTypeSelected = (selectedType != null);

boolean correctPremiereSeasonSelected = (selectedPremiereSeason != null);

boolean correctStatusSelected = (selectedStatus != null);

boolean correctAuthorSelected = (selectedAuthor != null);

StringBuilder warningBuilder = new StringBuilder();

if (!nameIsNotEmpty) warningBuilder.append(Repository.instance.getNamesBundleValue("nameIsEmptyError"));

if (!correctPremiereYear)

warningBuilder.append(Repository.instance.getNamesBundleValue("animePremiereYearIsWrongError"));

if (!correctAgeSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("ageRatingIsNotSelectedError"));

if (!correctSourceSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("sourceIsNotSelectedError"));

if (!correctTypeSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("typeIsNotSelectedError"));

if (!correctPremiereSeasonSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("premiereSeasonIsNotSelectedError"));

if (!correctStatusSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("statusIsNotSelectedError"));

if (!correctAuthorSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("authorIsNotSelectedError"));

warningBuilder.trimToSize();

if (warningBuilder.length() > 0) {

Main.warningAlert.setContentText(warningBuilder.toString());

Main.warningAlert.show();

}

return nameIsNotEmpty && correctPremiereYear && correctAgeSelected &&

correctSourceSelected && correctTypeSelected && correctPremiereSeasonSelected &&

correctStatusSelected && correctAuthorSelected;

}

private void centerImage(ImageView imageView) {

Image img = imageView.getImage();

if (img != null) {

double ratioX = imageView.getFitWidth() / img.getWidth();

double ratioY = imageView.getFitHeight() / img.getHeight();

double reducCoeff = Math.min(ratioX, ratioY);

double width = img.getWidth() \* reducCoeff;

double height = img.getHeight() \* reducCoeff;

imageView.setX((imageView.getFitWidth() - width) / 2);

imageView.setY((imageView.getFitHeight() - height) / 2);

}

}

@FXML

private void onBChooseFirstImageClick() {

FileChooser fileChooser = new FileChooser();

File file = fileChooser.showOpenDialog(Main.stage);

if (file == null) return;

Path destinationImagePath = Repository.instance.imageWasCopiedToDbImgFolder(file);

if (destinationImagePath == null) return;

InputStream fileStream = Main.getFileStream(destinationImagePath.toFile());

Image newImage = new Image(fileStream);

ivFirstImage.setImage(newImage);

centerImage(ivFirstImage);

currentAnimeImagePaths.add(destinationImagePath.getFileName().toString());

if (currentAnimeImagePaths.size() == 1) {

ivCurrentGalleryImage.setImage(newImage);

centerImage(ivCurrentGalleryImage);

bDeleteImage.setDisable(false);

}

try {

fileStream.close();

} catch (IOException e) {

logger.error("Error while closing the file stream: ", e);

}

}

private void setGalleryImage() {

String imageName = currentAnimeImagePaths.get(currentImageIndex);

Image image;

if (cachedGalleryImages.containsKey(imageName)) {

image = cachedGalleryImages.get(imageName);

} else {

image = Repository.loadImage(imageName);

cachedGalleryImages.put(imageName, image);

}

if (image == null) {

image = Repository.getNoImageImage();

}

ivCurrentGalleryImage.setImage(image);

centerImage(ivCurrentGalleryImage);

}

@FXML

private void onBPreviousImageClick() {

if (currentImageIndex == 0) {

currentImageIndex = currentAnimeImagePaths.size() - 1;

} else {

currentImageIndex--;

}

setGalleryImage();

}

@FXML

private void onBNextImageClick() {

if (currentImageIndex == currentAnimeImagePaths.size() - 1) {

currentImageIndex = 0;

} else {

currentImageIndex++;

}

setGalleryImage();

}

@FXML

private void onBAddImageClick() {

FileChooser fileChooser = new FileChooser();

File newImage = fileChooser.showOpenDialog(Main.stage);

if (newImage == null) return;

Path destinationImagePath = Repository.instance.imageWasCopiedToDbImgFolder(newImage);

if (destinationImagePath == null) return;

String newImageFileName = destinationImagePath.getFileName().toString();

currentAnimeImagePaths.add(newImageFileName);

currentImageIndex = currentAnimeImagePaths.indexOf(newImageFileName);

setGalleryImage();

if (currentAnimeImagePaths.size() == 1) {

bDeleteImage.setDisable(false);

} else if (currentAnimeImagePaths.size() == 2) {

bPreviousImage.setDisable(false);

bNextImage.setDisable(false);

}

logger.info("Image '" + newImageFileName + "' was added");

}

@FXML

private void onBDeleteImageClick() {

String imageNameToDelete = currentAnimeImagePaths.get(currentImageIndex);

currentAnimeImagePaths.remove(imageNameToDelete);

if (currentAnimeImagePaths.size() == 0) {

bDeleteImage.setDisable(true);

bNextImage.setDisable(true);

bPreviousImage.setDisable(true);

ivCurrentGalleryImage.setImage(Repository.getNoImageImage());

ivFirstImage.setImage(Repository.getNoImageImage());

centerImage(ivCurrentGalleryImage);

return;

} else if (currentImageIndex == currentAnimeImagePaths.size()) {

currentImageIndex--;

}

setGalleryImage();

logger.info("Image '" + imageNameToDelete + "' was deleted");

}

@Override

public void initialize(URL location, ResourceBundle resources) {

setUpTextBoxes();

setUpListViews();

}

private void setUpListViews() {

lvAgeRatings.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvAgeRatings.getItems().clear();

lvAgeRatings.getItems().addAll(Arrays.stream(AgeRating.values()).map(ageRating ->

ageRating.name).collect(Collectors.toCollection(ArrayList::new)));

lvSources.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvSources.getItems().clear();

lvSources.getItems().addAll(Arrays.stream(Anime.Source.values()).map(source ->

Repository.instance.getNamesBundleValue(source.name))

.collect(Collectors.toCollection(ArrayList::new)));

lvTypes.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvTypes.getItems().clear();

lvTypes.getItems().addAll(Arrays.stream(Anime.Type.values()).map(type ->

Repository.instance.getNamesBundleValue(type.name))

.collect(Collectors.toCollection(ArrayList::new)));

lvPremiereSeasons.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvPremiereSeasons.getItems().clear();

lvPremiereSeasons.getItems().addAll(Arrays.stream(YearSeason.values()).map(season ->

Repository.instance.getNamesBundleValue(season.name))

.collect(Collectors.toCollection(ArrayList::new)));

lvGenres.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

lvGenres.getItems().clear();

lvGenres.getItems().addAll(Repository.instance.getGenres(false).stream().map(genre ->

genre.name).collect(Collectors.toCollection(ArrayList::new)));

lvStatus.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvStatus.getItems().clear();

lvStatus.getItems().addAll(Arrays.stream(Status.values()).map(status ->

Repository.instance.getNamesBundleValue(status.name))

.collect(Collectors.toCollection(ArrayList::new)));

lvAuthors.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvAuthors.getItems().clear();

lvAuthors.getItems().addAll(Repository.instance.getAuthors(false).stream().map(Author::getName)

.collect(Collectors.toCollection(ArrayList::new)));

}

private void setUpTextBoxes() {

ChangeListener<String> tbChangeListener = (observable, oldValue, newValue) -> {

if (newValue.isEmpty()) {

((StringProperty) observable).setValue(String.valueOf(0));

} else if (!newValue.matches("[0-9]{0,4}")) {

((StringProperty) observable).setValue(oldValue);

}

};

tbEpisodeCount.textProperty().addListener(tbChangeListener);

tbPremiereYear.textProperty().addListener(tbChangeListener);

}

}

## EditComicsController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import com.folva.moderneastculture.model.dto.\*;

import javafx.beans.property.ObjectProperty;

import javafx.beans.property.SimpleObjectProperty;

import javafx.beans.property.StringProperty;

import javafx.beans.value.ChangeListener;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.\*;

import javafx.scene.image.Image;

import javafx.scene.image.ImageView;

import javafx.stage.FileChooser;

import javafx.util.Pair;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import java.io.File;

import java.io.IOException;

import java.io.InputStream;

import java.net.URL;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.\*;

import java.util.stream.Collectors;

import static com.folva.moderneastculture.model.Repository.DB\_IMAGES\_FOLDER;

import static com.folva.moderneastculture.model.Repository.loadImage;

/\*\*

\* Підменю редагування коміксів. Відповідає за зміну та добавлення нових

\* коміксів

\*/

@SuppressWarnings("DuplicatedCode")

public class EditComicsController implements Initializable {

private static final Logger logger = LogManager.getLogger(EditComicsController.class.getName());

@FXML

private TextArea taName;

@FXML

private TextArea taDescription;

@FXML

private TextArea taAltNames;

@FXML

private TextField tbChapterCount;

@FXML

private TextField tbPremiereYear;

@FXML

private ImageView ivFirstImage;

@FXML

private ListView<String> lvSources;

@FXML

private ListView<String> lvTypes;

@FXML

private ListView<String> lvGenres;

@FXML

private ListView<String> lvStatus;

@FXML

private ListView<String> lvAuthors;

@FXML

private ImageView ivCurrentGalleryImage;

@FXML

private Button bDeleteImage;

@FXML

private Button bPreviousImage;

@FXML

private Button bNextImage;

private final ObjectProperty<ObjectProperty<OpenPair<Boolean, Comics>>> editingObjectReference =

new SimpleObjectProperty<>();

private final ArrayList<String> currentComicsImagePaths = new ArrayList<>();

private final Map<String, Image> cachedGalleryImages = new HashMap<>();

private int currentImageIndex = 0;

private boolean isNew = false;

/\*\*

\* Встановлює посилання на об'єкт-властивість стану змінення об'єкту та самого об'єкту коміксу

\* @param reference посилання на об'єкт-властивість стану змінення об'єкту та самого об'єкту коміксу

\*/

public void setEditingObjectReference(SimpleObjectProperty<OpenPair<Boolean, Comics>> reference) {

editingObjectReference.set(reference);

currentComicsImagePaths.clear();

setComicsDetails();

}

private void setComicsDetails() {

Comics currentComics = editingObjectReference.get().get().getValue();

if (currentComics == null) {

isNew = true;

ivFirstImage.setImage(Repository.getNoImageImage());

return;

}

ArrayList<Pair<Integer, String>> comicsImagePaths = Repository.instance.getComicsImagePaths(true);

ArrayList<String> currentComicsImagePaths = comicsImagePaths

.stream()

.filter(comicsImage -> comicsImage.getKey() == currentComics.getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

this.currentComicsImagePaths.addAll(currentComicsImagePaths);

if (currentComicsImagePaths.isEmpty()) {

ivFirstImage.setImage(Repository.getNoImageImage());

} else {

String firstImageName = currentComicsImagePaths.get(0);

Image image = loadImage(firstImageName);

if (image == null) {

image = Repository.getNoImageImage();

}

ivFirstImage.setImage(image);

}

centerImage(ivFirstImage);

taName.setText(currentComics.getName());

taDescription.setText(currentComics.getDescription());

String concatenatedString = currentComics.getAltNames()

.stream()

.reduce("", (s, s2) -> s.concat(";").concat(s2));

String altNamesString = (concatenatedString.length() > 0) ? concatenatedString.substring(1) : "";

taAltNames.setText(altNamesString); // remove 1 symbol because it will be ";"

tbChapterCount.setText(String.valueOf(currentComics.getChapterCount()));

tbPremiereYear.setText(String.valueOf(currentComics.getPremiereYear()));

lvSources.getSelectionModel().select(Repository.instance.getNamesBundleValue(currentComics.getSource().name));

lvTypes.getSelectionModel().select(Repository.instance.getNamesBundleValue(currentComics.getType().name));

ArrayList<Genre> currentComicsGenres = Repository.instance.getComicsGenreMap(true).stream()

.filter(comicsGenrePair -> comicsGenrePair.getKey().getId() == currentComics.getId())

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

ArrayList<Integer> genresIdsList = currentComicsGenres

.stream()

.map(genre -> genre.id - 1)

.collect(Collectors.toCollection(ArrayList::new));

int[] genresIds = new int[genresIdsList.size()];

for (int i = 0; i < genresIdsList.size(); i++) {

genresIds[i] = genresIdsList.get(i);

}

lvGenres.getSelectionModel().clearSelection();

lvGenres.getSelectionModel().selectIndices(-1, genresIds);

lvStatus.getSelectionModel().select(Repository.instance.getNamesBundleValue(currentComics.getStatus().name));

lvAuthors.getSelectionModel().select(currentComics.getAuthor().getName());

bNextImage.setDisable(false);

bPreviousImage.setDisable(false);

bDeleteImage.setDisable(false);

if (currentComicsImagePaths.size() <= 1) {

bNextImage.setDisable(true);

bPreviousImage.setDisable(true);

}

if (currentComicsImagePaths.size() == 0) {

bDeleteImage.setDisable(true);

ivCurrentGalleryImage.setImage(Repository.getNoImageImage());

return;

}

String firstImageName = currentComicsImagePaths.get(0);

Image galleryImage = Repository.loadImage(firstImageName);

if (galleryImage != null) {

ivCurrentGalleryImage.setImage(galleryImage);

centerImage(ivCurrentGalleryImage);

} else {

ivCurrentGalleryImage.setImage(Repository.getNoImageImage());

}

}

@FXML

private void onBExitEditWindowClick() {

removeRedundantImages();

ObjectProperty<OpenPair<Boolean, Comics>> reference = editingObjectReference.get();

reference.set(new OpenPair<>(false, null));

}

@SuppressWarnings("ConstantConditions")

private void removeRedundantImages() {

ArrayList<String> dbComicsImagePaths = Repository.instance.getComicsImagePaths(true)

.stream()

.map(Pair::getValue)

.collect(Collectors.toCollection(ArrayList::new));

Repository.instance.handleDirectoryExistence();

String[] list = Paths.get(DB\_IMAGES\_FOLDER).toFile().list();

ArrayList<String> folderComicsImagePaths = new ArrayList<>(Arrays.asList(list));

folderComicsImagePaths.forEach(folderComicsImagePath -> {

if (!dbComicsImagePaths.contains(folderComicsImagePath)) {

try {

Files.delete(Paths.get(DB\_IMAGES\_FOLDER, folderComicsImagePath));

} catch (IOException e) {

logger.error("Error while deleting file " + folderComicsImagePath + ": ", e);

}

}

});

}

@SuppressWarnings("OptionalGetWithoutIsPresent")

@FXML

private void onBSaveClick() {

if (!fieldsAreValidated()) return;

String newName = taName.getText();

String newDescription = taDescription.getText();

ArrayList<String> newAltNames = Arrays.stream(taAltNames.getText().split(";"))

.filter(s -> !s.matches("[ ]\*")).collect(Collectors.toCollection(ArrayList::new));

int newChapterCount = Integer.parseInt(tbChapterCount.getText());

int newPremiereYear = Integer.parseInt(tbPremiereYear.getText());

String newSourceString = lvSources.getSelectionModel().getSelectedItem();

Optional<Comics.Source> optionalSource = Arrays.stream(Comics.Source.values())

.filter(source -> Repository.instance.getNamesBundleValue(source.name).equalsIgnoreCase(newSourceString))

.findFirst();

Comics.Source newSource = optionalSource.orElse(Comics.Source.OTHER);

String newTypeString = lvTypes.getSelectionModel().getSelectedItem();

Optional<Comics.Type> optionalType = Arrays.stream(Comics.Type.values())

.filter(type -> Repository.instance.getNamesBundleValue(type.name).equalsIgnoreCase(newTypeString))

.findFirst();

Comics.Type newType = optionalType.orElse(Comics.Type.MANGA);

String newAuthorString = lvAuthors.getSelectionModel().getSelectedItem();

Optional<Author> optionalAuthor = Repository.instance.getAuthors(false).stream()

.filter(author -> author.getName().equalsIgnoreCase(newAuthorString))

.findFirst();

Author newAuthor;

try {

newAuthor = optionalAuthor.get();

} catch (Exception e) {

logger.fatal("Error while finding author from repository: ", e);

Main.errorAlert.setContentText(Repository.instance.getNamesBundleValue("fatalError"));

Main.errorAlert.showAndWait();

System.exit(-1);

return;

}

ArrayList<String> newGenresStrings = new ArrayList<>(lvGenres.getSelectionModel().getSelectedItems());

ArrayList<String> newGenresStringsLeftToFind = new ArrayList<>(lvGenres.getSelectionModel().getSelectedItems());

ArrayList<Genre> genres = Repository.instance.getGenres(false);

ArrayList<Genre> newGenres = new ArrayList<>();

for (String newGenreString : newGenresStrings) {

Optional<Genre> optionalGenre = genres.stream()

.filter(genre -> {

Optional<String> anyGenreWithTheSameName = newGenresStringsLeftToFind.stream()

.filter(localNewGenreStringLeft -> localNewGenreStringLeft.equalsIgnoreCase(genre.name))

.findFirst();

return anyGenreWithTheSameName.isPresent();

})

.findFirst();

newGenres.add(optionalGenre.get());

newGenresStringsLeftToFind.remove(newGenreString);

}

String newStatusString = lvStatus.getSelectionModel().getSelectedItem();

Optional<Status> optionalStatus = Arrays.stream(Status.values())

.filter(status -> Repository.instance.getNamesBundleValue(status.name).equalsIgnoreCase(newStatusString))

.findFirst();

Status newStatus = optionalStatus.orElse(Status.ONGOING);

if (isNew) {

Optional<Comics> comicsWithMaxId = Repository.instance.getComics(true)

.stream()

.max(Comparator.comparingInt(Comics::getId));

int nextId = comicsWithMaxId.get().getId() + 1;

logger.trace(nextId);

Comics newComics = Comics.Builder.newBuilder()

.setId(nextId)

.setAuthor(newAuthor)

.setName(newName)

.setDescription(newDescription)

.setChapterCount(newChapterCount)

.setPremiereYear(newPremiereYear)

.setSource(newSource)

.setType(newType)

.setAltNames(newAltNames)

.setStatus(newStatus)

.build();

Repository.instance.insertNewComics(newComics);

editingObjectReference.get().get().setValue(newComics);

} else {

Comics currentComics = editingObjectReference.get().get().getValue();

currentComics.setName(newName);

currentComics.setAuthor(newAuthor);

currentComics.setDescription(newDescription);

currentComics.setAltNames(newAltNames);

currentComics.setChapterCount(newChapterCount);

currentComics.setPremiereYear(newPremiereYear);

currentComics.setSource(newSource);

currentComics.setType(newType);

currentComics.setStatus(newStatus);

Repository.instance.updateComics(currentComics);

currentComics.invalidate();

}

int currentComicsId = editingObjectReference.get().get().getValue().getId();

Pair<Comics, ArrayList<Genre>> comicsGenres = new Pair<>(

editingObjectReference.get().get().getValue(), newGenres);

Pair<Comics, ArrayList<String>> comicsImages = new Pair<>(

editingObjectReference.get().get().getValue(), currentComicsImagePaths);

Repository.instance.updateComicsImages(comicsImages);

Repository.instance.updateComicsGenres(comicsGenres);

Repository.instance.updateComicsCache();

logger.info(String.format("Comics with id %d was altered", currentComicsId));

}

private boolean fieldsAreValidated() {

String newName = taName.getText();

int newPremiereYear = Integer.parseInt(tbPremiereYear.getText());

String selectedSource = lvSources.getSelectionModel().getSelectedItem();

String selectedType = lvTypes.getSelectionModel().getSelectedItem();

String selectedStatus = lvStatus.getSelectionModel().getSelectedItem();

String selectedAuthor = lvAuthors.getSelectionModel().getSelectedItem();

boolean nameIsNotEmpty = !newName.matches("[ ]\*");

boolean correctPremiereYear = newPremiereYear >= Repository.FIRST\_ANIME\_PREMIERE\_YEAR;

boolean correctSourceSelected = (selectedSource != null);

boolean correctTypeSelected = (selectedType != null);

boolean correctStatusSelected = (selectedStatus != null);

boolean correctAuthorSelected = (selectedAuthor != null);

StringBuilder warningBuilder = new StringBuilder();

if (!nameIsNotEmpty) warningBuilder.append(Repository.instance.getNamesBundleValue("nameIsEmptyError"));

if (!correctPremiereYear)

warningBuilder.append(Repository.instance.getNamesBundleValue("animePremiereYearIsWrongError"));

if (!correctSourceSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("sourceIsNotSelectedError"));

if (!correctTypeSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("typeIsNotSelectedError"));

if (!correctStatusSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("statusIsNotSelectedError"));

if (!correctAuthorSelected)

warningBuilder.append(Repository.instance.getNamesBundleValue("authorIsNotSelectedError"));

warningBuilder.trimToSize();

if (warningBuilder.length() > 0) {

Main.warningAlert.setContentText(warningBuilder.toString());

Main.warningAlert.show();

}

return nameIsNotEmpty && correctPremiereYear &&

correctSourceSelected && correctTypeSelected &&

correctStatusSelected && correctAuthorSelected;

}

private void centerImage(ImageView imageView) {

Image img = imageView.getImage();

if (img != null) {

double ratioX = imageView.getFitWidth() / img.getWidth();

double ratioY = imageView.getFitHeight() / img.getHeight();

double reducCoeff = Math.min(ratioX, ratioY);

double width = img.getWidth() \* reducCoeff;

double height = img.getHeight() \* reducCoeff;

imageView.setX((imageView.getFitWidth() - width) / 2);

imageView.setY((imageView.getFitHeight() - height) / 2);

}

}

@FXML

private void onBChooseFirstImageClick() {

FileChooser fileChooser = new FileChooser();

File file = fileChooser.showOpenDialog(Main.stage);

if (file == null) return;

Path destinationImagePath = Repository.instance.imageWasCopiedToDbImgFolder(file);

if (destinationImagePath == null) return;

InputStream fileStream = Main.getFileStream(destinationImagePath.toFile());

Image newImage = new Image(fileStream);

ivFirstImage.setImage(newImage);

centerImage(ivFirstImage);

currentComicsImagePaths.add(destinationImagePath.getFileName().toString());

if (currentComicsImagePaths.size() == 1) {

ivCurrentGalleryImage.setImage(newImage);

centerImage(ivCurrentGalleryImage);

bDeleteImage.setDisable(false);

}

try {

fileStream.close();

} catch (IOException e) {

logger.error("Error while closing the file stream: ", e);

}

}

private void setGalleryImage() {

String imageName = currentComicsImagePaths.get(currentImageIndex);

Image image;

if (cachedGalleryImages.containsKey(imageName)) {

image = cachedGalleryImages.get(imageName);

} else {

image = Repository.loadImage(imageName);

cachedGalleryImages.put(imageName, image);

}

if (image == null) {

image = Repository.getNoImageImage();

}

ivCurrentGalleryImage.setImage(image);

centerImage(ivCurrentGalleryImage);

}

@FXML

private void onBPreviousImageClick() {

if (currentImageIndex == 0) {

currentImageIndex = currentComicsImagePaths.size() - 1;

} else {

currentImageIndex--;

}

setGalleryImage();

}

@FXML

private void onBNextImageClick() {

if (currentImageIndex == currentComicsImagePaths.size() - 1) {

currentImageIndex = 0;

} else {

currentImageIndex++;

}

setGalleryImage();

}

@FXML

private void onBAddImageClick() {

FileChooser fileChooser = new FileChooser();

File newImage = fileChooser.showOpenDialog(Main.stage);

if (newImage == null) return;

Path destinationImagePath = Repository.instance.imageWasCopiedToDbImgFolder(newImage);

if (destinationImagePath == null) return;

String newImageFileName = destinationImagePath.getFileName().toString();

currentComicsImagePaths.add(newImageFileName);

currentImageIndex = currentComicsImagePaths.indexOf(newImageFileName);

setGalleryImage();

if (currentComicsImagePaths.size() == 1) {

bDeleteImage.setDisable(false);

} else if (currentComicsImagePaths.size() == 2) {

bPreviousImage.setDisable(false);

bNextImage.setDisable(false);

}

logger.info("Image '" + newImageFileName + "' was added");

}

@FXML

private void onBDeleteImageClick() {

String imageNameToDelete = currentComicsImagePaths.get(currentImageIndex);

currentComicsImagePaths.remove(imageNameToDelete);

if (currentComicsImagePaths.size() == 0) {

bDeleteImage.setDisable(true);

bNextImage.setDisable(true);

bPreviousImage.setDisable(true);

ivCurrentGalleryImage.setImage(Repository.getNoImageImage());

ivFirstImage.setImage(Repository.getNoImageImage());

centerImage(ivCurrentGalleryImage);

return;

} else if (currentImageIndex == currentComicsImagePaths.size()) {

currentImageIndex--;

}

setGalleryImage();

logger.info("Image '" + imageNameToDelete + "' was deleted");

}

@Override

public void initialize(URL location, ResourceBundle resources) {

setUpTextBoxes();

setUpListViews();

}

private void setUpListViews() {

lvSources.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvSources.getItems().clear();

lvSources.getItems().addAll(Arrays.stream(Comics.Source.values())

.map(source -> Repository.instance.getNamesBundleValue(source.name))

.collect(Collectors.toCollection(ArrayList::new)));

lvTypes.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvTypes.getItems().clear();

lvTypes.getItems().addAll(Arrays.stream(Comics.Type.values())

.map(type -> Repository.instance.getNamesBundleValue(type.name))

.collect(Collectors.toCollection(ArrayList::new)));

lvGenres.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

lvGenres.getItems().clear();

lvGenres.getItems().addAll(Repository.instance.getGenres(false).stream()

.map(genre -> genre.name).collect(Collectors.toCollection(ArrayList::new)));

lvStatus.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvStatus.getItems().clear();

lvStatus.getItems().addAll(Arrays.stream(Status.values())

.map(status -> Repository.instance.getNamesBundleValue(status.name))

.collect(Collectors.toCollection(ArrayList::new)));

lvAuthors.getSelectionModel().setSelectionMode(SelectionMode.SINGLE);

lvAuthors.getItems().clear();

lvAuthors.getItems().addAll(Repository.instance.getAuthors(false).stream()

.map(Author::getName)

.collect(Collectors.toCollection(ArrayList::new)));

}

private void setUpTextBoxes() {

ChangeListener<String> tbChangeListener = (observable, oldValue, newValue) -> {

if (newValue.isEmpty()) {

((StringProperty) observable).setValue(String.valueOf(0));

} else if (!newValue.matches("[0-9]{0,4}")) {

((StringProperty) observable).setValue(oldValue);

}

};

tbChapterCount.textProperty().addListener(tbChangeListener);

tbPremiereYear.textProperty().addListener(tbChangeListener);

}

}

## LoginController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.control.Alert;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import java.net.URL;

import java.util.ResourceBundle;

/\*\*

\* Контроллер підменю авторизації. Відповідає лише за саму авторизацію

\*/

public class LoginController implements Initializable {

@FXML

private PasswordField tbPassword;

@FXML

private TextField tbLogin;

@FXML

private void onBLogInClick() {

String login = tbLogin.getText();

String password = tbPassword.getText();

boolean maybeTwoSentences = false;

StringBuilder textBuilder = new StringBuilder();

if (!Repository.instance.loginIsCorrect(login)) {

textBuilder.append(Repository.instance.getNamesBundleValue("loginIncorrect"));

maybeTwoSentences = true;

}

if (!Repository.instance.passwordIsCorrect(password)) {

if (maybeTwoSentences) textBuilder.append("\n");

textBuilder.append(Repository.instance.getNamesBundleValue("passwordIncorrect"));

}

if (textBuilder.length() > 0) {

Main.warningAlert.setContentText(textBuilder.toString());

Main.warningAlert.show();

return;

}

Main.confirmationAlert.setContentText(Repository.instance.getNamesBundleValue("credentialsAreCorrect"));

Main.confirmationAlert.show();

tbLogin.setText("");

tbPassword.setText("");

Repository.adminIsAuthorizedProperty.set(true);

}

@Override

public void initialize(URL location, ResourceBundle resources) {

}

}

## MenuController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import javafx.fxml.FXML;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.control.Tab;

import javafx.scene.control.TabPane;

import javafx.util.Pair;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import java.net.URL;

import java.util.ResourceBundle;

/\*\*

\* Контроллер меню, яке містить всі інші підменю. Відповідає за переключення з

\* одного підменю до іншого тип, що надає доступ до вкладок відповідних підменю,

\* а також за переключення підменю авторизації та підменю налаштувань

\*/

public class MenuController implements Initializable {

private static final Logger logger = LogManager.getLogger(MenuController.class);

@FXML

private TabPane tabPane;

@FXML

private Tab tabAnime;

@FXML

private AnimeController animeController;

@FXML

private Tab tabComics;

@FXML

private ComicsController comicsController;

@FXML

private Tab tabLogin;

@FXML

private LoginController loginController;

@FXML

private void adminHasAuthorized() {

logger.info("Admin has logged in!");

Parent settingsPanel = Main.getForm("SettingsForm");

tabLogin.setContent(settingsPanel);

}

@FXML

private void adminHasUnauthorized() {

logger.info("Admin has logged out!");

Parent loginPanel = Main.getForm("LoginForm");

tabLogin.setContent(loginPanel);

}

@Override

public void initialize(URL location, ResourceBundle resources) {

Repository.adminIsAuthorizedProperty.addListener(observable -> {

if (Repository.adminIsAuthorizedProperty.get()) {

adminHasAuthorized();

} else {

adminHasUnauthorized();

}

});

Parent loginForm = (Parent)(tabLogin.getContent());

Main.formMap.put("LoginForm", new Pair<>(loginForm, loginController));

Parent animeContentForm = (Parent)(tabAnime.getContent());

Main.formMap.put("AnimeForm", new Pair<>(animeContentForm, animeController));

Parent comicsContentForm = (Parent)(tabComics.getContent());

Main.formMap.put("ComicsForm", new Pair<>(comicsContentForm, comicsController));

animeController.setTabAnime(tabAnime);

comicsController.setTabComics(tabComics);

tabPane.getSelectionModel().select(1);

}

}

## SettingsController

package com.folva.moderneastculture.controller;

import com.folva.moderneastculture.Main;

import com.folva.moderneastculture.model.Repository;

import javafx.fxml.FXML;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.scene.layout.Region;

/\*\*

\* Контроллер меню налаштувань. Відповідає лише за зміну облікових даних адміністратора

\*/

public class SettingsController {

@FXML

private TextField tbNewLogin;

@FXML

private PasswordField tbNewPassword;

@FXML

private void onBExitClick() {

Repository.adminIsAuthorizedProperty.set(false);

tbNewLogin.clear();

tbNewPassword.clear();

}

@FXML

private void onBConfirmChanges() {

String newLogin = tbNewLogin.getText();

String newPassword = tbNewPassword.getText();

if (fieldsAreValidated(newLogin, newPassword)) {

Repository.instance.updateAdminLogin(newLogin);

Repository.instance.updateAdminPassword(newPassword);

Main.confirmationAlert.setContentText(

Repository.instance.getNamesBundleValue("adminCredentialsChangesSuccessfully"));

Main.confirmationAlert.show();

}

}

private boolean fieldsAreValidated(String login, String password) {

boolean maybeTwoSentences = false;

StringBuilder textBuilder = new StringBuilder();

if (!Repository.instance.loginIsValid(login)) {

textBuilder.append(Repository.instance.getNamesBundleValue("loginFormatIncorrect"));

maybeTwoSentences = true;

}

if (!Repository.instance.passwordIsValid(password)) {

if (maybeTwoSentences) textBuilder.append("\n");

textBuilder.append(Repository.instance.getNamesBundleValue("passwordFormatIncorrect"));

}

if (textBuilder.length() > 0) {

Main.warningAlert.setContentText(textBuilder.toString());

Main.warningAlert.show();

return false;

}

return true;

}

}