package ru.rvdsystems.liferay.api.annotation;

public interface ConfigurationValueAccessor<T> {

T getValue();

}

package ru.rvdsystems.liferay.api.annotation;

import com.liferay.portal.search.sort.SortOrder;

public @interface DefaultSortOrder {

String by() default "";

SortOrder order() default SortOrder.ASC;

}

package ru.rvdsystems.liferay.api.annotation;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.FIELD)

public @interface LiferayField {

String reference() default "";

String title() default "";

Class<? extends LiferayDocument>[] linkedTypes() default {};

Class<? extends ConfigurationValueAccessor<Long>>[] linkedTypeFolders() default {};

boolean required() default false;

String uploadTitle() default "";

boolean allowUpload() default false;

}

package ru.rvdsystems.liferay.api.annotation;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

/\*\*

\* Annotation is used to mark field as an indexed by elasticsearch.

\*/

@Target({ ElementType.FIELD, ElementType.METHOD })

@Retention(RetentionPolicy.RUNTIME)

public @interface LiferayIndexed {

/\*\*

\* @return Name of a field on ElasticeSearch side. If not specified, the reference of a field is used (@see {@link LiferayField#reference()})

\*/

String name() default "";

/\*\*

\* Flag to mark a field as sortable (allows sorting results on ElasticSearch side). Does not work for link fields (DocumentLibraryLink, ImageLink etc.)

\* Makes sense only for text fields (String or String[]) marked as keywords (non-keyword text fields are sortable anyway), or for Double, Integer, Date, Option fields

\* @return Flag to mark a field as sortable

\*/

boolean sortable() default false;

/\*\*

\* Mark field as a keyword on ElasticSearch side. (Keyword fields allows only exact matching.)

\* This parameter makes sense only for text (String or List<String>) fields.

\* Fields of type DAte, Integer, Double are searched by their own ways, all other fields are keywords by default.

\* @return Mark field as a keyword on ElasticSearch side.

\*/

boolean keyword() default false;

}

package ru.rvdsystems.liferay.api.annotation;

import com.liferay.portal.kernel.search.Field;

import com.liferay.portal.search.sort.SortOrder;

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)

@Target(ElementType.TYPE)

public @interface LiferayType {

String reference() default "";

//Use title only for dev or descriptive purposes

String title() default "";

//Default sorting for searching for lists of instances for this class

//By default, use common "title" property

DefaultSortOrder[] defaultSortBy() default { @DefaultSortOrder(by = Field.TITLE, order = SortOrder.ASC) };

}

package ru.rvdsystems.liferay.api.fieldentity;

import com.liferay.document.library.kernel.model.DLFileEntry;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

import ru.rvdsystems.liferay.api.Util;

import java.net.URLEncoder;

@Getter

@Setter

@ToString

@EqualsAndHashCode(onlyExplicitlyIncluded = true)

public class DocumentLibraryLink {

private String extension;

@EqualsAndHashCode.Include

private String fileEntryId;

private String groupId;

private String title;

private String type;

private String url;

private String uuid;

public static DocumentLibraryLink from(DLFileEntry entry){

DocumentLibraryLink result = new DocumentLibraryLink();

result.setExtension(entry.getExtension());

result.setFileEntryId(Long.toString(entry.getFileEntryId()));

result.setGroupId(Long.toString(entry.getGroupId()));

result.setTitle(entry.getTitle());

result.setType("document");

result.setUrl(Util.getDownloadUrl(entry));

result.setUuid(entry.getUuid());

return result;

}

}

package ru.rvdsystems.liferay.api.fieldentity;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

@Getter

@Setter

@ToString

public class Geolocation {

private Float lat;

private Float lng;

}

package ru.rvdsystems.liferay.api.fieldentity;

import com.liferay.document.library.kernel.model.DLFileEntry;

import com.liferay.document.library.kernel.service.DLFileEntryLocalServiceUtil;

import com.liferay.portal.kernel.exception.PortalException;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

import ru.rvdsystems.liferay.api.Util;

import javax.imageio.ImageIO;

import javax.imageio.ImageReader;

import javax.imageio.stream.ImageInputStream;

import java.awt.\*;

import java.awt.image.BufferedImage;

import java.io.IOException;

import java.io.InputStream;

import java.util.Iterator;

@Getter

@Setter

@ToString

public class ImageLink {

private String alt;

private Long classNameId;

private String description;

private String fileEntryId;

private String groupId;

private Integer height;

private String title;

private String type;

private String url;

private String uuid;

private Integer width;

public static ImageLink from(DLFileEntry entry) throws PortalException, IOException {

if(entry.getMimeType()==null || ! entry.getMimeType().startsWith("image")){

throw new IllegalStateException(String.format("Cannot produce image link from non-image entry %s (id: %s, mime type: %s)", entry.getTitle(), entry.getFileEntryId(), entry.getMimeType()));

}

Dimension imgDim = getImageDimensions(entry);

if(imgDim==null){

throw new IllegalStateException(String.format("Cannot produce image link from image entry %s (id: %s, mime type: %s): cannot determine image dimensions", entry.getTitle(), entry.getFileEntryId(), entry.getMimeType()));

}

ImageLink result = new ImageLink();

result.setAlt(entry.getDescription());

result.setClassNameId(entry.getClassNameId());

result.setDescription(entry.getDescription());

result.setGroupId(Long.toString(entry.getGroupId()));

result.setFileEntryId(Long.toString(entry.getFileEntryId()));

result.setTitle(entry.getTitle());

result.setType("document");

result.setUrl(Util.getDownloadUrl(entry));

result.setUuid(entry.getUuid());

result.setHeight(Double.valueOf(imgDim.getHeight()).intValue());

result.setWidth(Double.valueOf(imgDim.getWidth()).intValue());

return result;

}

private static Dimension getImageDimensions(DLFileEntry entry) throws PortalException, IOException {

try(InputStream entryIs = DLFileEntryLocalServiceUtil.getFileAsStream(entry.getFileEntryId(), entry.getVersion())){

try(ImageInputStream imageIs = ImageIO.createImageInputStream(entryIs)){

final Iterator<ImageReader> imageReaders = ImageIO.getImageReaders(imageIs);

if(imageReaders.hasNext()){

ImageReader reader = imageReaders.next();

try{

reader.setInput(imageIs);

return new Dimension(reader.getWidth(0), reader.getHeight(0));

} catch (IOException e){

//BYPASS, TRY NEXT READER

}finally {

reader.dispose();

}

}

}

}

return null;

}

}

package ru.rvdsystems.liferay.api.fieldentity;

import java.lang.reflect.InvocationTargetException;

import java.lang.reflect.Method;

public interface Option {

//String getReference();

String getTitle();

static <T extends Option> T fromLabel(String value, Class<T> optionClass) {

if(value==null){

return null;

}

try {

Method valuesMethod = optionClass.getMethod("values");

T[] values = (T[]) valuesMethod.invoke(null, null);

for (T option : values) {

if (value.equals(option.getTitle())) {

return option;

}

}

}catch (NoSuchMethodException | IllegalAccessException | InvocationTargetException e){

throw new IllegalStateException(String.format("Unexpectedly cannot invoke static 'values' method of class %s", optionClass),e);

}

return null;

}

}

package ru.rvdsystems.liferay.api.fieldentity;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

@Getter

@Setter

@ToString

public class PageLink {

private Long groupId;

private String id;

private String layoutId;

private String name;

private Boolean privateLayout;

private String title;

private String value;

}

package ru.rvdsystems.liferay.api.repository.exception.validation;

public class LiferayEmptyRequiredFieldException extends LiferayFieldValidationException {

public LiferayEmptyRequiredFieldException(String msg) {

super(msg);

}

}

package ru.rvdsystems.liferay.api.repository.exception.validation;

import com.liferay.portal.kernel.exception.PortalException;

public class LiferayFieldValidationException extends PortalException {

public LiferayFieldValidationException(String msg) {

super(msg);

}

}

package ru.rvdsystems.liferay.api.repository.exception.validation;

public class LiferayIncorrectLinkedTypeException extends LiferayFieldValidationException{

public LiferayIncorrectLinkedTypeException(String msg) {

super(msg);

}

}

package ru.rvdsystems.liferay.api.repository.exception;

public class LiferayFieldMappingException extends LiferayMappingException{

public LiferayFieldMappingException(String message) {

super(message);

}

public LiferayFieldMappingException(String message, Throwable cause) {

super(message, cause);

}

}

package ru.rvdsystems.liferay.api.repository.exception;

import com.liferay.portal.kernel.exception.PortalException;

public class LiferayIncompatibleMediatorException extends PortalException {

public LiferayIncompatibleMediatorException(String msg) {

super(msg);

}

}

package ru.rvdsystems.liferay.api.repository.exception;

import com.liferay.portal.kernel.exception.PortalException;

public class LiferayIndexConfigurationException extends PortalException {

public LiferayIndexConfigurationException(String msg) {

super(msg);

}

}

package ru.rvdsystems.liferay.api.repository.exception;

import com.liferay.portal.kernel.exception.PortalException;

/\*\*

\* Exception that describes problem while mapping between Liferay entity and Java object

\*/

public class LiferayMappingException extends PortalException {

public LiferayMappingException(String message) {

super(message);

}

public LiferayMappingException(String message, Throwable cause) {

super(message, cause);

}

}

package ru.rvdsystems.liferay.api.repository.exception;

import com.liferay.portal.kernel.exception.PortalException;

public class LiferaySecurityException extends PortalException {

public LiferaySecurityException(String msg) {

super(msg);

}

}

package ru.rvdsystems.liferay.api.repository.exception;

import com.liferay.portal.kernel.exception.PortalException;

public class LiferayTransactionInvocationException extends PortalException {

public LiferayTransactionInvocationException(Throwable throwable) {

super("Error while invoking transaction", throwable);

}

}

package ru.rvdsystems.liferay.api.repository.exception;

public class LiferayTypeMappingException extends LiferayMappingException{

public LiferayTypeMappingException(String message) {

super(message);

}

public LiferayTypeMappingException(String message, Throwable cause) {

super(message, cause);

}

}

package ru.rvdsystems.liferay.api.repository.exception;

public class UnsupportedException extends RuntimeException{

public UnsupportedException(String message) {

super(message);

}

}

package ru.rvdsystems.liferay.api.repository;

import com.liferay.document.library.kernel.model.DLFolder;

import com.liferay.document.library.kernel.service.DLFileEntryLocalServiceUtil;

import com.liferay.portal.kernel.exception.PortalException;

/\*\*

\* Liferay document that automatically placed into additional subfolder during saving.

\*/

public interface AutoFolded {

/\*\*

\* Te main methos of interface.

\* @return The name of a subfolder that must be created inside parent folder.

\* The object will be created inside this subfolder

\*/

String getAutoFolderName();

}

package ru.rvdsystems.liferay.api.repository;

/\*\*

\* An implementation for basic liferay document (that does not have DDM structures)

\*/

public class BasicLiferayFileDocument extends LiferayDocument{

}

package ru.rvdsystems.liferay.api.repository;

/\*\*

\* Common marker for class that contains fields. Must be used for both file entry and Field set

\*/

public interface DDMFieldContainer {

}

package ru.rvdsystems.liferay.api.repository;

import com.liferay.document.library.kernel.model.DLFileEntry;

import com.liferay.document.library.kernel.service.DLFileEntryLocalServiceUtil;

import com.liferay.portal.kernel.exception.PortalException;

import com.liferay.portal.kernel.model.User;

import lombok.Getter;

import lombok.Setter;

import org.apache.commons.io.IOUtils;

import ru.rvdsystems.liferay.api.fieldentity.DocumentLibraryLink;

import ru.rvdsystems.liferay.api.fieldentity.ImageLink;

import java.io.IOException;

import java.io.InputStream;

@Getter

public abstract class LiferayDocument implements DDMFieldContainer {

@Setter

private Long dlFileEntryId;

@Setter

private Long typeId;

@Setter

private String title;

@Setter

private String description;

@Setter

private String fileName;

private byte[] contentToSave;

private String mimeTypeToSave;

/\*\*

\* To be overriden by descendant classes.

\* Method is invoked before save a new object and checks if current user has access to create it.

\* @param user

\* @return

\*/

public boolean hasCreateAccess(User user) throws PortalException {

return true;

}

/\*\*

\* To be overriden by descendant classes.

\* Method is invoked before save existing object and checks if current user has access to update the object.

\* @param user

\* @return

\*/

public boolean hasUpdateAccess(User user) throws PortalException {

return true;

}

public void setContentToSave(InputStream fileContent, String mimeType, String fileName) throws IOException {

contentToSave = IOUtils.toByteArray(fileContent);

mimeTypeToSave = mimeType;

this.fileName = fileName;

}

public DocumentLibraryLink toDocumentLibraryLink() throws PortalException {

if(dlFileEntryId==null){

throw new IllegalStateException(String.format("Cannot create %s instance for non-saved entity", DocumentLibraryLink.class.getSimpleName()));

}

DLFileEntry dlEntry = DLFileEntryLocalServiceUtil.getDLFileEntry(dlFileEntryId);

return DocumentLibraryLink.from(dlEntry);

}

public ImageLink toImageLink() throws PortalException, IOException {

if(dlFileEntryId==null){

throw new IllegalStateException(String.format("Cannot create %s instance for non-saved entity", ImageLink.class.getSimpleName()));

}

DLFileEntry dlEntry = DLFileEntryLocalServiceUtil.getDLFileEntry(dlFileEntryId);

return ImageLink.from(dlEntry);

}

}

package ru.rvdsystems.liferay.api.repository;

import com.liferay.document.library.kernel.model.DLFileEntryType;

import com.liferay.document.library.kernel.model.DLFolder;

import com.liferay.osgi.util.ServiceTrackerFactory;

import com.liferay.portal.kernel.exception.PortalException;

import com.liferay.portal.kernel.model.User;

import com.liferay.portal.kernel.service.ServiceContext;

import com.liferay.portal.kernel.upload.UploadRequest;

import com.liferay.portal.search.query.Queries;

import com.liferay.portal.search.query.Query;

import com.liferay.portal.search.sort.Sort;

import com.liferay.portal.search.sort.SortBuilderFactory;

import lombok.NonNull;

import lombok.extern.slf4j.Slf4j;

import org.osgi.annotation.versioning.ProviderType;

import org.osgi.framework.FrameworkUtil;

import org.osgi.util.tracker.ServiceTracker;

import ru.rvdsystems.liferay.api.search.LiferayDocumentSearchResult;

import ru.rvdsystems.liferay.api.upload.UploadSaveResult;

import java.util.List;

import java.util.concurrent.Callable;

@ProviderType

@Slf4j

public class LiferayDocumentRepository {

private static final long DELAYED\_REG\_PACKAGE\_TIMEOUT = 60000;

private static final long DELAYED\_REG\_PACKAGE\_INTERVAL = 1000;

private static final ServiceTracker<LiferayDocumentRepositoryLocalService, LiferayDocumentRepositoryLocalService>

\_repoService = ServiceTrackerFactory.open(FrameworkUtil.getBundle(ServiceTrackerFactory.class), LiferayDocumentRepositoryLocalService.class);

private static LiferayDocumentRepositoryLocalService.ListSort[] DEFAULT\_SORTS = {LiferayDocumentRepositoryLocalService.ListSort.BY\_TITLE\_ASC};

public static <T extends LiferayDocument> T get(long fileEntryId, Class<T> clazz) throws PortalException {

return \_repoService.getService().get(fileEntryId, clazz);

}

public static <T extends LiferayDocument> T save(ServiceContext serviceContext, User user, T entry) throws PortalException {

return \_repoService.getService().save(serviceContext, user, entry);

}

public static <T extends LiferayDocument> T save(ServiceContext serviceContext, User user, long folderId, T entry) throws PortalException {

return \_repoService.getService().save(serviceContext,user,folderId,entry);

}

@Deprecated

/\*

\* To be removed.

\* Use instead:

\* {@link #searchInFolder(DLFolder, boolean, com.liferay.portal.search.sort.Sort[], Class, String...)}

\*/

public static <T extends LiferayDocument> List<T> list(long groupId, long folderId, Class<T> clazz, LiferayDocumentRepositoryLocalService.ListSort ...sorts) throws PortalException {

return \_repoService.getService().list(groupId, folderId, clazz, sorts!=null && sorts.length > 0 ? sorts : DEFAULT\_SORTS);

}

public static DLFolder checkAutoFolder(final @NonNull AutoFolded autoFolded, final long parentFolderId, final @NonNull User user, final @NonNull ServiceContext serviceContext) throws PortalException {

return \_repoService.getService().checkAutoFolder(autoFolded, parentFolderId,user,serviceContext);

}

public static <T extends LiferayDocument> UploadSaveResult<T> save(UploadRequest uploadRequest, long folderId, Class<T> clazz) throws PortalException {

return \_repoService.getService().save(uploadRequest, folderId, clazz);

}

public static <T extends LiferayDocument> UploadSaveResult<T> save(UploadRequest uploadRequest, T entry) throws PortalException {

return \_repoService.getService().save(uploadRequest, entry);

}

public static <T> T invokeTransaction(Callable<T> callable) throws PortalException {

return \_repoService.getService().invokeTransaction(callable);

}

public static Queries getQueries(){

return \_repoService.getService().getQueries();

}

public static SortBuilderFactory getSortBuilderFactory(){

return \_repoService.getService().getSortBuilderFactory();

}

public static <T extends LiferayDocument> LiferayDocumentSearchResult<T> searchAll(User user, Class<T> clazz, int from, int size, String... returnFields) throws PortalException{

return \_repoService.getService().searchAll(user, clazz, from, size, returnFields);

}

public static <T extends LiferayDocument> LiferayDocumentSearchResult<T> search(User user, Class<T> clazz, Query query, int from, int size, String ...returnfields) throws PortalException {

return \_repoService.getService().search(user, clazz, query, from, size, returnfields);

}

public static <T extends LiferayDocument> LiferayDocumentSearchResult<T> search(User user, Class<T> clazz, Query query, Sort[] sorts, int from, int size, String ...returnfields) throws PortalException {

return \_repoService.getService().search(user, clazz, query, sorts, from, size, returnfields);

}

public static <T extends LiferayDocument> LiferayDocumentSearchResult<T> listFolder(User user, Class<T> clazz, DLFolder folder, boolean includeSubFolders, Sort[] sorts, String... returnFields) throws PortalException {

return \_repoService.getService().listFolder(user, clazz, folder, includeSubFolders, sorts, returnFields);

}

public static <T extends LiferayDocument> LiferayDocumentSearchResult<T> listFolder(User user, Class<T> clazz, DLFolder folder, boolean includeSubFolders, String... returnFields) throws PortalException {

return \_repoService.getService().listFolder(user, clazz, folder, includeSubFolders, returnFields);

}

public static void registerAllLiferayDocumentClassesInPackageAndSubpackages(final Class<?> clazz) throws PortalException {

LiferayDocumentRepositoryLocalService service = \_repoService.getService();

if(service!=null) {

service.registerAllLiferayDocumentClassesInPackageAndSubpackages(clazz);

} else {

//At startup, it is a common situation when dependent bundles starts BEFORE

//In that situation, if this method is called, service is not started yet.

//Start thread that will wait for the service

log.warn("Service has not started yet, delaying registration of class package {}", clazz);

new Thread(() -> {

try {

long millis = 0;

while (millis < DELAYED\_REG\_PACKAGE\_TIMEOUT) {

long m = System.currentTimeMillis();

Thread.sleep(DELAYED\_REG\_PACKAGE\_INTERVAL);

LiferayDocumentRepositoryLocalService srvc = \_repoService.getService();

if(srvc != null){

srvc.registerAllLiferayDocumentClassesInPackageAndSubpackages(clazz);

return;

}

millis+=System.currentTimeMillis() - m;

log.debug("No service available after {} ms, will waiting {} ms...", millis, DELAYED\_REG\_PACKAGE\_TIMEOUT - millis);

}

log.error("Delayed register package error: could not get working service in {} ms", DELAYED\_REG\_PACKAGE\_INTERVAL);

} catch (InterruptedException | PortalException e) {

log.error("Delayed register package error",e);

}

}).start();

}

}

public static Class<? extends LiferayDocument> findRegisteredClass(DLFileEntryType dlType){

return \_repoService.getService().findRegisteredClass(dlType);

}

public static String getTypeReference(DLFileEntryType entryType){

return \_repoService.getService().getTypeReference(entryType);

}

}

package ru.rvdsystems.liferay.api.repository;

import com.liferay.document.library.kernel.model.DLFileEntry;

import com.liferay.document.library.kernel.model.DLFileEntryType;

import com.liferay.document.library.kernel.model.DLFolder;

import com.liferay.portal.kernel.exception.PortalException;

import com.liferay.portal.kernel.model.User;

import com.liferay.portal.kernel.service.BaseLocalService;

import com.liferay.portal.kernel.service.ServiceContext;

import com.liferay.portal.kernel.upload.UploadRequest;

import com.liferay.portal.search.query.Queries;

import com.liferay.portal.search.query.Query;

import com.liferay.portal.search.sort.Sort;

import com.liferay.portal.search.sort.SortBuilderFactory;

import org.osgi.annotation.versioning.ProviderType;

import ru.rvdsystems.liferay.api.annotation.LiferayType;

import ru.rvdsystems.liferay.api.search.LiferayDocumentSearchResult;

import ru.rvdsystems.liferay.api.upload.UploadSaveResult;

import java.util.List;

import java.util.concurrent.Callable;

@ProviderType

public interface LiferayDocumentRepositoryLocalService extends BaseLocalService {

String BASIC\_DOCUMENT\_TYPE\_REFERENCE="@\_BASIC\_DOCUMENT\_@";

String getTypeReference(DLFileEntryType dlType);

enum ListSort{

BY\_CREATED\_ASC, BY\_MODIFIED\_ASC, BY\_TITLE\_ASC, BY\_SIZE\_ASC, BY\_READ\_COUNT\_ASC,

BY\_CREATED\_DESC, BY\_MODIFIED\_DESC, BY\_TITLE\_DESC, BY\_SIZE\_DESC, BY\_READ\_COUNT\_DESC

}

/\*\*

\* Get a file entry in form of necessary class

\* @param fileEntryId

\* @param clazz Perform type resolution based on the class only. (Suitable for annotation-driven approach)

\* @param <T>

\* @return

\* @throws PortalException

\*/

<T extends LiferayDocument> T get(long fileEntryId, Class<T> clazz) throws PortalException;

/\*\*

\* Get file entry bean in form of necessary class

\* @param fileEntryId

\* @param entry Perform type resolution based on the instance (suitable when type cannot be determined at compile time)

\* @param <T>

\* @return

\* @throws PortalException

\*/

<T extends LiferayDocument> T get(long fileEntryId, T entry) throws PortalException;

/\*\*

\* Get file entry bean

\* @param dlFileEntry entry

\* @param clazz entry bean class

\* @param <T>

\* @return

\* @throws PortalException

\*/

<T extends LiferayDocument> T get(DLFileEntry dlFileEntry, Class<T> clazz) throws PortalException;

<T extends LiferayDocument> T get(DLFileEntry dlFileEntry, T entry) throws PortalException;

/\*\*

\* Invoke a liferay transaction

\* @param callable a callable to execute in transaction scope

\* @param <T> return type parameter

\* @return value returned by <code>callable</code>

\* @throws PortalException

\*/

<T> T invokeTransaction(Callable<T> callable) throws PortalException;

/\*\*

\* Save entry to liferay repository folder, without file content

\* @param <T>

\* @param serviceContext

\* @param user

\* @param folderId Folder to place entry. If entry with the same title exists in that folder, that entry will be updated (new version will be created).

\* @param entry

\* @throws PortalException

\* @return

\*/

<T extends LiferayDocument> T save(ServiceContext serviceContext, User user, long folderId, T entry) throws PortalException;

<T extends LiferayDocument> UploadSaveResult<T> save(UploadRequest uploadRequest, long folderId, Class<T> clazz) throws PortalException;

/\*\*

\* Save already existing entry to liferay repository, without file content

\* @param <T>

\* @param serviceContext

\* @param user

\* @param entry

\* @throws PortalException

\* @return

\*/

<T extends LiferayDocument> T save(ServiceContext serviceContext, User user, T entry) throws PortalException;

<T extends LiferayDocument> UploadSaveResult<T> save(UploadRequest uploadRequest, T entry) throws PortalException;

/\*\*

\* Get entries of desired type from the folder.

\* @param groupId

\* @param folderId

\* @param clazz Perform type resolution based on the class only. (Suitable for annotation-driven approach)

\* @param sorts

\* @param <T>

\* @return

\* @throws PortalException

\*/

@Deprecated

/\*

\* To be removed.

\* Use instead:

\* {@link #searchInFolder(DLFolder, boolean, com.liferay.portal.search.sort.Sort[], Class, String...)}

\*/

<T extends LiferayDocument> List<T> list(long groupId, long folderId, Class<T> clazz, ListSort... sorts) throws PortalException;

/\*\*

\* Check if an auto-created folder exists in parent folder, create sub-folder according to AutoFolded parameter

\* @param autoFolded Autofolded object

\* @param parentFolderId parent folder id

\* @param user current user

\* @param serviceContext service context

\* @return Checked or created subfolder

\* @throws PortalException

\*/

DLFolder checkAutoFolder(AutoFolded autoFolded, long parentFolderId, User user, ServiceContext serviceContext) throws PortalException;

/\*\*

\* Scan given class's package and subpackages for @see @link{{@link LiferayDocument}} classes, register them in the API.

\* Registration means that classes are:

\* 1) checked according their mappings (@see {@link ru.rvdsystems.liferay.api.annotation.LiferayType}, {@link ru.rvdsystems.liferay.api.annotation.LiferayField})

\* 2) checked according their Liferay indexing settings,and included in indexing processing

\* (@see {@link ru.rvdsystems.liferay.api.annotation.LiferayIndexed} annotation)

\* @param clazz

\* @throws PortalException

\*/

void registerAllLiferayDocumentClassesInPackageAndSubpackages(Class<?> clazz) throws PortalException;

/\*\*

\* Find class by given liferay type. Class must be registered before (@see {@link #registerAllLiferayDocumentClassesInPackageAndSubpackages(Class)})

\* @param dlType Liferay type to find correponding bean class

\* @return Found class, or null if not found

\*/

Class<? extends LiferayDocument> findRegisteredClass(DLFileEntryType dlType);

/\*\*

\* @return Object to construct liferay search queries

\*/

Queries getQueries();

/\*\*

\* @return Object to construct liferay search Sort objects

\*/

SortBuilderFactory getSortBuilderFactory();

/\*\*

\* List objects for given class in given folder and optionally in subfolders

\* @param clazz Bean class to search and return

\* @param folder A folder to search in

\* @param includeSubFolders return objects in given fodler only (false) or also in it's subfolders hierarchy

\* @param sorts List of sorts of results

\* @param returnFields Additional fields to return from search index. It allows to avoid requests to liferay db to construct beans. Useful for lists.

\* @param <T>

\* @return Search result

\* @throws PortalException

\*/

<T extends LiferayDocument> LiferayDocumentSearchResult<T> listFolder(User user, Class<T> clazz, DLFolder folder, boolean includeSubFolders, Sort[] sorts, String... returnFields) throws PortalException;

/\*\*

\* List objects for given class in given folder and optionally in subfolders, sorted by default sorting of given class (@see {@link LiferayType#defaultSortBy()} )

\* @param clazz Bean class to search and return

\* @param folder A folder to search in

\* @param includeSubFolders return objects in given fodler only (false) or also in it's subfolders hierarchy

\* @param returnFields Additional fields to return from search index. It allows to avoid requests to liferay db to construct beans. Useful for lists.

\* @param <T>

\* @return Search result

\* @throws PortalException

\*/

<T extends LiferayDocument> LiferayDocumentSearchResult<T> listFolder(User user, Class<T> clazz, DLFolder folder, boolean includeSubFolders, String... returnFields) throws PortalException;

/\*\*

\* Search all beans of given class.

\* @param clazz Bean class to search and return

\* @param from Return objects from given number (For pagination purposes)

\* @param size Limit the number of returned objects (For pagination purposes)

\* @param returnFields Additional fields to return from search index. It allows to avoid requests to liferay db to construct beans. Useful for lists.

\* @param <T>

\* @return Search result

\* @throws PortalException

\*/

<T extends LiferayDocument> LiferayDocumentSearchResult<T> searchAll(User user, Class<T> clazz, int from, int size, String... returnFields) throws PortalException;

/\*\*

\* Search for beans of given class by given query, sorted by default sorting of given class (@see {@link LiferayType#defaultSortBy()} )

\* @param clazz Bean class to search and return

\* @param query A query to search. @see {@link #getQueries()}

\* @param from Return objects from given number (For pagination purposes)

\* @param size Limit the number of returned objects (For pagination purposes)

\* @param returnFields Additional fields to return from search index. It allows to avoid requests to liferay db to construct beans. Useful for lists.

\* @param <T>

\* @return Search result

\* @throws PortalException

\*/

<T extends LiferayDocument> LiferayDocumentSearchResult<T> search(User user, Class<T> clazz, Query query, int from, int size, String... returnFields) throws PortalException;

/\*\*

\* Search for beans of given class by given query

\* @param clazz Bean class to search and return

\* @param query A query to search. @see {@link #getQueries()}

\* @param sorts A list of sorts

\* @param from Return objects from given number (For pagination purposes)

\* @param size Limit the number of returned objects (For pagination purposes)

\* @param returnFields Additional fields to return from search index. It allows to avoid requests to liferay db to construct beans. Useful for lists.

\* @param <T>

\* @return Search result

\* @throws PortalException

\*/

<T extends LiferayDocument> LiferayDocumentSearchResult<T> search(User user, Class<T> clazz, Query query, Sort[] sorts, int from, int size, String ...returnFields) throws PortalException;

/\*\*

\* Meve given entry to trash

\* @param user User

\* @param entry An aentry to move to thash

\* @param <T>

\* @throws PortalException

\*/

<T extends LiferayDocument> void moveToTrash(User user, T entry) throws PortalException;

}

package ru.rvdsystems.liferay.api.search;

import lombok.Getter;

import lombok.Setter;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import java.util.Collections;

import java.util.List;

/\*\*

\* Result of searching using lieray indexing API

\*/

@Getter

@Setter

public class LiferayDocumentSearchResult<E extends LiferayDocument> {

/\*\*

\* total results found

\*/

private long total;

/\*\*

\* List of returned objects

\*/

private List<LiferayIndexSearchHit<E>> results;

public static <T extends LiferayDocument> LiferayDocumentSearchResult<T> emptyResult(){

LiferayDocumentSearchResult<T> result = new LiferayDocumentSearchResult<>();

result.setResults(Collections.emptyList());

return result;

}

}

package ru.rvdsystems.liferay.api.search;

import com.liferay.portal.kernel.exception.PortalException;

import com.liferay.portal.kernel.search.Field;

import com.liferay.portal.search.hits.SearchHit;

import lombok.experimental.Delegate;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import ru.rvdsystems.liferay.api.repository.LiferayDocumentRepository;

/\*\*

\* A single result of indexed searching

\* @param <E>

\*/

public class LiferayIndexSearchHit<E extends LiferayDocument> implements SearchHit {

@Delegate

private final SearchHit searchHit;

private final Class<E> clazz;

private E entry;

public LiferayIndexSearchHit(SearchHit searchHit, Class<E> clazz) {

this.searchHit = searchHit;

this.clazz = clazz;

}

/\*\*

\* @return an id of lieferay entry

\*/

public long getDlFileEntryId(){

return getDocument().getLong(Field.ENTRY\_CLASS\_PK);

}

/\*\*

\* @return A bean.

\* @throws PortalException

\*/

public E getEntry(){

if(entry==null) {

try {

entry = LiferayDocumentRepository.get(getDlFileEntryId(), clazz);

} catch (PortalException e) {

throw new RuntimeException(e);

}

}

return entry;

}

}

package ru.rvdsystems.liferay.api.search;

public class SearchConstants {

public static final String FIELD\_FILE\_ENTRY\_TYPE\_ID = "fileEntryTypeId";

public static final String FIELD\_VISIBLE = "visible";

public static final String FIELD\_SIZE = "size";

public static final String FIELD\_VIEW\_COUNT="viewCount";

}

package ru.rvdsystems.liferay.api.upload;

import com.liferay.portal.kernel.upload.UploadRequest;

import lombok.extern.slf4j.Slf4j;

import ru.rvdsystems.liferay.api.Util;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import java.util.TreeMap;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import java.util.stream.Collectors;

@Slf4j

public class RequestParameterUtil {

private static final Pattern NUMBERED\_PARAM = Pattern.compile("^(.\*)\_(\\d+)$");

private static class ListMap extends TreeMap<Integer, Object> { }

public static Map<String, Object> getFormParameterMap(UploadRequest uploadRequest, final boolean includeFiles, final String prefix) {

Map<String, Object> result = new LinkedHashMap<>();

Map<String,List<Object>> params =

uploadRequest.getRegularParameterMap().entrySet().stream().collect(

Collectors.toMap(Map.Entry::getKey, entry-> new ArrayList<>(entry.getValue()))

);

if(includeFiles) {

//Add all file items to the map

//Remove empty file items

//Remove empty lists

params.putAll(

uploadRequest.getMultipartParameterMap().entrySet().stream().collect(

Collectors.toMap(

Map.Entry::getKey,

entry ->

Arrays.stream(entry.getValue()).filter(item -> !Util.isEmpty(item.getFullFileName()))

.collect(Collectors.toList())

)

).entrySet().stream().filter(entry -> entry.getValue() != null && !entry.getValue().isEmpty())

.collect(Collectors.toMap(Map.Entry::getKey, entry -> new ArrayList<>(entry.getValue())))

);

}

params.entrySet().stream().sorted(Map.Entry.comparingByKey()).filter(entry -> prefix == null || entry.getKey().startsWith(prefix)).forEach(entry -> {

log.info("Mapping parameter '{}' with value: {}", entry.getKey(), entry.getValue());

String paramName = entry.getKey();

List<Object> paramValues = entry.getValue();

Object paramValue = paramValues.size()==1 ? paramValues.get(0) : paramValues;

String[] paramPath = paramName.split("\\.");

paramName = paramPath[paramPath.length - 1];

paramPath = Arrays.copyOf(paramPath, paramPath.length - 1);

Map<String, Object> parent = result;

for (String pathElem : paramPath) {

Matcher numberedMatcher = NUMBERED\_PARAM.matcher(pathElem);

if (numberedMatcher.matches()) {

pathElem = numberedMatcher.group(1);

int elemNo = Integer.parseInt(numberedMatcher.group(2));

ListMap listMap = (ListMap) parent.computeIfAbsent(pathElem, v -> new ListMap());

parent = (Map<String, Object>) listMap.computeIfAbsent(elemNo, v -> new LinkedHashMap<>());

} else {

parent = (Map<String, Object>) parent.computeIfAbsent(pathElem, v -> new LinkedHashMap<>());

}

}

Matcher numberedMatcher = NUMBERED\_PARAM.matcher(paramName);

if(numberedMatcher.matches()){

paramName = numberedMatcher.group(1);

int elemNo = Integer.parseInt(numberedMatcher.group(2));

ListMap listMap = (ListMap) parent.computeIfAbsent(paramName, v -> new ListMap());

listMap.put(elemNo, paramValue);

} else {

parent.put(paramName, paramValue);

}

});

//convert all ListMaps to Lists

convertListMaps(result);

return result;

}

private static void convertListMaps(Map<String,Object> paramsMap) {

paramsMap.entrySet().forEach( entry -> {

Object v = entry.getValue();

if(v instanceof Map){

convertListMaps((Map<String, Object>) v);

if(v instanceof ListMap){

entry.setValue( new ArrayList<>(((ListMap) v).values()));

}

}

});

}

}

package ru.rvdsystems.liferay.api.upload;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.Setter;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import java.util.List;

@Getter

@AllArgsConstructor

public class UploadSaveResult<E extends LiferayDocument> {

private E mainDocument;

private List<LiferayDocument> linkedDocuments;

}

package ru.rvdsystems.liferay.api;

import com.liferay.osgi.util.ServiceTrackerFactory;

import com.liferay.portal.search.query.\*;

import com.liferay.portal.search.sort.SortBuilderFactory;

import lombok.experimental.Delegate;

import org.osgi.framework.FrameworkUtil;

import org.osgi.util.tracker.ServiceTracker;

import ru.rvdsystems.liferay.api.QueryLocalService;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import ru.rvdsystems.liferay.api.repository.exception.LiferayMappingException;

import java.sql.Date;

import java.util.Arrays;

import java.util.Objects;

public class LiferayQueries{

private static final ServiceTracker<QueryLocalService, QueryLocalService>

queryServiceTracker = ServiceTrackerFactory.open(FrameworkUtil.getBundle(ServiceTrackerFactory.class), QueryLocalService.class);

public static Queries getQueries(){

return queryServiceTracker.getService().getQueries();

}

public static SortBuilderFactory getSortBuilderFactory(){

return queryServiceTracker.getService().getSortBuilderFactory();

}

public static String getSortableFieldName(Class<? extends LiferayDocument> clazz, String fieldName) throws LiferayMappingException {

return queryServiceTracker.getService().getSortableFieldName(clazz, fieldName);

}

public static TermQuery term(Class<? extends LiferayDocument> clazz, String fieldName, Object value) throws LiferayMappingException {

return getQueries().term(getSortableFieldName(clazz,fieldName),value);

}

public static TermsQuery terms(Class<? extends LiferayDocument> clazz, String fieldName, Object ...values) throws LiferayMappingException {

TermsQuery result = getQueries().terms(getSortableFieldName(clazz,fieldName));

if(values !=null){

result.addValues(values);

}

return result;

}

public static RangeTermQuery rangeTerm(Class<? extends LiferayDocument> clazz, String fieldName, Object value1, Object value2) throws LiferayMappingException {

return getQueries().rangeTerm(getSortableFieldName(clazz,fieldName), true,true, value1, value2);

}

public static BooleanQuery booleanMust(Query ...mustQueries){

BooleanQuery result = getQueries().booleanQuery();

if(mustQueries != null){

mustQueries = Arrays.stream(mustQueries).filter(Objects::nonNull).toArray(Query[]::new);

if(mustQueries.length > 0) {

result.addMustQueryClauses(mustQueries);

}

}

return result;

}

}

package ru.rvdsystems.liferay.api;

import com.liferay.portal.kernel.service.BaseLocalService;

import com.liferay.portal.search.query.Queries;

import com.liferay.portal.search.sort.SortBuilderFactory;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import ru.rvdsystems.liferay.api.repository.exception.LiferayMappingException;

public interface QueryLocalService extends BaseLocalService {

Queries getQueries();

SortBuilderFactory getSortBuilderFactory();

String getSortableFieldName(Class<? extends LiferayDocument> clazz, String indexedField) throws LiferayMappingException;

}

package ru.rvdsystems.liferay.api;

import com.liferay.document.library.kernel.model.DLFileEntry;

import com.liferay.portal.kernel.exception.PortalException;

import com.liferay.portal.kernel.model.User;

import com.liferay.portal.kernel.model.role.RoleConstants;

import com.liferay.portal.kernel.service.RoleLocalServiceUtil;

import java.io.UnsupportedEncodingException;

import java.net.URLEncoder;

import java.time.LocalDate;

import java.time.ZoneId;

import java.time.format.DateTimeFormatter;

import java.util.Date;

import java.util.List;

public class Util {

private static DateTimeFormatter DATE\_FORMATTER=DateTimeFormatter.ofPattern("dd.MM.yyyy");

public static String formatDate(LocalDate date){

return date!=null ? date.format(DATE\_FORMATTER) : "";

}

public static String formatDate(Date date) {

Date dt = date instanceof java.sql.Date ? new Date(date.getTime()) : date;

return date != null ? formatDate(dt.toInstant().atZone(ZoneId.systemDefault()).toLocalDate()) : "";

}

public static Date parseDate(String dateStr){

return Date.from(LocalDate.parse(dateStr, DATE\_FORMATTER).atStartOfDay(ZoneId.systemDefault()).toInstant());

}

public static boolean isEmpty(String value){

return value==null || "".equals(value);

}

public static boolean isEmpty(List<String> values){

return values==null || values.isEmpty() || values.stream().allMatch(Util::isEmpty);

}

public static String prepareWildcard(String wildcard){

return wildcard.indexOf('\*')==-1 && wildcard.indexOf('?')==-1 ? "\*"+wildcard+"\*" : wildcard;

}

public static boolean isAdministrator(User user) throws PortalException {

return user != null && RoleLocalServiceUtil.hasUserRole(user.getUserId(), user.getCompanyId(), RoleConstants.ADMINISTRATOR, true);

}

public static String getDownloadUrl(DLFileEntry entry){

try {

return "/documents/"+entry.getGroupId()+"/"+entry.getFolderId()+"/"+ URLEncoder.encode(entry.getFileName(),"UTF-8")+"/"+entry.getUuid()+"?download=true";

} catch (UnsupportedEncodingException e) {

throw new RuntimeException(e);

}

}

}

package ru.rvdsystems.liferay.impl.mediator.annotated;

import com.liferay.document.library.kernel.model.DLFileEntryType;

import com.liferay.document.library.kernel.model.DLFileEntryTypeConstants;

import com.liferay.document.library.kernel.service.DLFileEntryTypeLocalServiceUtil;

import com.liferay.portal.kernel.exception.PortalException;

import com.liferay.portal.kernel.search.Field;

import com.liferay.portal.search.sort.Sort;

import com.liferay.portal.search.sort.SortBuilderFactory;

import com.liferay.portal.search.sort.SortOrder;

import lombok.EqualsAndHashCode;

import lombok.NonNull;

import lombok.extern.slf4j.Slf4j;

import org.springframework.util.StringUtils;

import ru.rvdsystems.liferay.api.annotation.DefaultSortOrder;

import ru.rvdsystems.liferay.api.annotation.LiferayType;

import ru.rvdsystems.liferay.api.repository.BasicLiferayFileDocument;

import ru.rvdsystems.liferay.api.repository.DDMFieldContainer;

import ru.rvdsystems.liferay.api.repository.LiferayDocumentRepository;

import ru.rvdsystems.liferay.api.repository.exception.LiferayMappingException;

import ru.rvdsystems.liferay.api.repository.exception.LiferayTypeMappingException;

import ru.rvdsystems.liferay.impl.mediator.DDMFieldContainerMediator;

import java.util.Arrays;

import java.util.List;

@Slf4j

@EqualsAndHashCode(callSuper = true)

public class AnnotatedDDMFieldContainerMediator extends DDMFieldContainerMediator {

private String typeReference;

private Sort[] defSort;

public AnnotatedDDMFieldContainerMediator(Class<? extends DDMFieldContainer> clazz) throws LiferayMappingException {

super(AnnotatedDDMFieldMediator.getInstance(clazz));

}

@Override

public DLFileEntryType resolveType() throws PortalException {

return resolveType(getFieldContainerClass());

}

@Override

public String resolveTypeReference() throws LiferayTypeMappingException {

if(typeReference==null) {

LiferayType typeAnnotation = getFieldContainerClass().getAnnotation(LiferayType.class);

if (typeAnnotation == null) {

throw new LiferayTypeMappingException(String.format("Class %s must be annotated with %s annotation", getFieldContainerClass(), LiferayType.class));

}

String ref = typeAnnotation.reference();

if (ref == null) {

ref = typeAnnotation.title();

}

typeReference = ref;

}

return typeReference;

}

public Sort[] getDefaultSort() throws LiferayTypeMappingException {

if(defSort ==null) {

SortBuilderFactory sortBuilderFactory = LiferayDocumentRepository.getSortBuilderFactory();

LiferayType typeAnnotation = getFieldContainerClass().getAnnotation(LiferayType.class);

if (typeAnnotation == null) {

//Commented out to allow list \ search basic (untyped) liferay documents

//throw new LiferayTypeMappingException(String.format("Class %s must be annotated with %s annotation", getFieldContainerClass(), LiferayType.class));

defSort = new Sort[]{ sortBuilderFactory.getSortBuilder().field(Field.TITLE).sortOrder(SortOrder.ASC).build() };

} else {

DefaultSortOrder[] sortBy = typeAnnotation.defaultSortBy();

defSort = Arrays.stream(sortBy)

.map(dsb -> sortBuilderFactory.getSortBuilder().field(dsb.by()).sortOrder(dsb.order()).build())

.toArray(Sort[]::new);

}

}

return defSort;

}

private static DLFileEntryType resolveTypeByTitle(@NonNull final String typeTitle) throws PortalException {

List<DLFileEntryType> types = DLFileEntryTypeLocalServiceUtil.getDLFileEntryTypes(0, DLFileEntryTypeLocalServiceUtil.getDLFileEntryTypesCount());

return types.stream().filter(t -> t.getNameMap().containsValue(typeTitle)).findFirst().orElseThrow(

() -> new LiferayTypeMappingException(String.format("Cannot find type by title %s", typeTitle))

);

}

private static boolean isTypeReferencedBy(final @NonNull DLFileEntryType type, final @NonNull String reference){

return reference.equals(resolveTypeReference(type));

}

private static DLFileEntryType resolveTypeByReference(@NonNull final String reference) throws PortalException {

List<DLFileEntryType> types = DLFileEntryTypeLocalServiceUtil.getDLFileEntryTypes(0, DLFileEntryTypeLocalServiceUtil.getDLFileEntryTypesCount());

return types.stream().filter(t -> isTypeReferencedBy(t, reference)).findFirst().orElseThrow(

() -> new LiferayTypeMappingException(String.format("Cannot find type by reference %s", reference))

);

}

private static DLFileEntryType resolveType(final @NonNull Class<?> clazz) throws PortalException, IllegalArgumentException {

if(BasicLiferayFileDocument.class.isAssignableFrom(clazz)){

return DLFileEntryTypeLocalServiceUtil.getDLFileEntryType(DLFileEntryTypeConstants.FILE\_ENTRY\_TYPE\_ID\_BASIC\_DOCUMENT);

}

LiferayType typeAnnotation = clazz.getAnnotation(LiferayType.class);

if(typeAnnotation==null){

throw new LiferayTypeMappingException(String.format("Class %s must be annotated with %s annotation", clazz, LiferayType.class));

}

String ref = typeAnnotation.reference();

DLFileEntryType type;

if(!StringUtils.isEmpty(ref)) {

type = resolveTypeByReference(ref);

} else {

type = resolveTypeByTitle(typeAnnotation.title());

log.warn("DLFileEntryType resolved by title '{}'. It is strongly recommended to use resolution by reference", typeAnnotation.title());

}

return type;

}

}

package ru.rvdsystems.liferay.impl.mediator.annotated;

import com.liferay.document.library.kernel.model.DLFileEntry;

import com.liferay.document.library.kernel.model.DLFileEntryType;

import com.liferay.document.library.kernel.service.DLFileEntryLocalServiceUtil;

import com.liferay.document.library.kernel.service.DLFileEntryTypeLocalServiceUtil;

import com.liferay.dynamic.data.mapping.kernel.DDMFormField;

import com.liferay.dynamic.data.mapping.kernel.DDMFormFieldValue;

import com.liferay.dynamic.data.mapping.kernel.LocalizedValue;

import com.liferay.dynamic.data.mapping.kernel.UnlocalizedValue;

import com.liferay.dynamic.data.mapping.kernel.Value;

import com.liferay.osgi.util.ServiceTrackerFactory;

import com.liferay.portal.kernel.exception.PortalException;

import com.liferay.portal.kernel.search.Document;

import com.liferay.portal.search.sort.Sort;

import lombok.AccessLevel;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.NonNull;

import lombok.extern.slf4j.Slf4j;

import org.osgi.framework.FrameworkUtil;

import org.osgi.util.tracker.ServiceTracker;

import ru.rvdsystems.liferay.api.Util;

import ru.rvdsystems.liferay.api.annotation.ConfigurationValueAccessor;

import ru.rvdsystems.liferay.api.fieldentity.DocumentLibraryLink;

import ru.rvdsystems.liferay.api.fieldentity.Geolocation;

import ru.rvdsystems.liferay.api.fieldentity.ImageLink;

import ru.rvdsystems.liferay.api.fieldentity.Option;

import ru.rvdsystems.liferay.api.fieldentity.PageLink;

import ru.rvdsystems.liferay.api.repository.DDMFieldContainer;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import ru.rvdsystems.liferay.api.repository.LiferayDocumentRepository;

import ru.rvdsystems.liferay.api.repository.exception.validation.LiferayEmptyRequiredFieldException;

import ru.rvdsystems.liferay.api.repository.exception.LiferayFieldMappingException;

import ru.rvdsystems.liferay.api.repository.exception.validation.LiferayFieldValidationException;

import ru.rvdsystems.liferay.api.repository.exception.LiferayIncompatibleMediatorException;

import ru.rvdsystems.liferay.api.repository.exception.LiferayIndexConfigurationException;

import ru.rvdsystems.liferay.api.repository.exception.LiferayMappingException;

import ru.rvdsystems.liferay.api.repository.exception.UnsupportedException;

import ru.rvdsystems.liferay.api.repository.exception.validation.LiferayIncorrectLinkedTypeException;

import ru.rvdsystems.liferay.impl.mediator.upload.DelayedDocumentContainer;

import ru.rvdsystems.liferay.impl.mediator.DDMFieldContainerMediator;

import ru.rvdsystems.liferay.impl.mediator.DDMFieldMediator;

import ru.rvdsystems.liferay.impl.repository.ImplUtil;

import ru.rvdsystems.liferay.impl.repository.JsonUtil;

import ru.rvdsystems.liferay.impl.repository.cache.InternalCacheLocalService;

import java.beans.BeanInfo;

import java.beans.IntrospectionException;

import java.beans.Introspector;

import java.beans.PropertyDescriptor;

import java.io.Serializable;

import java.lang.reflect.InvocationTargetException;

import java.sql.Date;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Locale;

import java.util.Map;

import java.util.Objects;

import java.util.Optional;

import java.util.stream.Collectors;

import java.util.stream.Stream;

import static ru.rvdsystems.liferay.impl.repository.DDMFormFieldTypeConstants.\*;

@Slf4j

@EqualsAndHashCode(callSuper = true)

public class AnnotatedDDMFieldMediator extends DDMFieldMediator implements Serializable {

private static final ServiceTracker<InternalCacheLocalService, InternalCacheLocalService>

cacheServiceTracker = ServiceTrackerFactory.open(FrameworkUtil.getBundle(ServiceTrackerFactory.class), InternalCacheLocalService.class);

private final Map<String, ClassFieldInfo> byTitle = new HashMap<>();

private final Map<String, ClassFieldInfo> byReference = new HashMap<>();

@Getter(AccessLevel.PACKAGE)

private final Map<String, ClassFieldInfo> byName = new HashMap<>();

@Getter(AccessLevel.PACKAGE)

private final Map<String, ClassFieldInfo> indexeds = new LinkedHashMap<>();

static AnnotatedDDMFieldMediator getInstance(Class<? extends DDMFieldContainer> clazz) throws LiferayMappingException {

return new AnnotatedDDMFieldMediator(clazz);

}

private AnnotatedDDMFieldMediator(Class<? extends DDMFieldContainer> clazz) throws LiferayMappingException {

super(clazz);

log.debug("Scanning class {}", clazz);

BeanInfo beanInfo;

try {

beanInfo = Introspector.getBeanInfo(clazz);

} catch (IntrospectionException e) {

throw new LiferayMappingException(String.format("Cannot read bean introspection info from class %s",clazz),e);

}

for(PropertyDescriptor pd : beanInfo.getPropertyDescriptors()){

ClassFieldInfo fieldInfo = new ClassFieldInfo(clazz, pd);

log.trace("Processing property {}#{}...", clazz.getName(), fieldInfo.getName());

byName.put(fieldInfo.getName(), fieldInfo);

if(fieldInfo.isLiferayField()){

log.debug("Found @LiferayField property {}#{}", clazz.getName(), fieldInfo.getName());

String ref;

if( ! Util.isEmpty(fieldInfo.getLiferayFieldReference())){

ref = fieldInfo.getLiferayFieldReference();

log.trace("PUT field {} by reference {}", fieldInfo.getName(), ref);

byReference.put(ref,fieldInfo);

} else if ( !Util.isEmpty(fieldInfo.getLiferayFieldTitle())){

ref = fieldInfo.getLiferayFieldTitle();

byTitle.put(ref,fieldInfo);

log.trace("PUT field {} by title {}", fieldInfo.getName(), ref);

} else {

// use property name as reference

ref = fieldInfo.getName();

byReference.put(ref, fieldInfo);

log.trace("PUT field {} by poproperty name as a reference: {}", ref, ref);

}

}

if(fieldInfo.isLiferayIndexed()){

indexeds.put(fieldInfo.getIndexedName()!=null ? fieldInfo.getIndexedName() : fieldInfo.getName(), fieldInfo);

}

}

}

@Override

public List<DDMFormFieldValue> get(DDMFieldContainer entry, DDMFormField ddmFormField) throws LiferayMappingException, LiferayIncompatibleMediatorException {

checkCompatibility(entry);

ClassFieldInfo fieldInfo = findField(ddmFormField);

if(fieldInfo == null){

return fromNullValue(ddmFormField);

}

checkExpectedValueClass(ddmFormField, fieldInfo);

Object value= getValue(entry, fieldInfo);

return get(ddmFormField, value);

}

@Override

public void set(final DDMFieldContainer entry, final DDMFormField ddmFormField, final List<DDMFormFieldValue> ddmFormFieldValues) throws LiferayMappingException, LiferayIncompatibleMediatorException {

checkCompatibility(entry);

if(log.isDebugEnabled()) {

log.debug("SETTING {} values {}",

ddmFormField.getLabel().getString(ddmFormField.getLabel().getDefaultLocale()),

ddmFormFieldValues != null ? ddmFormFieldValues.stream().map(ImplUtil::getNonEmptyValue).collect(Collectors.toList()) : "NULL"

);

}

if(ddmFormFieldValues == null || ddmFormFieldValues.isEmpty()){

return;

}

final ClassFieldInfo fieldInfo = findField(ddmFormField);

if (fieldInfo != null) {

checkExpectedValueClass(ddmFormField, fieldInfo);

final boolean repeatable = ddmFormField.isRepeatable();

final boolean multiple = ImplUtil.isMultiple(ddmFormField);

Object value;

final Class<?> destClass = fieldInfo.getPropertyType();

if (!(repeatable)) {

if (ddmFormFieldValues.size() > 1) {

throw new LiferayFieldMappingException(String.format(

"Unexpectedly got repeatable values for non-repeatable field %s in class %s", fieldInfo.getName(), this.getFieldContainerClass()));

}

if (!multiple) {

value = constructSingleValue(destClass, ddmFormField, ddmFormFieldValues.get(0));

} else {

value = constructMultipleValueList(fieldInfo.getListParameterClass(), ddmFormField, ddmFormFieldValues.get(0));

}

} else {

value = constructRepeatableValueList(fieldInfo, ddmFormField, ddmFormFieldValues, multiple);

}

setValue(entry,fieldInfo,value);

}

}

@Override

public void populateIndexedFields(DDMFieldContainer container, Document document) throws LiferayIncompatibleMediatorException, LiferayFieldMappingException {

checkCompatibility(container);

IndexerUtil.populateIndexedFields(container, document, this);

}

@Override

public Sort[] prepareSorts(Sort[] sorts) throws LiferayIndexConfigurationException {

return IndexerUtil.prepareSorts(sorts, this);

}

@Override

public Class<?> getIndexedType(final String indexedFieldName) {

final ClassFieldInfo info = indexeds.get(indexedFieldName);

return info != null ?

info.isList() ? info.getListParameterClass()

: info.isListOfList() ? info.getListSublistParameterClass()

: info.getPropertyType()

: null;

}

@Override

public List<DelayedDocumentContainer> set(@NonNull DDMFieldContainer container, @NonNull Map.Entry<String,Object> entry) throws PortalException {

checkCompatibility(container);

return RequestParameterMapUtil.setContainerProperty(container,entry, this);

}

@Override

public void verifyFields(@NonNull DDMFieldContainer entry) throws PortalException {

checkCompatibility(entry);

for(ClassFieldInfo fieldInfo : getByName().values()){

if(fieldInfo.isRequired()){

if(getValue(entry,fieldInfo)==null){

throw new LiferayEmptyRequiredFieldException(String.format("Required field %s#%s is null: %s", entry.getClass().getName(),fieldInfo.getName(), entry));

}

}

if(

(fieldInfo.getLinkedTypes()!=null && fieldInfo.getLinkedTypes().length > 0)

|| (fieldInfo.getLinkedTypeFolders()!=null && fieldInfo.getLinkedTypeFolders().length > 0)

){

Object value = getValue(entry,fieldInfo);

if(value!=null){

validateLinkedTypes(fieldInfo, value);

}

}

}

}

Object getValue(@NonNull DDMFieldContainer entry, @NonNull ClassFieldInfo fieldInfo) {

try {

return fieldInfo.getReadMethod().invoke(entry);

} catch (IllegalAccessException | InvocationTargetException e) {

throw new IllegalStateException(String.format(

"Unable to get value from entry of class %s using field %s",

entry.getClass(),

this.getFieldContainerClass().getName()+"#"+fieldInfo.getName()

), e);

}

}

void setValue(@NonNull DDMFieldContainer entry, @NonNull ClassFieldInfo fieldInfo, Object value){

try {

log.debug("Setting bean property {}:{} to value {}", fieldInfo.getDeclaringClass().getName(), fieldInfo.getName(), value);

fieldInfo.getWriteMethod().invoke(entry, value);

} catch (IllegalAccessException | InvocationTargetException e) {

throw new IllegalStateException(String.format("Unexpectedly cannot set field %s#%s to value %s", entry.getClass().getName(), fieldInfo.getName(), value), e);

}

}

private static void validateLinkedTypes(ClassFieldInfo fieldInfo, Object value) throws PortalException {

List<Object> values = new ArrayList<>();

if(fieldInfo.isListOfList()){

((List<?>)value).forEach(subList -> values.addAll(((List<?>) subList)));

} else if(fieldInfo.isList()){

values.addAll((List<?>)value);

} else {

values.add(value);

}

for(Object o : values){

validateLinkedObject(fieldInfo, o);

}

}

private static void validateLinkedObject(ClassFieldInfo fieldInfo, Object scalar) throws PortalException {

log.info("Validating fild {}#{} linked object {}", fieldInfo.getDeclaringClass().getName(), fieldInfo.getName(), scalar);

long fileEntryId;

if(scalar instanceof DocumentLibraryLink){

fileEntryId = Long.parseLong(((DocumentLibraryLink)scalar).getFileEntryId());

} else if(scalar instanceof ImageLink) {

fileEntryId = Long.parseLong(((ImageLink)scalar).getFileEntryId());

} else {

return;

}

DLFileEntry fileEntry = DLFileEntryLocalServiceUtil.getFileEntry(fileEntryId);

if(fieldInfo.getLinkedTypes() != null && fieldInfo.getLinkedTypes().length > 0) {

long linkedTypeId = fileEntry.getFileEntryTypeId();

DLFileEntryType entryType = DLFileEntryTypeLocalServiceUtil.getDLFileEntryType(linkedTypeId);

//String typeRef = LiferayDocumentRepository.getTypeReference(entryType);

//log.info("linkedTypeId: {}, found type ref: {}", linkedTypeId, typeRef);

//Class<? extends LiferayDocument> clazz = cacheServiceTracker.getService().getRegisteredLiferayClassByReference(typeRef);

Class<? extends LiferayDocument> clazz = LiferayDocumentRepository.findRegisteredClass(entryType);

if (Arrays.stream(fieldInfo.getLinkedTypes()).noneMatch(linkedClass -> linkedClass.isAssignableFrom(clazz))) {

throw new LiferayIncorrectLinkedTypeException(String.format(

"Field %s#%s does not allow links to %s. Allowed linked types: %s",

fieldInfo.getDeclaringClass().getName(), fieldInfo.getName(), clazz.getName(), Arrays.asList(fieldInfo.getLinkedTypes()))

);

}

}

if(fieldInfo.getLinkedTypeFolders() != null && fieldInfo.getLinkedTypeFolders().length > 0){

List<Long> ids = Arrays.stream(fieldInfo.getLinkedTypeFolders()).map(AnnotatedDDMFieldMediator::getFolderIdFromConfiguration).filter(Objects::nonNull).collect(Collectors.toList());

if(! ids.isEmpty()){

if(ids.contains(fileEntry.getFolderId())){

return;

}

List<Long> ancestorIds = fileEntry.getFolder().getAncestorFolderIds();

if(ids.stream().noneMatch(ancestorIds::contains)){

throw new LiferayIncorrectLinkedTypeException(String.format(

"Field %s#%s allows only linked entries from folders %s",

fieldInfo.getDeclaringClass().getName(), fieldInfo.getName(), ids)

);

}

}

}

}

static DDMFieldContainerMediator getFieldContainerMediator(Class<? extends DDMFieldContainer> destClass) throws LiferayMappingException {

return cacheServiceTracker.getService().getDDMFieldContainerMediatorInstance(destClass);

}

private static Long getFolderIdFromConfiguration(Class<? extends ConfigurationValueAccessor<Long>> confClass){

try {

return confClass.getConstructor(null).newInstance(null).getValue();

} catch (InstantiationException | IllegalAccessException | InvocationTargetException | NoSuchMethodException e) {

return null;

}

}

private void checkCompatibility(DDMFieldContainer entry) throws LiferayIncompatibleMediatorException {

if( !getFieldContainerClass().isAssignableFrom(entry.getClass()) ){

throw new LiferayIncompatibleMediatorException(

String.format("Attempt to map class %s by incompatible mediator for class %s", entry.getClass(), getFieldContainerClass())

);

}

}

private static List<DDMFormFieldValue> get(DDMFormField ddmFormField, Object value) throws LiferayMappingException, LiferayIncompatibleMediatorException {

if(value == null){

return fromNullValue(ddmFormField);

}

final boolean repeatable = ddmFormField.isRepeatable();

final boolean multiple = ImplUtil.isMultiple(ddmFormField);

List<DDMFormFieldValue> result = new ArrayList<>();

if(!repeatable && !multiple){

//single value

result.add(fromSingleValue(ddmFormField, value));

} else if (!repeatable){

//Multiple value

result.add(fromMultipleValue(ddmFormField, value));

} else {

//repeatable (and possibly multiple) value

result.addAll(fromRepeatableValue(ddmFormField, value));

}

if(log.isTraceEnabled()) {

logDmmValues(ddmFormField, result);

}

return result;

}

private static List<DDMFormFieldValue> fromNullValue(DDMFormField ddmFormField){

switch (ddmFormField.getType()){

case IMAGE:

case DOCUMENT\_LIBRARY:

case GEOLOCATION:

case LINK\_TO\_LAYOUT:

return Collections.singletonList(createDdmStringValue(ddmFormField, "{}"));

default:

return null;

}

}

private static void logDmmValues(DDMFormField field, List<DDMFormFieldValue> result){

log.trace("Constructed ddm values for field {} ({}): {}", field.getLabel().getString(field.getLabel().getDefaultLocale()), field.getName(), result==null ? "null" : result.isEmpty() ? "<EMPTY LIST>" : "");

if(result !=null) {

result.forEach(v -> {

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

List<String> vals = v.getValue().getValues().entrySet().stream().map(e -> e.getKey() + "='" + e.getValue() + "'").collect(Collectors.toList());

log.trace("\t{}", vals);

} else {

List<DDMFormFieldValue> nested = v.getNestedDDMFormFieldValues();

if(nested !=null && ! nested.isEmpty()){

log.trace("\tNested Values length: {}", nested.size());

} else {

log.trace("\tUNKNOWN");

}

}

});

}

}

private static List<DDMFormFieldValue> fromRepeatableValue(final DDMFormField ddmFormField, final Object value) throws LiferayMappingException, LiferayIncompatibleMediatorException {

List<Object> repeatableValues;

try {

repeatableValues = (List<Object>) value;

} catch (ClassCastException e) {

throw new LiferayFieldMappingException(String.format("Unexpected non-list value %s (class %s) for repeatable ddm field %s",

value, value.getClass(), ddmFormField.getLabel().getString(ddmFormField.getLabel().getDefaultLocale())

));

}

final boolean isMultiple = ImplUtil.isMultiple(ddmFormField);

List<DDMFormFieldValue> result = new ArrayList<>();

for (Object v : repeatableValues) {

result.add(isMultiple ? fromMultipleValue(ddmFormField, v) : fromSingleValue(ddmFormField, v));

}

return result;

}

private static DDMFormFieldValue fromMultipleValue(DDMFormField ddmFormField, Object value) throws LiferayFieldMappingException {

List<Object> multipleValues;

try {

multipleValues = (List<Object>) value;

} catch (ClassCastException e){

throw new LiferayFieldMappingException(String.format("Unexpected non-list value %s (class %s) for multiple ddm field %s",

value, value.getClass(), ddmFormField.getLabel().getString(ddmFormField.getLabel().getDefaultLocale())

));

}

List<String> stringValues = new ArrayList<>();

for( Object v: multipleValues) {

stringValues.add(singleValueToString(ddmFormField,v));

}

String arrayJson = JsonUtil.toJson(stringValues);

return createDdmStringValue(ddmFormField, arrayJson);

}

private static DDMFormFieldValue fromSingleValue(final DDMFormField ddmFormField, final Object value) throws LiferayMappingException, LiferayIncompatibleMediatorException {

if(value instanceof DDMFieldContainer){

final DDMFieldContainerMediator fieldSetMediator = cacheServiceTracker.getService().getDDMFieldContainerMediatorInstance((Class<? extends DDMFieldContainer>) value.getClass());

final DDMFormFieldValue result = new DDMFormFieldValue();

result.setName(ddmFormField.getName());

result.setNestedDDMFormFields(

fieldSetMediator.populateDDMStructure((DDMFieldContainer) value, ddmFormField.getNestedDDMFormFields())

);

return result;

}

String stringValue = singleValueToString(ddmFormField, value);

if(SELECT.equals(ddmFormField.getType())){

stringValue = JsonUtil.toJson(Collections.singletonList(stringValue));

}

return createDdmStringValue(ddmFormField, stringValue);

}

private static String singleValueToString(final DDMFormField ddmFormField, final Object value) throws LiferayFieldMappingException {

if(value == null){

return "";

}

if(value instanceof String|| value instanceof Integer || value instanceof Double || value instanceof Date){

return value.toString();

} else if(value instanceof DocumentLibraryLink || value instanceof ImageLink || value instanceof Geolocation || value instanceof PageLink){

return JsonUtil.toJson(value);

} else if( value instanceof Option){

final String optionTitle = ((Option)value).getTitle();

Optional<String> optValue = ddmFormField.getDDMFormFieldOptions().getOptions().entrySet().stream()

.filter(entry -> isOptionHaslabel(entry, optionTitle)).map(Map.Entry::getKey)

.findFirst();

if(! optValue.isPresent()){

throw new LiferayFieldMappingException(String.format("Unexpectedly cannot map option '%s' to DDM field '%s' option",

optionTitle, ddmFormField.getLabel().getString(ddmFormField.getLabel().getDefaultLocale())

));

}

return optValue.get();

} else {

throw new LiferayFieldMappingException(String.format("Unexpected value %s (class %s) for DDM field %s (type: %s)",

value,

value!=null ? value.getClass() : "null",

ddmFormField.getLabel().getString(ddmFormField.getLabel().getDefaultLocale()),

ddmFormField.getType()

));

}

}

private static boolean isOptionHaslabel(final Map.Entry<String, LocalizedValue> entry, final String title){

return entry.getValue().getValues().values().stream().anyMatch(title::equals);

}

private static DDMFormFieldValue createDdmStringValue(final DDMFormField ddmFormField, final String value){

final DDMFormFieldValue ddmFormFieldValue = new DDMFormFieldValue();

ddmFormFieldValue.setName(ddmFormField.getName());

final Value ddmValue;

if(ddmFormField.isLocalizable()){

ddmValue = new LocalizedValue();

Locale defaultLocale = ddmFormField.getDDMForm().getDefaultLocale();

ddmValue.setDefaultLocale(defaultLocale);

ddmFormField.getDDMForm().getAvailableLocales().forEach( locale -> ddmValue.addString(locale,value));

} else {

ddmValue = new UnlocalizedValue(value);

}

ddmFormFieldValue.setValue(ddmValue);

return ddmFormFieldValue;

}

private Class<?> checkExpectedValueClass(DDMFormField ddmField, ClassFieldInfo fieldInfo) throws LiferayFieldMappingException {

Class<?> expectedValueClass = cacheServiceTracker.getService().getDdmFieldCheckedClass(ddmField, fieldInfo);

if (expectedValueClass == null) {

switch (ddmField.getType()) {

case TEXT:

case RICH\_TEXT:

case COLOR: //todo

expectedValueClass = String.class;

break;

case SELECT:

case RADIO:

case CHECKBOX\_MULTIPLE:

expectedValueClass = Option.class;

break;

case GRID:

//TODO

expectedValueClass = String.class;

break;

case DATE:

expectedValueClass = Date.class;

break;

case NUMERIC:

switch (ddmField.getDataType()) {

case "integer":

expectedValueClass = Integer.class;

break;

case "double":

expectedValueClass = Double.class;

break;

default:

throw new UnsupportedException(String.format("Unsupported DDM structure field numeric data type %s", ddmField.getDataType()));

}

break;

case IMAGE:

expectedValueClass = ImageLink.class;

break;

case DOCUMENT\_LIBRARY:

expectedValueClass = DocumentLibraryLink.class;

break;

case GEOLOCATION:

expectedValueClass = Geolocation.class;

break;

case LINK\_TO\_LAYOUT:

expectedValueClass = PageLink.class;

break;

case FIELDSET:

expectedValueClass = DDMFieldContainer.class;

break;

default:

throw new UnsupportedException(String.format("Unsupported DDM structure field data type %s", ddmField.getType()));

}

checkFieldAgainstClass(fieldInfo, ddmField, expectedValueClass);

cacheServiceTracker.getService().setDdmFieldCheckedClass(ddmField,fieldInfo,expectedValueClass);

}

return expectedValueClass;

}

private void checkFieldAgainstClass(final ClassFieldInfo fieldInfo, final DDMFormField ddmField, final Class<?> ofClass) throws LiferayFieldMappingException {

final boolean repeatable = ddmField.isRepeatable();

final boolean multiple = ImplUtil.isMultiple(ddmField);

//Field resultField = fieldInfo.getField();

if( !(! multiple && ! repeatable && fieldInfo.isClassOf(ofClass) )

&& ! ( (multiple ^ repeatable) && fieldInfo.isListOf(ofClass))

&& ! ( multiple && repeatable && fieldInfo.isListOfListOf(ofClass))

){

throw new LiferayFieldMappingException(

String.format("Class %s field %s is not compatible with mapped DDM field '%s': field type is %s, expected %s",

this.getFieldContainerClass(),

fieldInfo.getName(),

ddmField.getLabel().getString(ddmField.getLabel().getDefaultLocale()),

fieldInfo.getPropertyType()

+ (fieldInfo.getListParameterClass()!=null ? "<"+fieldInfo.getListParameterClass()

+ (fieldInfo.getListSublistParameterClass()!=null ? "<"+fieldInfo.getListSublistParameterClass()+">" : "")

+">" : ""),

multiple && repeatable? String.format("List<List<%s>> (because ddm field is both repeatable & multiple)", ofClass)

: multiple ^ repeatable ? String.format("List<%s>, because ddm field is %s", ofClass, multiple ? "multiple" : "repeatable")

: ofClass)

);

}

}

private ClassFieldInfo findField(DDMFormField field){

String ref = field.getFieldReference();

log.debug("finding by ddm field {}", ref);

ClassFieldInfo fieldInfo = null;

if(byReference.containsKey(ref)){

fieldInfo = byReference.get(ref);

log.debug("FOUND by ddm field ref {}: {}", ref, fieldInfo.getName());

} else {

Optional<String> foundTitle = field.getLabel().getValues().values().stream().filter(byTitle::containsKey).findFirst();

if(foundTitle.isPresent()){

fieldInfo = byTitle.get(foundTitle.get());

log.debug("Found by DDM field title {}: {}", foundTitle.get(), fieldInfo.getName());

}

}

return fieldInfo;

}

private static List<Object> constructRepeatableValueList(final ClassFieldInfo fieldInfo, final DDMFormField ddmField, final List<DDMFormFieldValue> ddmValues, final boolean isMultiple) throws LiferayMappingException, LiferayIncompatibleMediatorException {

List<Object> result = new ArrayList<>();

for( DDMFormFieldValue ddmFormFieldValue : ddmValues){

result.add( isMultiple ? constructMultipleValueList(fieldInfo.getListSublistParameterClass(), ddmField, ddmFormFieldValue) : constructSingleValue(fieldInfo.getListParameterClass(), ddmField, ddmFormFieldValue) );

}

return result;

}

private static List<Object> constructMultipleValueList(final Class<?> destClass, final DDMFormField ddmFormField, final DDMFormFieldValue ddmValue) throws LiferayFieldMappingException {

String multiValue = getDdmValueString(ddmValue);

if(multiValue !=null) {

List<Object> result = new ArrayList<>();

String[] values = JsonUtil.readValue(getDdmValueString(ddmValue), String[].class);

for (String v : values) {

result.add(constructSingleValue(destClass, ddmFormField, v));

}

return result;

}

return Collections.emptyList();

}

private static String getDdmValueString(DDMFormFieldValue ddmFormFieldValue){

//TODO locales

//return Util.getDefaultLocaleValue(ddmFormFieldValue.getValue());

return ImplUtil.getNonEmptyValue(ddmFormFieldValue);

}

private static Object constructSingleValue(final Class<?> destClass, final DDMFormField ddmField, final DDMFormFieldValue ddmValue) throws LiferayMappingException, LiferayIncompatibleMediatorException {

if(DDMFieldContainer.class.isAssignableFrom(destClass)) {

//DDMFieldContainerMediator fieldSetMediator = cacheServiceTracker.getService().getDDMFieldContainerMediatorInstance((Class<? extends DDMFieldContainer>) destClass);

DDMFieldContainerMediator fieldSetMediator = getFieldContainerMediator((Class<? extends DDMFieldContainer>) destClass);

DDMFieldContainer fieldSetObj = fieldSetMediator.createEntry();

fieldSetMediator.readDDMStructure(fieldSetObj, ddmField.getNestedDDMFormFields(),ddmValue.getNestedDDMFormFieldValues());

return fieldSetObj;

}

return constructSingleValue(destClass, ddmField, getDdmValueString(ddmValue) );

}

private static Object constructSingleValue(final Class<?> destClass, final DDMFormField ddmField, final String value) throws LiferayFieldMappingException {

if(value==null){

return null;

}

if(String.class==destClass){

return value;

} else if(Option.class.isAssignableFrom(destClass)){

LocalizedValue labels = ddmField.getDDMFormFieldOptions().getOptionLabels(ImplUtil.getSingleOptionsValue(value));

if(labels == null){

throw new LiferayFieldMappingException(String.format("Unexpectedly cannot get option labels for options field '%s' value '%s'", ddmField.getLabel().getString(ddmField.getLabel().getDefaultLocale()), value));

}

return Option.fromLabel(labels.getString(labels.getDefaultLocale()), (Class<Option>) destClass);

} else if(Integer.class==destClass){

return Integer.valueOf(value);

} else if(Double.class==destClass){

return Double.valueOf(value.replace(",","."));

} else if(Date.class==destClass){

return Date.valueOf(value);

} else if(DocumentLibraryLink.class==destClass){

return JsonUtil.readValue(value, DocumentLibraryLink.class);

} else if(ImageLink.class == destClass){

return JsonUtil.readValue(value, ImageLink.class);

} else if(Geolocation.class==destClass){

return JsonUtil.readValue(value, Geolocation.class);

} else if(PageLink.class == destClass){

return JsonUtil.readValue(value, PageLink.class);

} else {

throw new LiferayFieldMappingException(String.format("Unexpected destination field class %s", destClass));

}

}

}

package ru.rvdsystems.liferay.impl.mediator.annotated;

import lombok.AccessLevel;

import lombok.Getter;

import lombok.ToString;

import lombok.extern.slf4j.Slf4j;

import ru.rvdsystems.liferay.api.Util;

import ru.rvdsystems.liferay.api.annotation.ConfigurationValueAccessor;

import ru.rvdsystems.liferay.api.annotation.LiferayField;

import ru.rvdsystems.liferay.api.annotation.LiferayIndexed;

import ru.rvdsystems.liferay.api.fieldentity.Option;

import ru.rvdsystems.liferay.api.repository.DDMFieldContainer;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import ru.rvdsystems.liferay.api.repository.exception.LiferayFieldMappingException;

import java.beans.PropertyDescriptor;

import java.lang.annotation.Annotation;

import java.lang.reflect.Field;

import java.lang.reflect.Method;

import java.lang.reflect.ParameterizedType;

import java.lang.reflect.Type;

import java.util.List;

@Getter

@Slf4j

@ToString

public class ClassFieldInfo {

//private final PropertyDescriptor descriptor;

private final Class<?> propertyType;

private final String name;

private final Method readMethod;

private final Method writeMethod;

private boolean liferayField;

private String liferayFieldReference;

private String liferayFieldTitle;

private boolean required;

private Class<? extends LiferayDocument>[] linkedTypes;

private Class<? extends ConfigurationValueAccessor<Long>>[] linkedTypeFolders;

private boolean allowUpload;

private String uploadTitle;

private boolean liferayIndexed;

private boolean sortable;

private boolean keyword;

private String indexedName;

private Class<?> listParameterClass;

@Getter(AccessLevel.NONE)

private Boolean isList;

@Getter(AccessLevel.NONE)

private Boolean isListOfList;

public String getName() {

return name;

}

public Class<?> getDeclaringClass() {

return getReadMethod().getDeclaringClass();

}

public Class<?> getListParameterClass() {

if (isList != null) {

return isList ? listParameterClass : null;

}

if (List.class.isAssignableFrom(getPropertyType())) {

try {

ParameterizedType parameterizedType = (ParameterizedType) getReadMethod().getGenericReturnType();

Type listclass = parameterizedType.getActualTypeArguments()[0];

if (listclass instanceof Class<?>) {

isList = true;

listParameterClass = (Class<?>) listclass;

} else {

ParameterizedType pType = (ParameterizedType) listclass;

isList = true;

listParameterClass = (Class<?>) pType.getRawType();

}

} catch (ClassCastException e) {

isList = false;

}

}

if (isList == null) {

isList = false;

}

return isList ? listParameterClass : null;

}

public boolean isList(){

return isList!=null ? isList : getListParameterClass() != null;

}

public boolean isListOfList(){

return isListOfList!=null ? isListOfList : getListSublistParameterClass() !=null;

}

public Class<?> getListSublistParameterClass() {

if (isListOfList != null) {

return isListOfList ? listParameterClass : null;

}

if (List.class.isAssignableFrom(getPropertyType())) {

try {

ParameterizedType parameterizedType = (ParameterizedType) getReadMethod().getGenericReturnType();

Type listclass = parameterizedType.getActualTypeArguments()[0];

if (listclass instanceof Class) {

isListOfList = false;

} else {

ParameterizedType pType = (ParameterizedType) listclass;

if (List.class.isAssignableFrom((Class) pType.getRawType())) {

isListOfList = true;

listParameterClass = (Class) pType.getActualTypeArguments()[0];

}

}

} catch (ClassCastException | ArrayIndexOutOfBoundsException e) {

isListOfList = false;

}

}

if (isListOfList == null) {

isListOfList = false;

}

return isListOfList ? listParameterClass : null;

}

public boolean isClassOf(Class<?> ofType) {

return ofType == Option.class || ofType == DDMFieldContainer.class ? ofType.isAssignableFrom(getPropertyType()) : ofType == getPropertyType();

}

public boolean isListOf(Class<?> ofClass) {

Class<?> listClass = getListParameterClass();

return listClass != null && ofClass.isAssignableFrom(listClass);

}

public boolean isListOfListOf(Class<?> ofClass) {

Class<?> subListClass = getListSublistParameterClass();

return subListClass != null && ofClass.isAssignableFrom(subListClass);

}

ClassFieldInfo(Class<?> parentClass, PropertyDescriptor pd/\*, Field field\*/) throws LiferayFieldMappingException {

propertyType = pd.getPropertyType();

name = pd.getName();

if (pd.getReadMethod() == null) {

throw new LiferayFieldMappingException(String.format("Cannot get getter method for property %s#%s", parentClass, pd.getName()));

}

readMethod = pd.getReadMethod();

writeMethod = pd.getWriteMethod();

LiferayField fieldAnn = getAnnotation(pd, LiferayField.class);

if (fieldAnn != null) {

if (pd.getWriteMethod() == null) {

throw new LiferayFieldMappingException(String.format("Cannot get setter method for property %s#%s", parentClass, pd.getName()));

}

liferayField = true;

liferayFieldReference = fieldAnn.reference();

liferayFieldTitle = fieldAnn.title();

required = fieldAnn.required();

linkedTypes = fieldAnn.linkedTypes();

linkedTypeFolders= fieldAnn.linkedTypeFolders();

allowUpload = fieldAnn.allowUpload();

uploadTitle = fieldAnn.uploadTitle();

}

LiferayIndexed indexedAnn = getAnnotation(pd, LiferayIndexed.class);

if (indexedAnn != null) {

liferayIndexed = true;

sortable = indexedAnn.sortable();

keyword = indexedAnn.keyword();

indexedName = Util.isEmpty(indexedAnn.name()) ? null : indexedAnn.name();

}

}

static <T extends Annotation> T getAnnotation(PropertyDescriptor pd, Class<T> annClass) {

T result = null;

log.trace("Finding annotation {} in property {}...", annClass.getName(), pd.getName());

try {

log.trace("Finding annotation {} in field {}#{}...", annClass.getName(), pd.getReadMethod().getDeclaringClass().getName(), pd.getName());

Field field = pd.getReadMethod().getDeclaringClass().getDeclaredField(pd.getName());

result = field.getAnnotation(annClass);

} catch (NoSuchFieldException e) {

//

}

log.trace("Annotation {} in field {}#{} found: {}", annClass.getName(), pd.getReadMethod().getDeclaringClass().getName(), pd.getName(), result);

return result != null ? result : pd.getReadMethod().getAnnotation(annClass);

}

}

package ru.rvdsystems.liferay.impl.mediator.annotated;

import com.liferay.portal.kernel.search.Document;

import com.liferay.portal.kernel.search.Field;

import com.liferay.portal.search.sort.FieldSort;

import com.liferay.portal.search.sort.Sort;

import lombok.extern.slf4j.Slf4j;

import ru.rvdsystems.liferay.api.fieldentity.DocumentLibraryLink;

import ru.rvdsystems.liferay.api.fieldentity.Option;

import ru.rvdsystems.liferay.api.repository.DDMFieldContainer;

import ru.rvdsystems.liferay.api.repository.LiferayDocumentRepository;

import ru.rvdsystems.liferay.api.repository.exception.LiferayFieldMappingException;

import ru.rvdsystems.liferay.api.repository.exception.LiferayIncompatibleMediatorException;

import ru.rvdsystems.liferay.api.repository.exception.LiferayIndexConfigurationException;

import ru.rvdsystems.liferay.api.search.SearchConstants;

import java.sql.Date;

import java.util.Arrays;

import java.util.List;

import java.util.Map;

import java.util.Objects;

import java.util.stream.Collectors;

@Slf4j

class IndexerUtil {

public static void populateIndexedFields(DDMFieldContainer container, Document document, AnnotatedDDMFieldMediator mediator) throws LiferayIncompatibleMediatorException, LiferayFieldMappingException {

for(Map.Entry<String, ClassFieldInfo> entry : mediator.getIndexeds().entrySet()){

String reference = entry.getKey();

ClassFieldInfo fieldInfo = entry.getValue();

Object value = mediator.getValue(container, fieldInfo);

if(value==null){

log.trace("Field {} has a null value, bypassing",reference);

continue;

}

Class<?> fieldClass = fieldInfo.getPropertyType();

if(String.class==fieldClass){

if(fieldInfo.isKeyword()){

if(fieldInfo.isSortable()){

document.addKeywordSortable(reference, (String) value);

log.trace("Added sortable keyword '{}' with value '{}'",reference, value);

} else {

document.addKeyword(reference, (String) value);

log.trace("Added keyword '{}' with value '{}'",reference, value);

}

} else {

if(fieldInfo.isSortable()) {

document.addTextSortable(reference, (String) value);

log.trace("Added sortable text '{}' with value '{}'", reference, value);

} else {

document.addText(reference, (String) value);

log.trace("Added text '{}' with value '{}'", reference, value);

}

}

} else if( fieldInfo.isListOf(String.class) ) {

String[] values = ((List<String>)value).toArray(new String[0]);

if(fieldInfo.isKeyword()){

if(fieldInfo.isSortable()){

document.addKeywordSortable(reference, values);

log.trace("Added sortable keyword[] '{}' with value '{}'",reference, value);

} else {

document.addKeyword(reference, values);

log.trace("Added keyword[] '{}' with value '{}'",reference, value);

}

} else {

if(fieldInfo.isSortable()) {

document.addTextSortable(reference, values);

log.trace("Added sortable text[] '{}' with value '{}'", reference, value);

} else {

document.addText(reference, values);

log.trace("Added text[] '{}' with value '{}'", reference, value);

}

}

} else if(Option.class.isAssignableFrom(fieldClass)){

String optValue = ((Option)value).getTitle();

if(fieldInfo.isSortable()){

document.addKeywordSortable(reference, optValue);

log.trace("Added sortable option keyword '{}' with value '{}'",reference, optValue);

} else {

document.addKeyword(reference, optValue);

log.trace("Added option keyword '{}' with value '{}'",reference, optValue);

}

} else if(fieldInfo.isListOf(Option.class)) {

List<String> optValue = ((List<Option>)value).stream().map(Option::getTitle).collect(Collectors.toList());

if(fieldInfo.isSortable()){

document.addKeywordSortable(reference, optValue.toArray(new String[0]));

log.trace("Added sortable option[] keyword[] '{}' with value '{}'",reference, optValue);

} else {

document.addKeyword(reference, optValue.toArray(new String[0]));

log.trace("Added option[] keyword[] '{}' with value '{}'",reference, optValue);

}

}else if(Integer.class==fieldClass){

Integer i = (Integer) value;

if(fieldInfo.isSortable()){

document.addNumberSortable(reference, i);

log.trace("Added sortable number '{}' with value '{}'",reference, value);

} else {

document.addNumber(reference, i);

log.trace("Added number '{}' with value '{}'",reference, value);

}

}else if(fieldInfo.isListOf(Integer.class)) {

List<Integer> i = (List<Integer>) value;

if(fieldInfo.isSortable()){

document.addNumberSortable(reference, i.toArray(new Integer[0]));

log.trace("Added sortable number[] '{}' with value '{}'",reference, value);

} else {

document.addNumber(reference, i.toArray(new Integer[0]));

log.trace("Added number[] '{}' with value '{}'",reference, value);

}

} else if(Double.class==fieldClass){

Double d = (Double) value;

if(fieldInfo.isSortable()){

document.addNumberSortable(reference, d);

log.trace("Added sortable number '{}' with value '{}'",reference, value);

} else {

document.addNumber(reference, d);

log.trace("Added number '{}' with value '{}'",reference, value);

}

} else if(fieldInfo.isListOf(Double.class)) {

List<Double> d = (List<Double>) value;

if(fieldInfo.isSortable()){

document.addNumberSortable(reference, d.toArray(new Double[0]));

log.trace("Added sortable number[] '{}' with value '{}'",reference, value);

} else {

document.addNumber(reference, d.toArray(new Double[0]));

log.trace("Added number[] '{}' with value '{}'",reference, value);

}

} else if(Date.class==fieldClass){

Date date = (Date) value;

if(fieldInfo.isSortable()){

document.addDateSortable(reference, date);

log.trace("Added sortable date '{}' with value '{}'",reference, value);

} else {

document.addDate(reference, date);

log.trace("Added date '{}' with value '{}'",reference, value);

}

} else if(fieldInfo.isListOf(Date.class)) {

List<Date> date = (List<Date>) value;

if(fieldInfo.isSortable()){

document.addDateSortable(reference, date.toArray(new Date[0]));

log.trace("Added sortable date[] '{}' with value '{}'",reference, value);

} else {

document.addDate(reference, date.toArray(new Date[0]));

log.trace("Added date[] '{}' with value '{}'",reference, value);

}

} else if(DocumentLibraryLink.class.isAssignableFrom(fieldClass)){

DocumentLibraryLink link = (DocumentLibraryLink)value;

document.addNumberSortable(reference, Long.parseLong(link.getFileEntryId()));

log.trace("Added DocumentLibraryLink as number '{}' with value '{}'",reference, link.getFileEntryId());

if(! fieldInfo.isKeyword()){

String titleRef = reference+"\_title";

if(fieldInfo.isSortable()){

document.addTextSortable(titleRef, link.getTitle());

log.trace("Added DocumentLibraryLink as sortable text '{}' with value '{}'",titleRef, link.getTitle());

} else {

document.addText(titleRef, link.getTitle());

log.trace("Added DocumentLibraryLink as text '{}' with value '{}'",titleRef, link.getTitle());

}

}

} else if(fieldInfo.isListOf(DocumentLibraryLink.class)){

List<DocumentLibraryLink> links = (List<DocumentLibraryLink>) value;

Long[] ids = links.stream().filter(Objects::nonNull).map(l -> Long.parseLong(l.getFileEntryId())).toArray(Long[]::new);

document.addNumberSortable(reference, ids);

log.trace("Added DocumentLibraryLink[] as number[] '{}' with value '{}'",reference, Arrays.asList(ids));

if(! fieldInfo.isKeyword()){

String titleRef = reference+"\_title";

List<String> titles = links.stream().filter(Objects::nonNull).map(DocumentLibraryLink::getTitle).collect(Collectors.toList());

if(fieldInfo.isSortable()){

document.addTextSortable(titleRef, titles.toArray(new String[0]));

log.trace("Added DocumentLibraryLink[] as sortable text[] '{}' with value '{}'",titleRef, titles);

} else {

document.addText(titleRef, titles.toArray(new String[0]));

log.trace("Added DocumentLibraryLink[] as text[] '{}' with value '{}'",titleRef, titles);

}

}

}

//TODO ImageLink.class, Geolocation.class PageLink.class, List<List<...>> ???

else {

log.warn("Class {}, field {}: Indexing for type of {} is not supported. Supported types: String, String[], Double, Double[] Integer, Integer[], Date, Date[], Option, Option[], DocumentLibraryLink, DocumentLibraryLink[]", container.getClass(), fieldInfo.getName(), fieldClass);

}

}

}

public static Sort[] prepareSorts(Sort[] sorts, AnnotatedDDMFieldMediator mediator) throws LiferayIndexConfigurationException {

if(sorts==null || sorts.length==0){

return sorts;

}

final Sort[] result = new Sort[sorts.length];

for(int i=0; i<sorts.length; i++){

final Sort sort = sorts[i];

String sortableFieldName = null, sourcefieldName = null;

if(sort instanceof FieldSort){

FieldSort fieldSort = (FieldSort) sort;

final ClassFieldInfo fieldInfo = mediator.getIndexeds().get(fieldSort.getField());

if( fieldInfo !=null ){

if(! fieldInfo.isSortable()){

throw new LiferayIndexConfigurationException(String.format(

"Class %s Field %s is not sortable, cannot prepare sort",

mediator.getFieldContainerClass().getName(),

fieldSort.getField()

));

}

sourcefieldName = fieldSort.getField();

Class<?> fieldClass = fieldInfo.getPropertyType();

if(String.class==fieldClass || fieldInfo.isListOf(String.class)

|| Option.class.isAssignableFrom(fieldClass) || fieldInfo.isListOf(Option.class)) {

sortableFieldName = stringSortable(fieldSort.getField());

} else if(Integer.class==fieldClass || fieldInfo.isListOf(Integer.class)

|| Double.class==fieldClass || fieldInfo.isListOf(Double.class)

||Date.class==fieldClass || fieldInfo.isListOf(Date.class)) {

sortableFieldName = numberSortable(fieldSort.getField());

}

//TODO ImageLink.class, Geolocation.class PageLink.class, List<List<...>> ???

else {

log.warn("Class {}, field {}: Indexing for type of {} is not supported. Supported types: String, String[], Double, Double[] Integer, Integer[], Date, Date[], Option, Option[], DocumentLibraryLink, DocumentLibraryLink[]", mediator.getFieldContainerClass(), fieldInfo.getName(), fieldClass);

}

} else {

if(! fieldSort.getField().endsWith(Field.SORTABLE\_FIELD\_SUFFIX)){

sortableFieldName = defaultSortable(fieldSort.getField());

}

}

}

result[i] = sortableFieldName!=null ?

LiferayDocumentRepository.getSortBuilderFactory().getSortBuilder()

.field(sortableFieldName).sortOrder(sort.getSortOrder()).build()

: sort;

if(sortableFieldName!=null){

log.trace("prepareSorts: Class {} sorted field {} changed to {}", mediator.getFieldContainerClass(), sourcefieldName, sortableFieldName);

}

}

return result;

}

public static String stringSortable(String fieldName){

return getSortableField(fieldName, "String");

}

public static String defaultSortable(String fieldName){

return getSortableField(fieldName, null);

}

public static String numberSortable(String fieldName){

return getSortableField(fieldName, "Number");

}

public static String dateSortable(String fieldName){

return numberSortable(fieldName);

}

private static String getSortableField(String fieldName, String type){

switch (fieldName){

case Field.TITLE:

case Field.PUBLISH\_DATE:

case Field.PRIORITY:

case Field.CREATE\_DATE:

case Field.MODIFIED\_DATE:

case Field.ASSET\_ENTRY\_ID:

case Field.EXPIRATION\_DATE:

case SearchConstants.FIELD\_SIZE:

case SearchConstants.FIELD\_VIEW\_COUNT:

return fieldName+"\_"+Field.SORTABLE\_FIELD\_SUFFIX;

default:

return fieldName+(type!=null ? "\_"+type: "")+"\_"+ Field.SORTABLE\_FIELD\_SUFFIX;

}

}

}

package ru.rvdsystems.liferay.impl.mediator.annotated;

import com.liferay.document.library.kernel.exception.NoSuchFileEntryException;

import com.liferay.document.library.kernel.model.DLFileEntry;

import com.liferay.document.library.kernel.service.DLFileEntryLocalServiceUtil;

import com.liferay.portal.kernel.exception.PortalException;

import com.liferay.portal.kernel.upload.FileItem;

import lombok.extern.slf4j.Slf4j;

import ru.rvdsystems.liferay.api.Util;

import ru.rvdsystems.liferay.api.fieldentity.DocumentLibraryLink;

import ru.rvdsystems.liferay.api.fieldentity.ImageLink;

import ru.rvdsystems.liferay.api.fieldentity.Option;

import ru.rvdsystems.liferay.api.repository.DDMFieldContainer;

import ru.rvdsystems.liferay.api.repository.LiferayDocument;

import ru.rvdsystems.liferay.api.repository.exception.LiferayFieldMappingException;

import ru.rvdsystems.liferay.impl.mediator.upload.DelayedDocumentContainer;

import ru.rvdsystems.liferay.impl.mediator.upload.DelayedDocumentLibraryLink;

import ru.rvdsystems.liferay.impl.mediator.upload.DelayedImageLink;

import ru.rvdsystems.liferay.impl.mediator.DDMFieldContainerMediator;

import ru.rvdsystems.liferay.impl.mediator.upload.UploadFormProcessResult;

import java.io.IOException;

import java.lang.reflect.Constructor;

import java.lang.reflect.InvocationTargetException;

import java.sql.Date;

import java.time.format.DateTimeParseException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Map;

@Slf4j

class RequestParameterMapUtil {

static List<DelayedDocumentContainer> setContainerProperty(DDMFieldContainer container, Map.Entry<String,Object> entry, AnnotatedDDMFieldMediator mediator) throws PortalException {

log.info("setting container {} property for mapper entry '{}'={}", container.getClass().getName(), entry.getKey(),entry.getValue());

ClassFieldInfo fieldInfo = mediator.getByName().get(entry.getKey());

log.trace("found fieldInfo for key {}: {}",entry.getKey(), fieldInfo);

Object mappedValue = entry.getValue();

if(mappedValue == null || fieldInfo==null){

return Collections.emptyList();

}

if(fieldInfo.getWriteMethod()==null){

throw new LiferayFieldMappingException(String.format("Cannot map submitted parameter %s because %s#%s property does not have setter method.", entry.getKey(), fieldInfo.getDeclaringClass().getName(), fieldInfo.getName() ));

}

Object castValue;

List<DelayedDocumentContainer> delayedDocumentContainers = new ArrayList<>();

try {

if (mappedValue instanceof List) {

//expect list value

if (fieldInfo.isList()) {

try {

castValue = constructList((List<?>) mappedValue, fieldInfo.getListParameterClass(), fieldInfo, delayedDocumentContainers);

} catch (ClassCastException e) {

throw new LiferayFieldMappingException(String.format("Unexpectedly cannot cast %s to List of values", mappedValue));

}

} else if (fieldInfo.isListOfList()) {

//expect list of list

try {

castValue = constructListOfList((List<List<?>>) mappedValue, fieldInfo.getListSublistParameterClass(), fieldInfo, delayedDocumentContainers);

} catch (ClassCastException e) {

throw new LiferayFieldMappingException(String.format("Unexpectedly cannot cast %s to List of subList values", mappedValue));

}

} else {

throw new LiferayFieldMappingException(String.format("Unexpected list value %s", mappedValue));

}

} else {

castValue = castValue(mappedValue, fieldInfo.getPropertyType(), fieldInfo, delayedDocumentContainers);

}

}catch (LiferayFieldMappingException | IOException e){

throw new LiferayFieldMappingException(String.format("%s#%s cannot map value from request : %s", mediator.getFieldContainerClass().getName(), fieldInfo.getName(), e.getMessage()));

}

mediator.setValue(container,fieldInfo,castValue);

return delayedDocumentContainers;

}

private static List<Object> constructList(final List<?> mappedValueList, final Class<?> ListClass, final ClassFieldInfo fieldInfo, final List<DelayedDocumentContainer> delayedDocumentContainers) throws PortalException, IOException {

List<Object> result = new ArrayList<>();

for( Object mappedValue : mappedValueList){

Object value = castValue(mappedValue,ListClass, fieldInfo, delayedDocumentContainers);

if(value !=null) {

result.add(value);

}

}

return result;

}

private static List<List<Object>> constructListOfList(final List<List<?>> mappedListOfLists, final Class<?> subListCLass, final ClassFieldInfo fieldInfo, final List<DelayedDocumentContainer> delayedDocumentContainers) throws PortalException, IOException {

List<List<Object>> result = new ArrayList<>();

for(List<?> subList : mappedListOfLists){

result.add(constructList(subList,subListCLass, fieldInfo, delayedDocumentContainers));

}

return result;

}

private static Object castValue(final Object mappedValue, final Class<?> fieldClass, final ClassFieldInfo fieldInfo, final List<DelayedDocumentContainer> delayedDocumentContainers) throws PortalException, IOException {

if(mappedValue instanceof Map){

if(DDMFieldContainer.class.isAssignableFrom(fieldClass)) {

//Map should be mapped to nested DDMFieldContainer field

Class<? extends DDMFieldContainer> fieldSetClass;

try {

fieldSetClass = (Class<? extends DDMFieldContainer>) fieldClass;

log.trace("field {}: nested container class: {}", fieldInfo.getName(), fieldSetClass.getName());

} catch (ClassCastException e) {

throw new LiferayFieldMappingException(String.format("Cannot convert map of values to %s", fieldClass.getName()));

}

final DDMFieldContainerMediator fieldSetMediator = AnnotatedDDMFieldMediator.getFieldContainerMediator(fieldSetClass);

UploadFormProcessResult fieldSetResult = fieldSetMediator.fromFormParameterMap(null, (Map<String, Object>) mappedValue);

log.trace("Field {} mapping result: {}", fieldInfo.getName(), fieldSetResult.getContainer());

delayedDocumentContainers.addAll(fieldSetResult.getDelayed());

return fieldSetResult.getContainer();

} else if(DocumentLibraryLink.class == fieldClass){

if(fieldInfo.getLinkedTypes() == null || fieldInfo.getLinkedTypes().length > 1){

throw new LiferayFieldMappingException(String.format("Linked document create rejected: field %s",

fieldInfo.getLinkedTypes()==null || fieldInfo.getLinkedTypes().length==0 ? "has no associated linked type" : "has multiple associated linked types")

);

}

if( !fieldInfo.isAllowUpload() ){

throw new LiferayFieldMappingException("Linked document rejected: Attempt to create linked document to field that does not allow uploads");

}

final DDMFieldContainerMediator mediator = DDMFieldContainerMediator.getInstance(fieldInfo.getLinkedTypes()[0]);

UploadFormProcessResult subResult = mediator.fromFormParameterMap(null, (Map<String, Object>) mappedValue);

log.info("subresult isDelete {}, container: {}", subResult.isDelete(), subResult.getContainer());

DelayedDocumentLibraryLink delayedDocumentLibraryLink = new DelayedDocumentLibraryLink((LiferayDocument) subResult.getContainer(), subResult.isDelete());

delayedDocumentContainers.add(delayedDocumentLibraryLink);

delayedDocumentContainers.addAll(subResult.getDelayed());

return !subResult.isDelete() ? delayedDocumentLibraryLink : null;

} else {

throw new LiferayFieldMappingException("Map of values may be mapped only to nested container or linked document(s)");

}

} else if(mappedValue instanceof FileItem) {

// File Upload

if(! fieldInfo.isAllowUpload()){

log.info("FIELD INFO: {}", fieldInfo);

throw new LiferayFieldMappingException("File upload rejected: Attempt to upload file to field that does not allow uploads");

}

if(fieldInfo.getPropertyType()==DocumentLibraryLink.class || fieldInfo.getPropertyType()==ImageLink.class){

if(fieldInfo.getLinkedTypes().length!=1){

throw new LiferayFieldMappingException(String.format("File upload rejected: field %s",

fieldInfo.getLinkedTypes().length==0 ? "has no associated linked type" : "has multiple associated linked types")

);

}

FileItem fileItem = (FileItem) mappedValue;

if(fieldInfo.getPropertyType()==ImageLink.class){

if( ! fileItem.getContentType().startsWith("image/")){

throw new LiferayFieldMappingException(String.format("File upload rejected: unsupported content type %s for image", fileItem.getContentType()));

}

}

Class<? extends LiferayDocument> linkedType = fieldInfo.getLinkedTypes()[0];

LiferayDocument linkedDoc = createLinkedDocument(linkedType, fileItem, fieldInfo.getUploadTitle() );

DelayedDocumentContainer delayedLink;

if(fieldInfo.getPropertyType()==ImageLink.class){

delayedLink = new DelayedImageLink(linkedDoc, false);

} else {

delayedLink = new DelayedDocumentLibraryLink(linkedDoc, false);

}

delayedDocumentContainers.add(delayedLink);

return delayedLink;

} else {

throw new LiferayFieldMappingException(String.format("File upload rejected: field type %s is not supported for uploads", fieldInfo.getPropertyType().getName()));

}

} else {

//scalar property

String stringMappedValue;

try {

stringMappedValue = (String) mappedValue;

} catch (ClassCastException e){

throw new LiferayFieldMappingException(String.format("Expected string map entry type instead of '%s' (%s)", mappedValue, mappedValue.getClass()));

}

return castScalarValue(stringMappedValue,fieldClass);

}

}

private static Object castScalarValue(String mappedValue, Class<?> destClass) throws PortalException, IOException {

//scalar property: may be String, Integer, Double, java.sql.Date, or Option

if( Util.isEmpty(mappedValue) ){

return null;

}

if (destClass == String.class) {

return mappedValue;

} else if (destClass == Integer.class) {

try {

return Integer.parseInt(mappedValue);

}catch (NumberFormatException e){

throw new LiferayFieldMappingException(String.format("Cannot convert value '%s' to integer", mappedValue));

}

} else if (destClass == Long.class) {

try {

return Long.parseLong(mappedValue);

}catch (NumberFormatException e){

throw new LiferayFieldMappingException(String.format("Cannot convert value '%s' to long", mappedValue));

}

} else if (destClass == Double.class) {

try {

return Double.parseDouble(mappedValue.replace(',', '.'));

} catch (NumberFormatException e) {

throw new LiferayFieldMappingException(String.format("Cannot convert value '%s' to Double", mappedValue));

}

} else if (destClass== Date.class){

try {

return new Date(Util.parseDate(mappedValue).getTime());

} catch (DateTimeParseException e){

throw new LiferayFieldMappingException(String.format("Cannot convert value '%s' to Date", mappedValue));

}

} else if (Option.class.isAssignableFrom(destClass)) {

return Option.fromLabel(mappedValue, (Class<? extends Option>) destClass);

} else if( DocumentLibraryLink.class==destClass ){

//Value must be a long

return toDocumentLink(mappedValue);

} else if (ImageLink.class==destClass) {

return toImageLink(mappedValue);

} else {

throw new IllegalStateException(String.format("Unexpected scalar type: %s (Expected: String, Integer, Long, Double, Date, Option)", destClass.getName()));

}

}

private static DLFileEntry getFileEntry(String mappedValue) throws PortalException {

try {

long fileEntryId = Long.parseLong(mappedValue);

try {

return DLFileEntryLocalServiceUtil.getFileEntry(fileEntryId);

}catch (NoSuchFileEntryException e){

throw new LiferayFieldMappingException(String.format("Cannot interpret '%s' value as a file entry id: linked file entry does not exist", mappedValue));

}

} catch (NumberFormatException e){

throw new LiferayFieldMappingException(String.format("Cannot interpret '%s' value as a file entry id: long value expected", mappedValue));

}

}

private static DocumentLibraryLink toDocumentLink(String mappedValue) throws PortalException {

return DocumentLibraryLink.from(getFileEntry(mappedValue));

}

private static ImageLink toImageLink(String mappedValue) throws PortalException, IOException {

return ImageLink.from(getFileEntry(mappedValue));

}

private static LiferayDocument createLinkedDocument(Class<? extends LiferayDocument> clazz, FileItem fileItem, String docTitle) throws IOException{

try {

log.info("Creating linked document {} with title '{}' from FileItem {} (size: {}, content-type: {})", clazz.getName(), docTitle, fileItem.getFileName(), fileItem.getSize(), fileItem.getContentType());

Constructor<? extends LiferayDocument> constr = clazz.getConstructor(null);

LiferayDocument doc = constr.newInstance(null);

doc.setTitle( Util.isEmpty(docTitle) ? fileItem.getFileName() : docTitle );

doc.setContentToSave(fileItem.getInputStream(),fileItem.getContentType(),fileItem.getFileName());

log.info("Linked document created");

return doc;

} catch (NoSuchMethodException | InstantiationException | IllegalAccessException | InvocationTargetException e) {

throw new IllegalStateException(String.format("Unexpectedly cannot instantiate a new instance of %s", clazz));

}

}

}