**BỘ TÀI NGUYÊN VÀ MÔI TRƯỜNG**

**TRƯỜNG ĐH TÀI NGUYÊN VÀ MÔI TRƯỜNG TP. HCM**

**KHOA HỆ THỐNG THÔNG TIN VÀ VIỄN THÁM**



**BÁO CÁO ĐỒ ÁN MÔN LẬP TRÌNH HĐT**

**QUẢN LÝ KHO DVD PHIM**

Thành viên của nhóm:

1. Nguyễn Khánh Nam

MSSV: 1050080065

1. Đào Ngọc Hòa

MSSV: 1050080050

1. Trầm Xuân Trọng

MSSV: 1050080081

Lớp: 10ĐH-CNTT2

# **Mục Lục**

1. **Giới thiệu**
2. **Phân tích**
3. **Thiết kế**
4. **Hiện thực hóa**
5. **Kết luận**
6. **Một vài snippet code**

# **Giới thiệu**

## **Giới thiệu chung**

Việc tự quản lý và sắp xếp nhiều loại hàng hóa khác nhau rất tốn thời gian và kém hiệu quả. Lịch sử đã giới thiệu việc sử dụng sổ cái để đảm bảo tính nhất quán cho dữ liệu buôn bán, nhưng việc tổ chức và sàng lọc vẫn phải thông qua bàn tay con người. Với dự án phần mềm quản lý kho DVD, việc ghi chép các dữ liệu này sẽ không những được đảm bảo hiệu quả và an toàn mà các công đoạn tính toán, sắp xếp và tổ chức sẽ được diễn ra một cách tự động.

## **Phạm vi dự án**

Phần mềm được tạo ra hỗ trợ giao diện và hệ thống xác thực người dùng. Nếu thành công, người dùng có thể…

* Xem/sắp xếp/lọc các băng DVD hiện hành
* Tạo giao dịch: nhập/xuất kho từ các đối tác
* Xem/Sắp xếp các giao dịch đã diễn ra trong quá khứ
* Thêm/Xóa/Sửa thông tin băng DVD
* Thêm/Xóa/Sửa thông tin đối tác

…tùy thuộc vào mức độ quyền hạn người dùng

## **Các công nghệ được sử dụng**

Danh Sách:

* Language: Java
* SDK: OpenJDK 19 (19.0.2)
* JVM: OpenJDK 64-Bit Server (19.0.2)
* IDE: IntelliJ Idea (2022.2.2)
* GUI Library: Java Swing (OpenJDK 19)
* Build system: IntelliJ
* Backend: Firebase
* Authentication provider: Firebase Authentication
* Database provider: Cloud Firestore
* Management tools: Firebase Admin

Java là một ngôn ngữ lập trình bậc cao, phân lớp, hướng đối tượng do Sun Microsystem tạo ra. Java đã trở thành ngôn ngữ lập trình nổi tiếng nhất hiện nay(1) nhờ nguyên lý *Viết một lần, chạy ở mọi nơi* giúp cắt bớt thời gian phát triển phần mềm. Java có hai nhà cung cấp SDK lớn nhất: Oracle và OpenJDK. Do tính dễ truy cập của phần mềm mã nguồn mở(2), dự án này sẽ sử dụng OpenJDK phiên bản 19.

Để giúp có được trải nghiệm sử dụng tốt nhất, ứng dụng quản lý cần phải có giao diện người dùng. Hiện nay có hai cách nổi tiếng nhất để tạo giao diện cho một ứng dụng: thông qua trình duyệt Web hoặc hỗ trợ native. Mặc dù khó tin nhưng việc sử dụng trình duyệt Web không giới chỉ hạn cho các trang mạng mà còn cho cả những ứng dụng desktop, một ví dụ điển hình có thể kể đến là Jupyter Notebook, một ứng dụng lập trình Python. Tuy nhiên, do phụ thuộc vào trình duyệt nên vẻ bề ngoài của ứng dụng có thể thay đổi đáng ngờ tùy thuộc vào trình duyệt nào đang mở ứng dụng, và nếu máy tính người dùng không có trình duyệt thì sẽ hoàn toàn không tương tác được với ứng dụng. Tuy nhiên Java với khẩu hiệu *Viết một lần, chạy ở mọi nơi* có thể giải quyết được vấn đề này bằng việc tạo giao diện người dùng trực tiếp trên máy tính (native) mà không thông qua một bên thứ ba nào. Có nhiều thư viện hỗ trợ tham vọng này nhưng dự án quản lý kho DVD sẽ dùng Java Swing vì tính tích hợp của nó ở trong IDE…

Môi trường phát triển tích hợp (IDE) là một ứng dụng/chuỗi công cụ giúp lập trình viên tạo ra phần mềm. Xét về Java nói riêng và chuỗi ngôn ngữ sử dụng JVM nói chung thì IntelliJ Idea là IDE nổi bật nhất cho việc lập trình(3). Idea có nhiều tính năng nổi bật như(4):

* Hỗ trợ Code: Tự hoàn thành (code completion), tái cấu trúc (refactoring), gỡ lỗi (debugging)
* Tích hợp nhiều công cụ: trình duyệt web, hệ thống build Gradle, Maven, hệ thống kiểm soát phiên bản Git, SVN, …, cơ sở dữ liệu như Microsoft SQL Server, Oracle, PostgreSQL, SQLite, MySQL
* Hệ sinh thái Plugin
* Hỗ trợ nhiều ngôn ngữ/môi trường

Firebase là một nền tảng điện toán đám mây do Google cung cấp. Firebase cung cấp một chuỗi ứng dụng đa dạng dưới hình thức IaaS (Cơ sở hạ tầng dưới dạng dịch vụ) giúp các nhà phát triển backend và quản trị hệ thống làm việc hiệu quả hơn. Một số công cụ nổi bật bao gồm:

* Authentication: Hệ thống CIAM(5) chính của chuỗi dịch vụ. Cung cấp các công cụ nhằm xác thực người dùng thông qua các nhà cung cấp truyền thống như email, số điện thoại, các nhà cung cấp thông bên thứ ba (sử dụng OAuth) như Google, Facebook, Twitter, Github. Ngoài ra còn hỗ trợ các tính năng quan trọng như quên mật khẩu, xác thực email/số điện thoại, vô hiệu hóa/gỡ tài khoản. Tất cả tính năng này đều được hỗ trợ thông qua REST API do Firebase cung cấp(6)
* Cloud Firestore: Cơ sở dữ liệu theo mô hình NoSQL được giới thiệu gần đây nhằm thay thế Realtime Database, một cơ sở dữ liệu NoSQL khác. Bằng việc tổ chức các dữ liệu theo các tập tin JSON (ở đây gọi là “document”), Firestore có kiến trúc dữ liệu vô cùng linh hoạt mà không ảnh hưởng lớn đến tốc độ, nhờ các tính năng bổ trợ như lập chỉ mục. Ngoài ra, mọi dữ liệu do Firestore lưu trữ được bảo vệ nhờ các quy luật mà người lập trình viết. Firestore sẽ bắt tay với Authentication và dựa vào quy luật để đảm bảo chỉ có những người có quyền hạn mới được đụng vào dữ liệu trong Firestore.

Do tính chặt chẽ và bảo mật của các công cụ này nên dự án sẽ dùng hai tính năng được nêu trên.

# **Phân tích**

## **Phân tích yêu cầu hệ thống**

* Xây dựng được phần mềm quản lý kho sản phẩm giúp cho người quản lý tiết kiệm thời gian và có thể quản lý được kho hàng của mình ở mọi nơi.
* Phần mềm phải có phân cấp bậc người dùng như sau:
* Nhân viên nhập xuất kho
* Nhân viên quản lý kho
* Sysadmin (Ngoài phạm vi phần mềm)

## **Các chức năng phần mềm**

* Nhân viên nhập xuất kho: cho phép xem, sắp xếp, lọc các băng DVD hiện hành. Cho phép nhập, xuất đĩa DVD hiện có với các đối tác và kiểm tra thông tin nhập xuất trong quá khứ, xem thông tin đối tác
* Nhân viên quản lý kho: có mọi quyền hạn của nhân viên nhập xuất kho cộng với việc: Thêm, xóa, sửa thông tin DVD, thêm, xóa, sửa thông tin đối tác
* Sysadmin: bên ngoài phạm vi phần mềm, Firebase còn cung cấp thêm các công cụ bổ sung thông qua Firebase Admin: Tạo người dùng (và thông tin cá nhân), đổi mật khẩu người dùng, thống kê người dùng, Tạo dữ liệu mới trong Firestore, viết quy luật, query, chỉ mục, thống kê lần truy cập, trực tiếp thay đổi dữ liệu của nó

## **Yêu cầu chức năng**

* Xác thực:
* Mỗi nhân viên đều có một tài khoản riêng
* Nhân viên nào cũng đăng nhập được, nhưng việc đăng ký chỉ được diễn ra thông qua ứng dụng admin hoặc giao diện Firebase Admin
* Sysadmin có quyền vô hiệu hóa/xóa tài khoản nhân viên
* Tổng quan kho:
* Cho phép người dùng xem tất cả các mục DVD hiện hành
* Cho phép tạo phễu lọc cho việc tổng quan
* Tạo giao dịch: Cho phép người dùng nhập/xuất DVD thông qua các đối tác khác nhau
* Tổng quan giao dịch: Cho phép người dùng xem/sắp xếp các giao dịch đã được thực hiện trong quá khứ
* Thông tin DVD: Cho phép nhân viên quản lý kho thêm/xóa/sửa thông tin DVD
* Thông tin đối tác: Cho phép nhân viên quản lý kho thêm/xóa/sửa thông tin đối tác

## **Công việc cần giải quyết**

* B1: Phân tích và mô hình hóa yêu cầu
* B2: Xây dựng API
* B3: Xây dựng cốt lõi ứng dụng
* B4: Kiểm thử phần mềm
* B5: Hoàn thiện hóa

# **Thiết kế**

## **Đường lối thiết kế**

Do ứng dụng được thiết kế để chạy trên những hệ điều hành hiện đại, có sức mạnh phần cứng tương đối cao nên nó sử dụng nhiều luồng để thực thi. Để đảm bảo tính an toàn luồng, ứng dụng đã sử dụng nhiều khối lệnh synchronized khác nhau hoặc command queue(7) nhằm hỗ trợ việc thực thi bất đối xứng.

## **Xây dựng API**

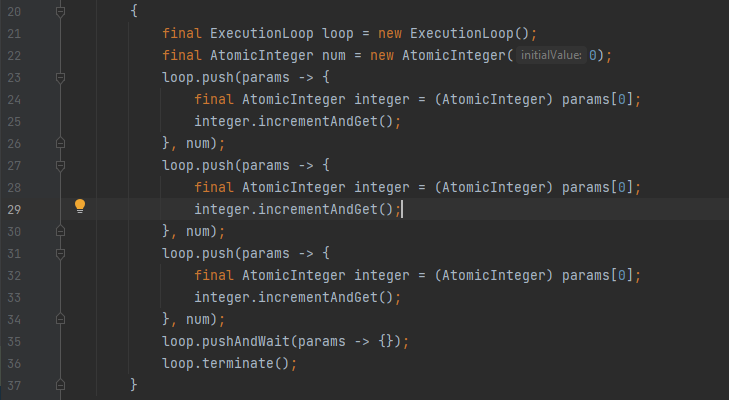
Tuy Firebase cung cấp nhiều ứng dụng, phạm vi hỗ trợ của nó tương đối thấp, với việc hỗ trợ nền tảng NodeJS là chủ yếu, và những nền tảng khác như Unity, Flutter, C++ chỉ là mục tiêu phụ. Và dù có hỗ trợ cho Java thì API của Firebase sẽ rất là “nặng” và kém tối ưu do dự án này chỉ sử dũng 2 trên hơn 15 tính năng chính đến từ Firebase. Do những lý do trên nên chúng tôi quyết định tạo tự một API mới để hỗ trợ cho dự án này, có tên là Javabase.

Để đảm bảo tính phân luồng của ứng dụng, Javabase có mang kèm hai hệ thống command queue: ExecutionLoop và AsyncEngine

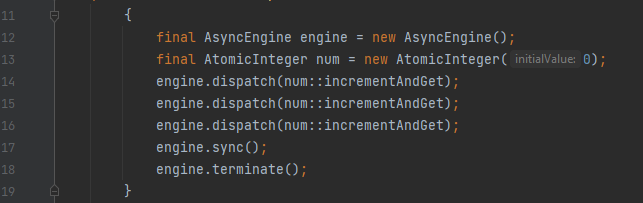
Hai command queue này có cách hoạt động ngầm giống nhau, nhưng ExecutionLoop được xây dựng dựa trên CommandQueueMT (viết bằng C++) từ dự án mã nguồn mở Godot(8), trong khi AsyncEngine dược thiết kế để nâng cao trải nghiệm lập trình bằng command queue trong Java và giảm bớt lỗi trong quá trình chạy.

AsyncEngine được tạo ra nhằm che lấp các lỗ hổng trong việc chuyển code từ C++ qua Java, đồng thời áp dụng các tính năng đặc trưng của Java như hàm/lớp ẩn danh, đồng thời nâng cao khả năng chịu lỗi của chương trình thông qua việc loại bỏ phần tử tham số trung gian.

ExecutionLoop:



AsyncEngine:



Để lưu trữ các tiến trình cần thực thi, ExecutionLoop sử dụng ReferenceList, một loại thùng chứa cho phép lưu trữ dữ liệu bằng địa chỉ. AsyncEngine cũng sử dụng một loại thùng chứa riêng có hỗ trợ FIFO để làm giảm độ phức tạp thuật toán.

Hai module FirebaseAuth và Firestore đều sử dụng AsyncEngine cho mục đích thực thi bất đồng bộ của chương trình, riêng Firestore có hai chế độ: “nóng” (sử dụng vòng lặp chính của AsyncEngine) và “lạnh” (tự tạo luồng mới sử dụng AsyncEngine). Do khác với module xác thực, lớp Firestore không có thuộc tính nào nhạy cảm với vấn đề an toàn luồng nên về lý thuyết, việc tạo luồng mới sẽ gia tăng tốc độ truy xuất cơ sở dữ liệu.

Cả hai module chính đều dùng hệ thống sự kiện nhằm năng cao tính bất đồng bộ và khiến thư viện trở nên quen thuộc hơn, do sự kiện được dùng rất nhiều trong Java.

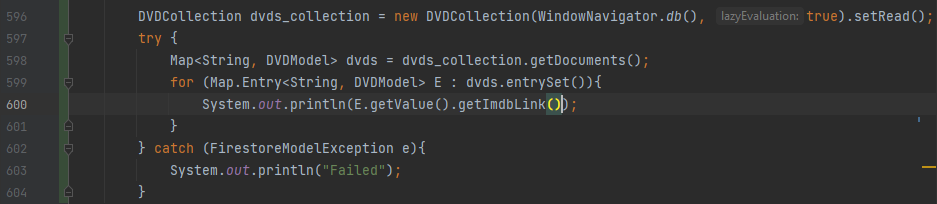


Tuy nhiên việc có API cho cơ sở dữ liệu và xác thực là chưa đủ, bởi dữ liệu lấy được từ Firestore được định dạng kiểu JSON, khiến cho việc đọc và viết dữ liệu rất khó khăn. Firestore có hỗ trợ tự động chuyển dữ liệu từ file JSON thành kiểu Map<String, Object>, nhưng việc đó là chưa đủ. Bởi vậy nên chúng tôi quyết định tạo thêm một hệ thống lớp mới giúp tự động bắt lỗi và chuyển dữ liệu JSON thành đối tượng trong Java và ngược lại.

Sử dụng phương thức cũ:



Sử dụng phương thức mới:

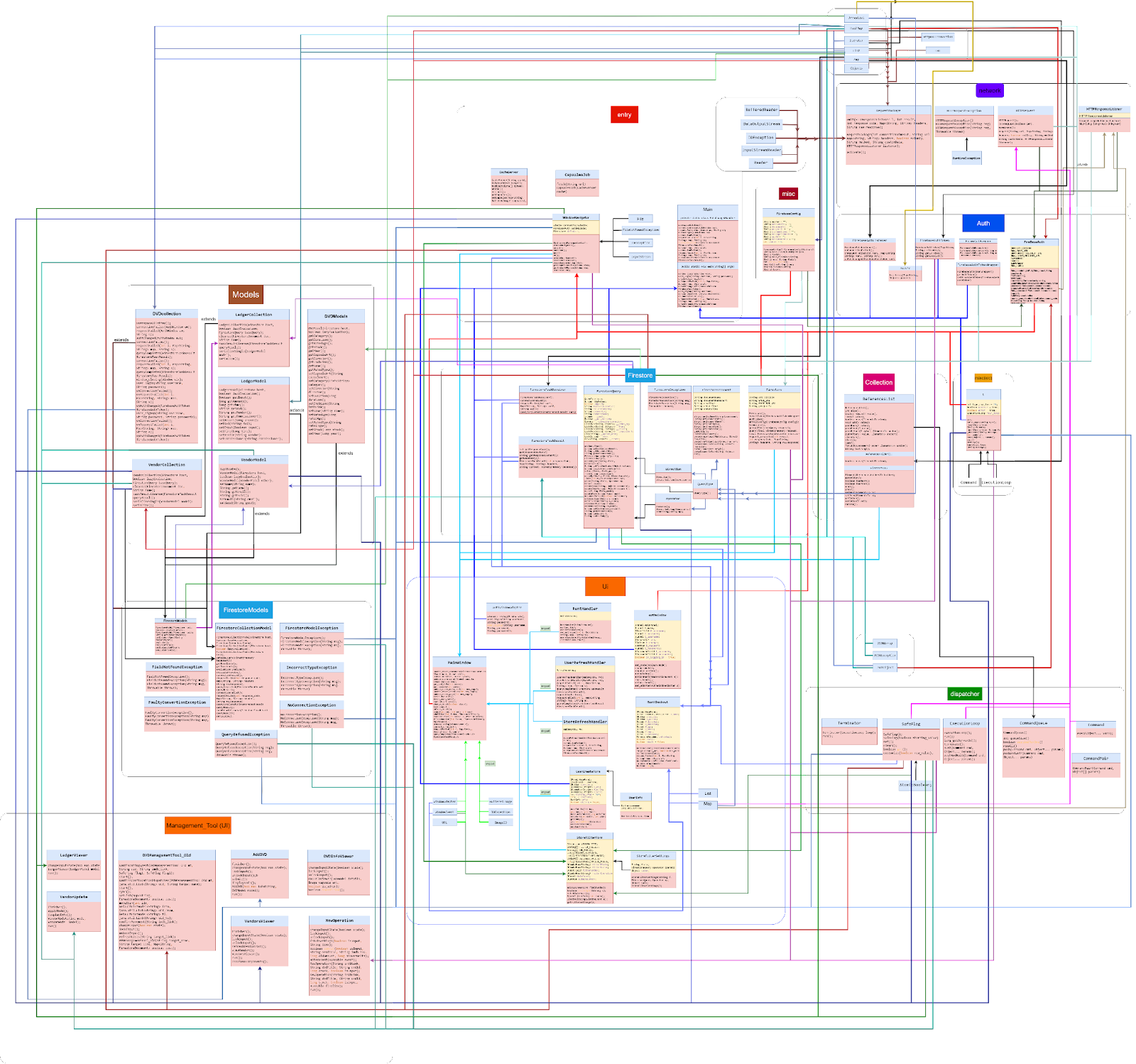


## **Xây dựng cốt lõi chương trình**

Với việc các công đoạn tốn thời gian nhất được lo liệu bởi Javabase, chúng tôi chỉ cần tạo Model, giao diện, Query cho chương trình chính.

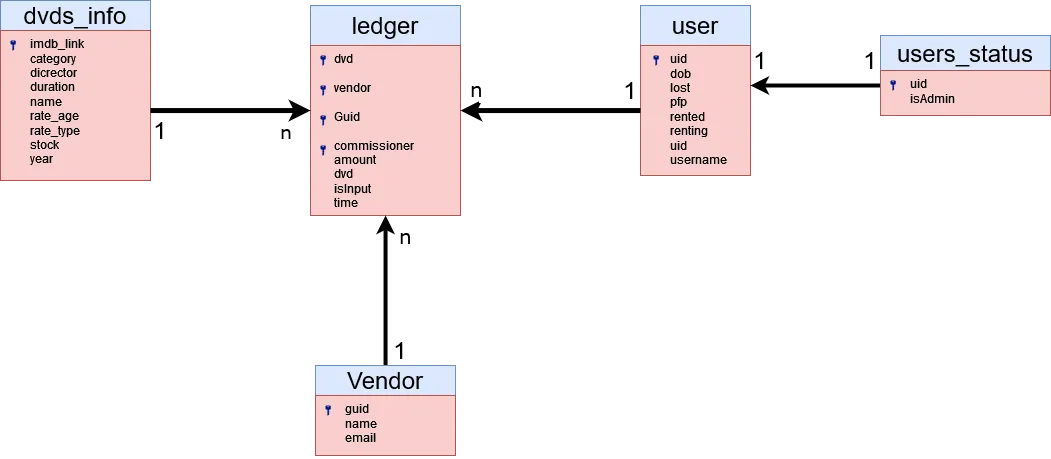
Như đã nêu trên, chương trình này sử dụng nhiều command queue cho mục đích thực thi bất đồng bộ, kể cả ở thành phần giao diện. Việc sử dụng command queue ở đây không những tăng hiệu năng và giảm xung đột tài nguyên, mà do cách hoạt động đơn luồng của Java Swing, việc này còn giúp cho chương trình không bị tắt nghẽn khi trong quá trình xử lý sự kiện của giao diện.

Mô hình lớp:



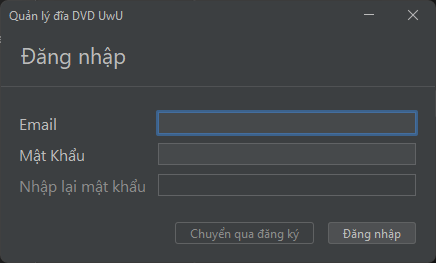
## **Xây dựng dữ liệu**

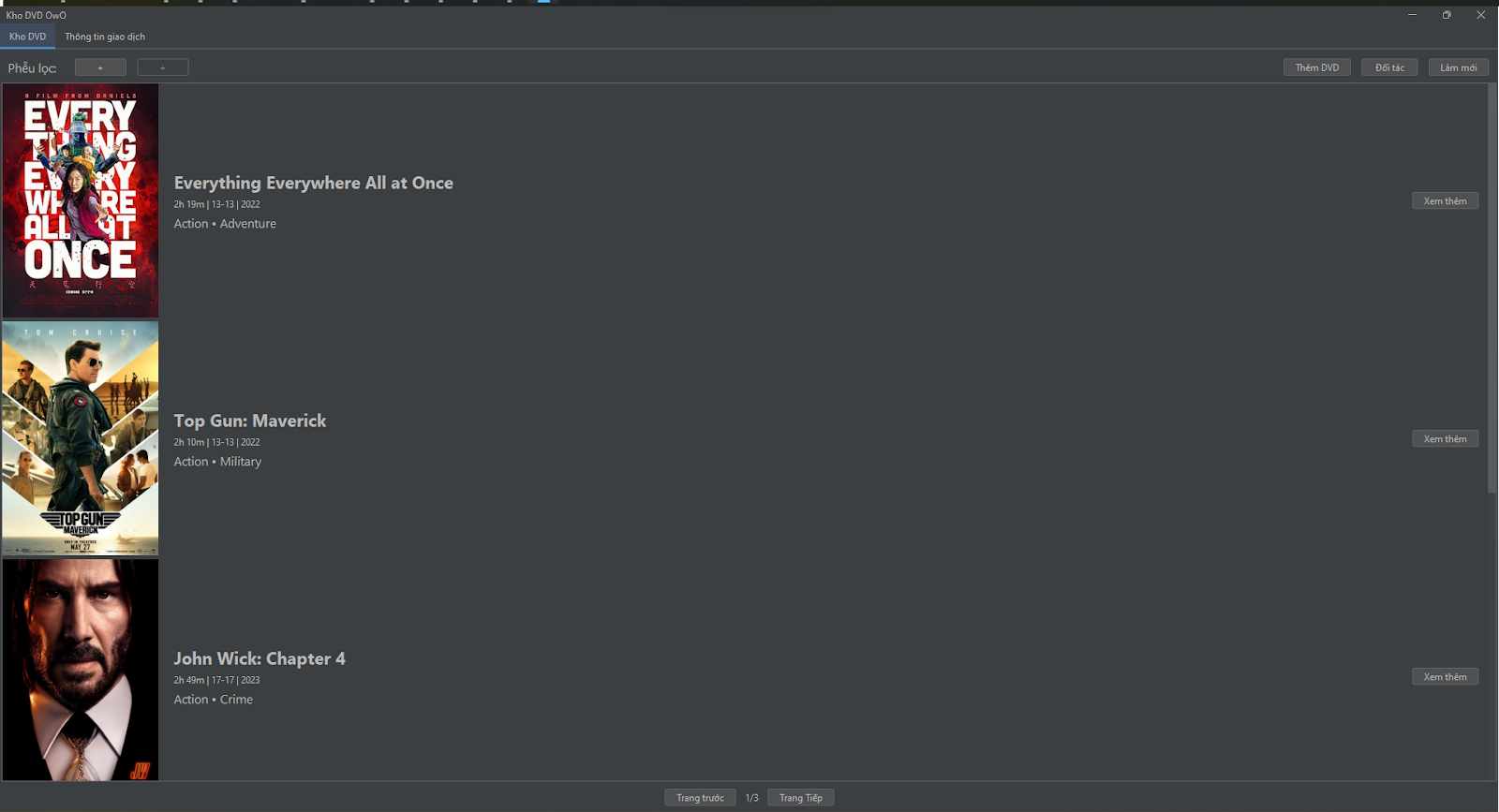
Tuy Firestore là cơ sở dữ liệu kiểu NoSQL, việc quy định ngầm cấu trúc dữ liệu sẽ khiến chương trình hoạt động tốt hơn

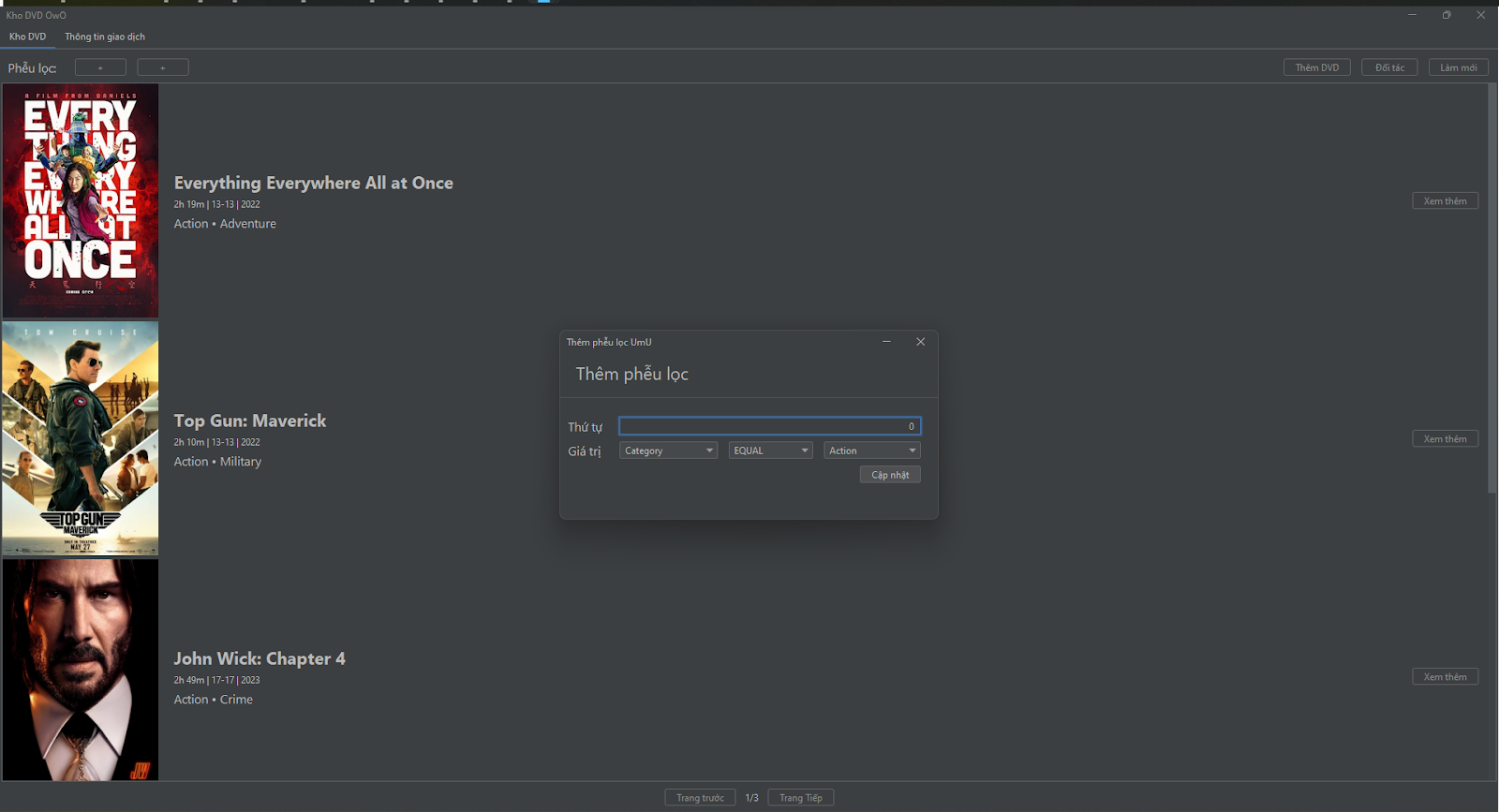


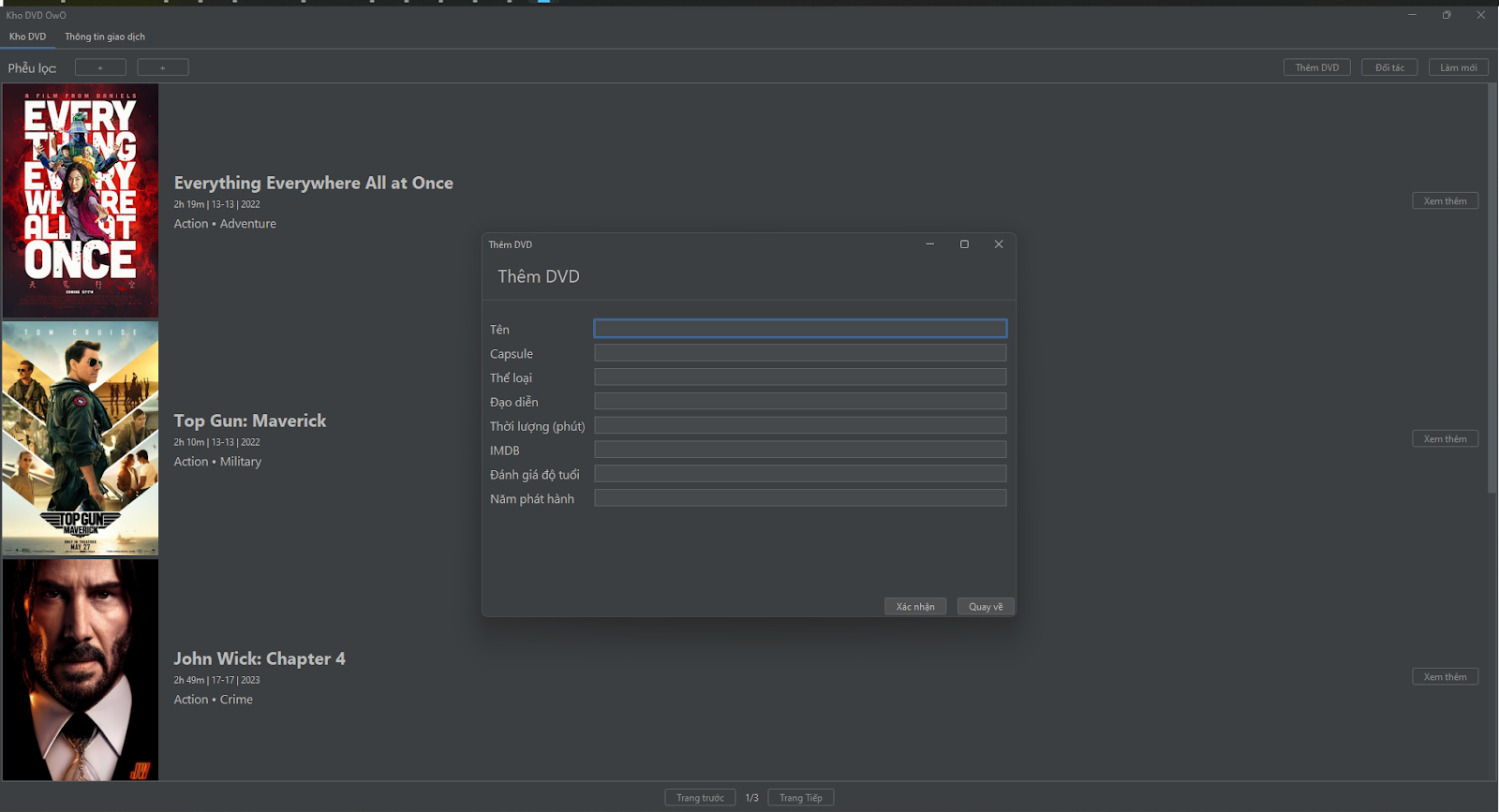
# **Hiện thực hóa**

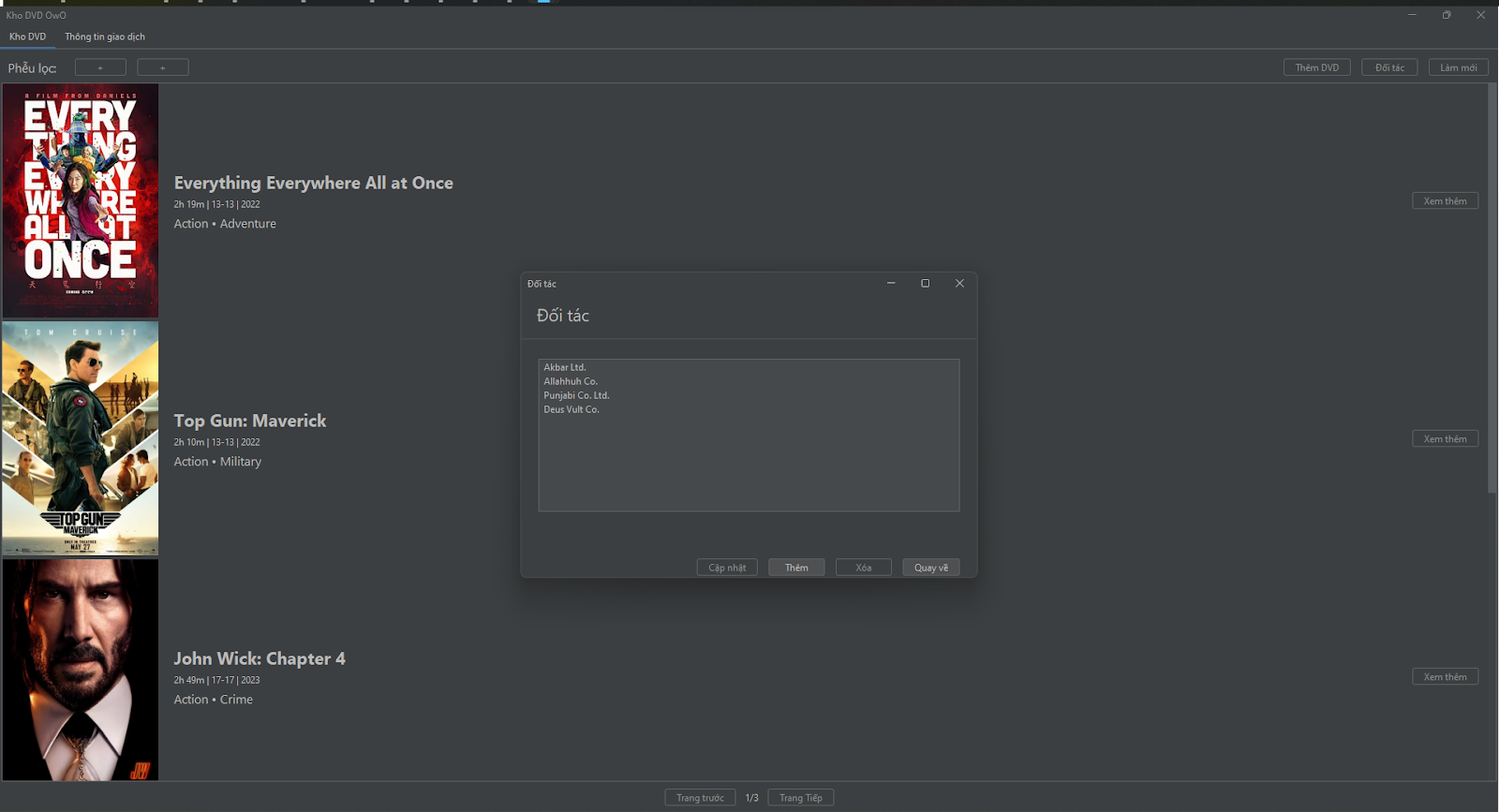
Chương trình được lưu tại: <https://github.com/NguyenKhanhNam1050080065/DVDPro>

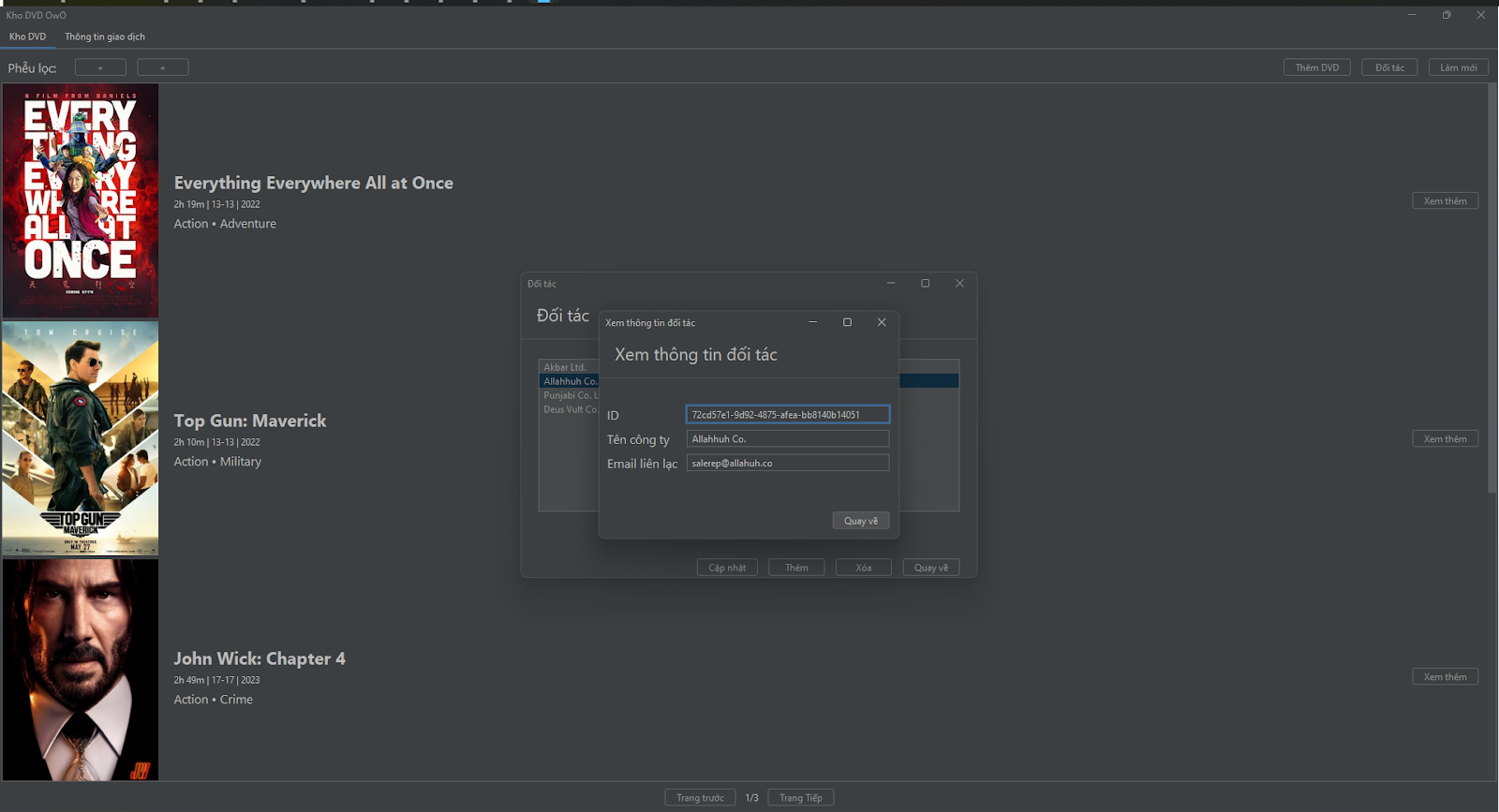


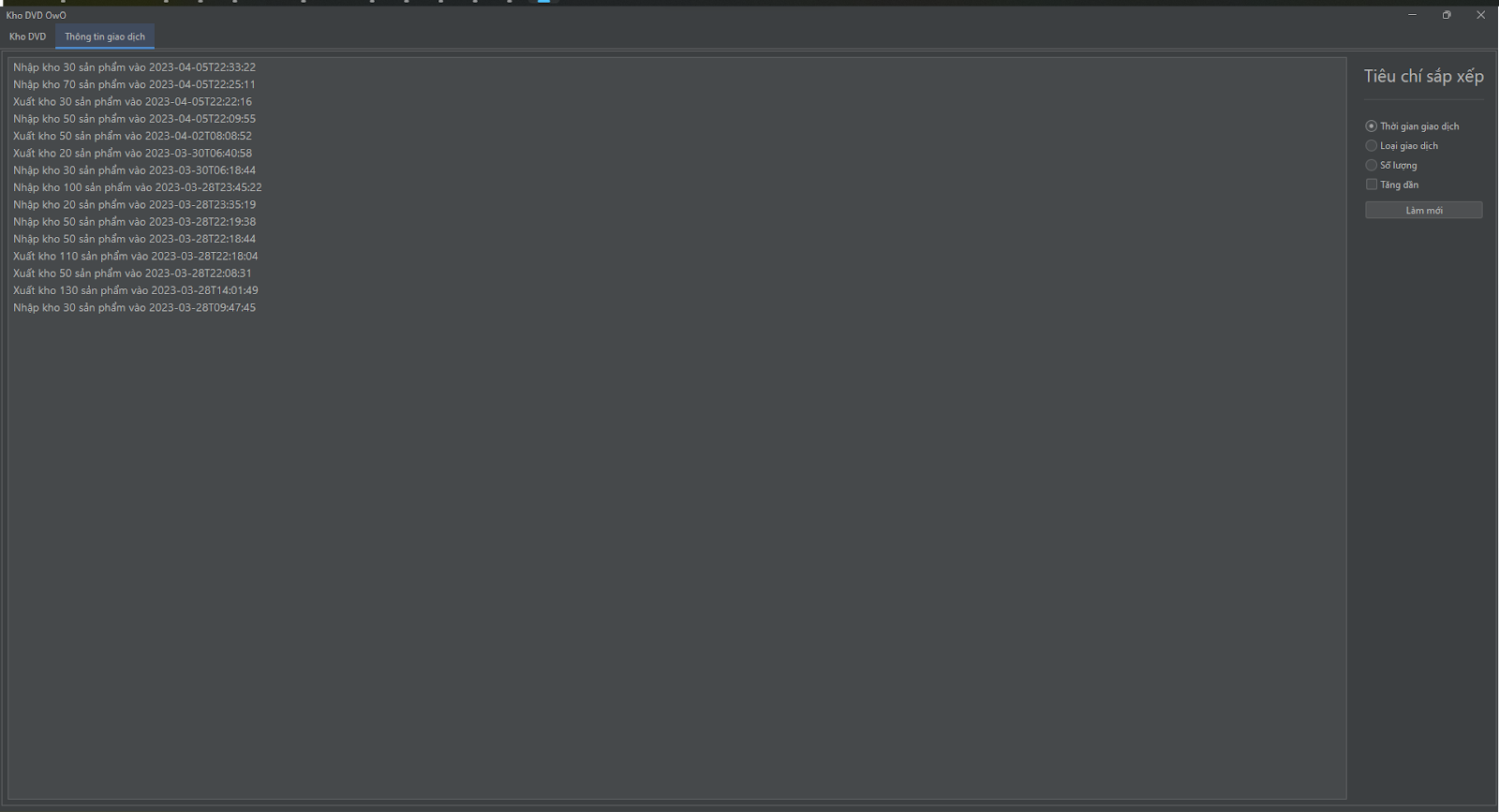


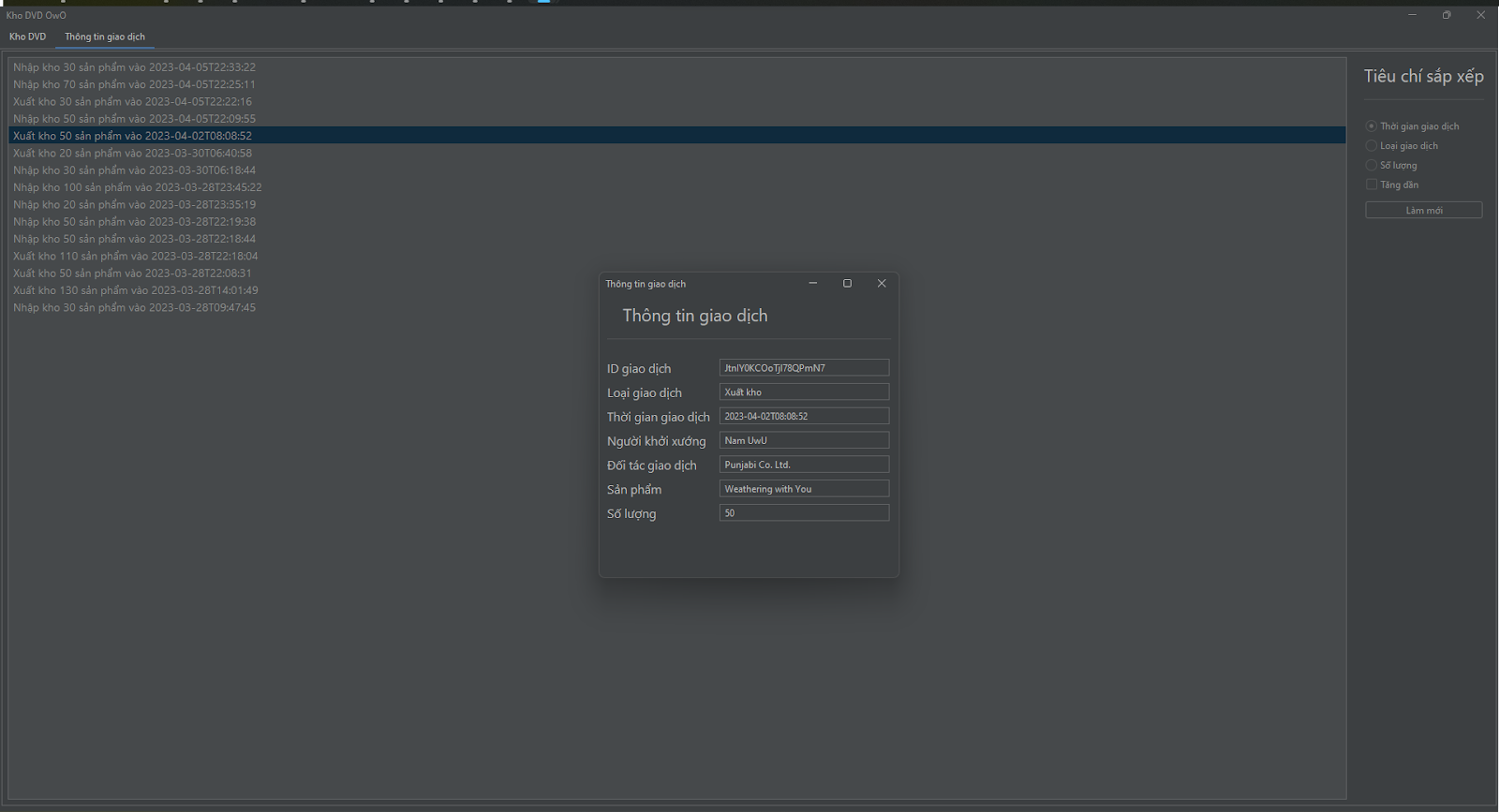


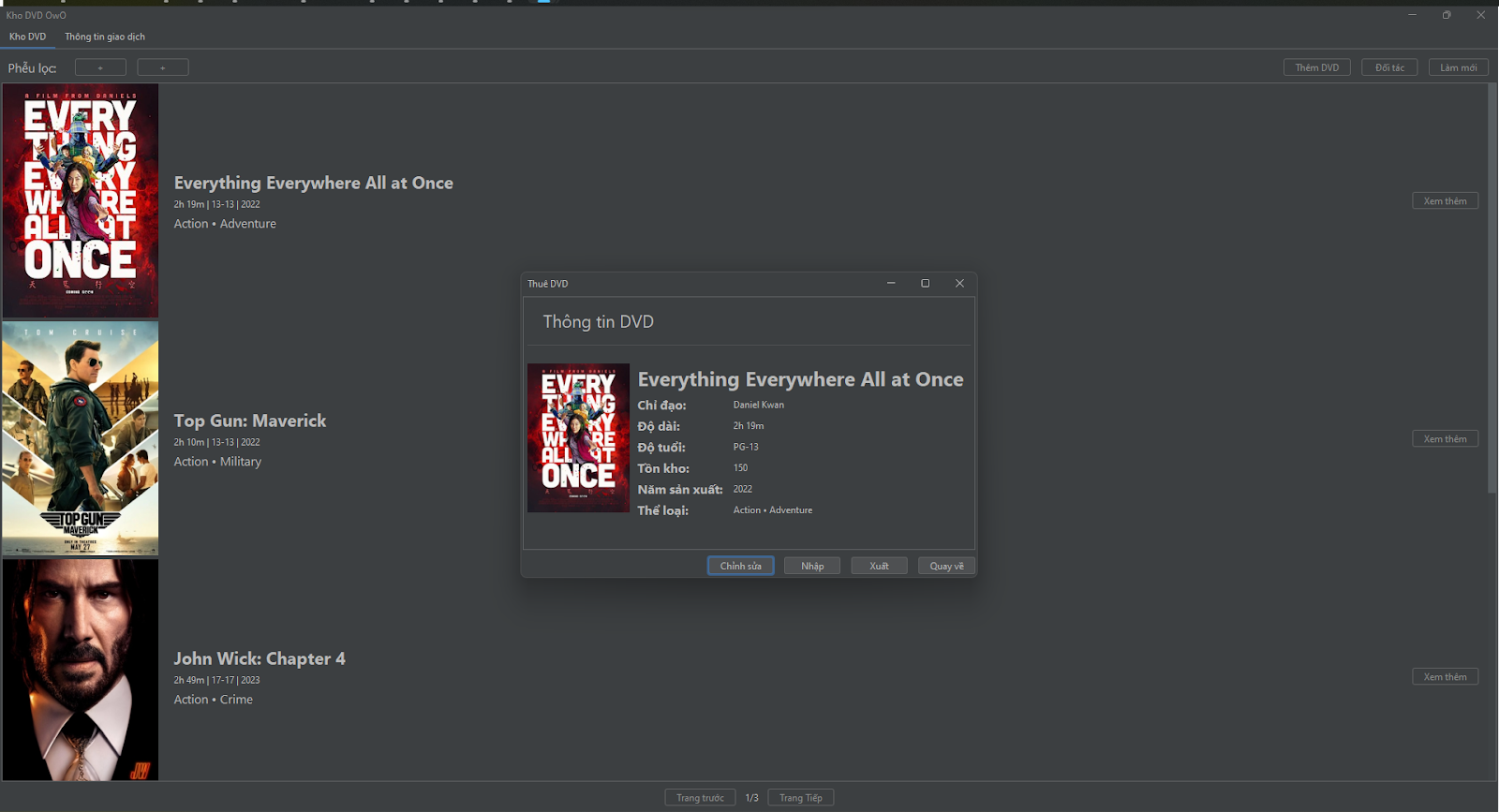


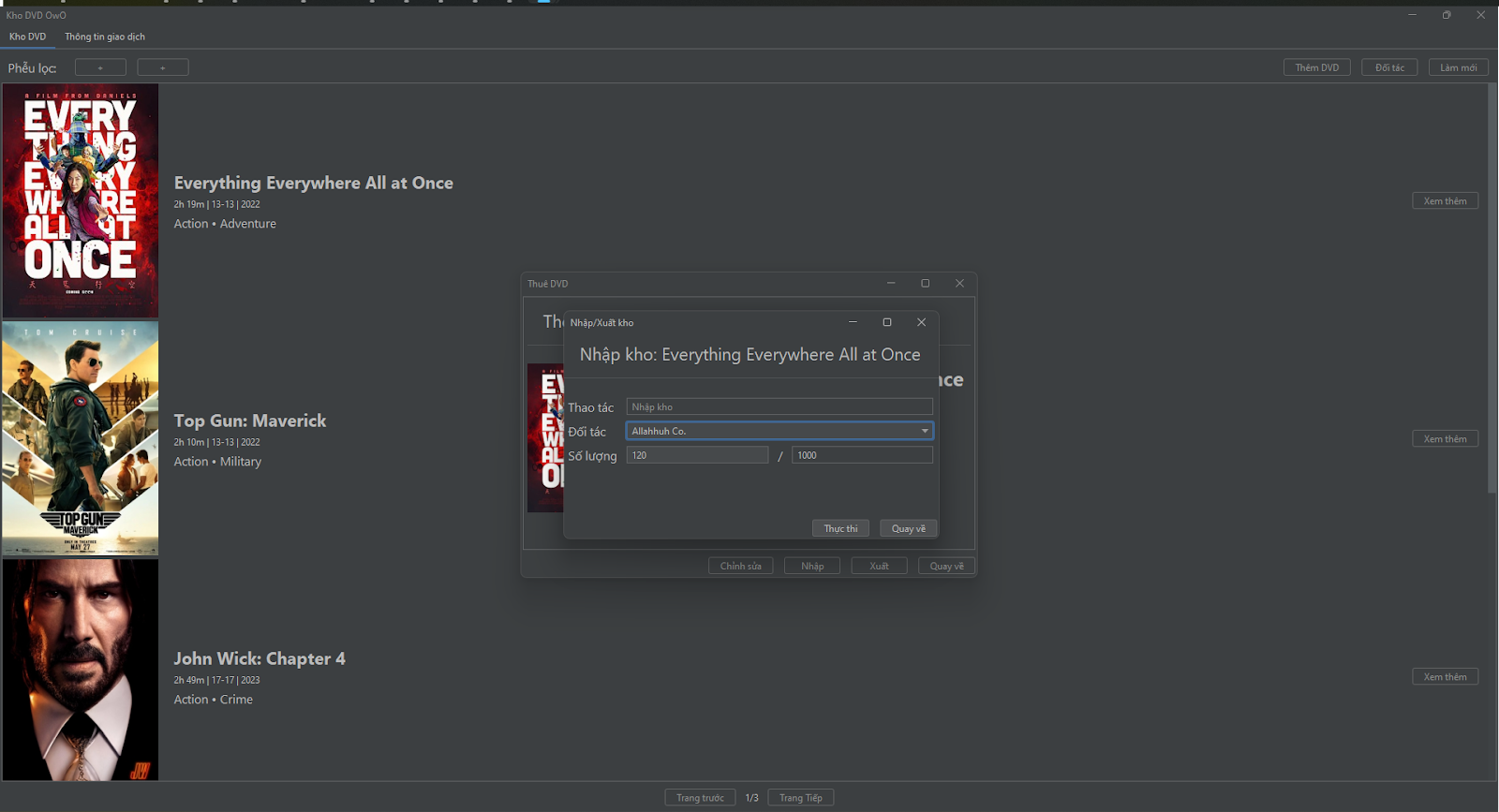












# **Kết luận**

Tuy chương trình còn nhiều thiếu sót như việc giao diện chưa quá đẹp mắt, hay các query chưa đảm bảo giao dịch cơ sở dữ liệu, nhưng tập thể chúng tôi đã rất cố gắng để hoàn thiện được một chương trình quản lý DVD cơ bản, áp dụng nhiều công nghệ mới cho chương trình này, chương trình cơ bản đã đáp ứng đủ nhu cầu cho yêu cầu đặt ra của bài toán. Tuy nhiên để giải quyết hết các vấn đề hiện rõ, phần giao diện sẽ phải được làm lại từ đầu, và Javabase cần được bổ sung thêm các tính năng để mở giao dịch cơ sở dữ liệu, nhằm đảm bảo tính hạt nhân của dữ liệu.

# **Một vài snippet code**

# Mã nguồn của module thực thi bất đồng bộ (com.cycastic.javabase.dispatcher.\*) và module phụ trợ của nó (com.cycastic.javabase.collection.\*)

**com.cycastic.javabase.dispatcher.SafeFlag:**

package com.cycastic.javabase.dispatcher;

import java.util.concurrent.atomic.AtomicBoolean;

public class SafeFlag {

private final AtomicBoolean atomic\_bool;

public SafeFlag(){

atomic\_bool = new AtomicBoolean();

atomic\_bool.set(false);

}

public SafeFlag(boolean starting\_value){

atomic\_bool = new AtomicBoolean();

atomic\_bool.set(starting\_value);

}

public void set(){

atomic\_bool.set(true);

}

public void clear(){

atomic\_bool.set(false);

}

public boolean get() {

return atomic\_bool.get();

}

public void customize(boolean new\_value){

atomic\_bool.set(new\_value);

}

public void waitToFinish() {

while (!get()) { continue; }

}

}

**com.cycastic.javabase.dispatcher.Dispatchable:**

package com.cycastic.javabase.dispatcher;

public class Dispatchable {

private final Runnable process;

private final SafeFlag finished = new SafeFlag(false);

public Dispatchable(Runnable process){

this.process = process;

}

public Runnable getProcess() {

return process;

}

public void setFinished() { finished.set(); }

public boolean isFinished() { return finished.get(); }

public void waitToFinish() { finished.waitToFinish(); }

@Deprecated(since = "1.0")

public void start() {}

}

**com.cycastic.javabase.dispatcher.CommandQueue:**

package com.cycastic.javabase.dispatcher;

import com.cycastic.javabase.collection.FIFOQueue;

public class CommandQueue {

private final FIFOQueue<Dispatchable> processList;

public CommandQueue(){

processList = new FIFOQueue<>();

}

public int queueSize() { return processList.size(); }

public boolean executeOne(){

synchronized (this){

if (processList.isEmpty()) return false;

Dispatchable actualProcess = processList.dequeue();

try {

actualProcess.getProcess().run();

} catch (Exception ex){

actualProcess.setFinished();

throw ex;

}

actualProcess.setFinished();

return true;

}

}

public void executeAll(){

while (executeOne()) { continue; }

}

public Dispatchable dispatch(Runnable process){

synchronized (this){

Dispatchable re = new Dispatchable(process);

processList.enqueue(re);

return re;

}

}

public void sync(Runnable process){

dispatch(process).waitToFinish();

}

}

**com.cycastic.javabase.dispatcher.AsyncEngine:**

package com.cycastic.javabase.dispatcher;

import java.util.concurrent.atomic.AtomicLong;

public class AsyncEngine extends Thread {

public static final int MODE\_HOT = 0;

public static final int MODE\_COLD = 1;

public static final int MODE\_ON\_SPOT = 2;

private final CommandQueue queue;

private final SafeFlag exit = new SafeFlag(false);

private final AtomicLong serverId = new AtomicLong(0L);

private final boolean faultTolerant;

private final int heatMode;

public AsyncEngine(int heatMode, boolean daemonMode, boolean faultTolerant){

queue = new CommandQueue();

this.faultTolerant = faultTolerant;

if (heatMode < MODE\_HOT || heatMode > MODE\_ON\_SPOT) this.heatMode = MODE\_HOT;

else this.heatMode = heatMode;

if (heatMode == MODE\_HOT){

setDaemon(daemonMode);

start();

}

}

public AsyncEngine() { this(MODE\_HOT, false, true); }

public AsyncEngine(int heatMode) { this(heatMode, false, true); }

public AsyncEngine(boolean daemonMode) { this(MODE\_HOT, daemonMode, true); }

public AsyncEngine(int heatMode, boolean daemonMode) { this(heatMode, daemonMode, true); }

@Override

public void run(){

serverId.set(Thread.currentThread().threadId());

while (!exit.get()){

if (!faultTolerant)

queue.executeOne();

else try {

queue.executeOne();

} catch (Exception ex){

System.err.print("Exception caught during asynchronous execution: ");

ex.printStackTrace(System.err);

System.err.println("This exception will be ignored due to fault tolerant policy");

}

}

queue.executeAll();

}

public long getServerId() { return serverId.get(); }

public int queueSize() { return queue.queueSize(); }

public void terminate(){

if (heatMode != MODE\_HOT || exit.get()) return;

if (Thread.currentThread().threadId() == getServerId()){

new Thread(this::terminate).start();

return;

}

exit.set();

while (isAlive()) { continue; }

}

public Dispatchable dispatch(final Runnable process){

if (exit.get()) return null;

switch (heatMode){

case MODE\_HOT -> {

return queue.dispatch(process);

}

case MODE\_COLD -> new Thread(process).start();

case MODE\_ON\_SPOT -> process.run();

}

return new Dispatchable(process);

}

public void sync(final Runnable process){

if (exit.get()) return;

switch (heatMode){

case MODE\_HOT -> queue.sync(process);

case MODE\_COLD -> {

final SafeFlag finished = new SafeFlag(false);

new Thread(() -> {

process.run();

finished.set();

}).start();

finished.waitToFinish();

}

case MODE\_ON\_SPOT -> process.run();

}

}

public void sync(){

sync(() -> {});

}

}

**(Legacy) com.cycastic.javabase.dispatcher.Command:**

package com.cycastic.javabase.dispatcher;

public interface Command {

void exec(Object ...params);

}

**(Legacy) com.cycastic.javabase.dispatcher.CommandPair:**

package com.cycastic.javabase.dispatcher;

public class CommandPair {

public Command cmd;

public Object[] params;

public final SafeFlag isFinished;

public CommandPair(Command cmd, Object[] params){

this.cmd = cmd;

this.params = params;

this.isFinished = new SafeFlag(false);

}

}

**(Legacy) com.cycastic.javabase.dispatcher.CommandQueueCompat:**

package com.cycastic.javabase.dispatcher;

import com.cycastic.javabase.collection.ReferencesList;

public class CommandQueueCompat {

private final ReferencesList<CommandPair> queue;

public CommandQueueCompat() {

queue = new ReferencesList<>();

}

public int queueSize() {

synchronized (queue) { return queue.size(); }

}

public boolean waitAndRunOne(){

synchronized (queue){

ReferencesList.Element<CommandPair> pair = queue.first();

if (pair == null) {

return false;

}

CommandPair actual\_pair = pair.getValue();

actual\_pair.cmd.exec(actual\_pair.params);

actual\_pair.isFinished.set();

queue.erase(pair);

return true;

}

}

public void runAll(){

while (waitAndRunOne()) { continue; }

}

public CommandPair push(Command cmd, Object... params){

synchronized (queue){

CommandPair pair = new CommandPair(cmd, params);

queue.add(pair);

return pair;

}

}

public void pushAndWait(Command cmd, Object... params){

CommandPair pair = push(cmd, params);

while (!pair.isFinished.get()) { continue; }

}

}

**(Legacy) com.cycastic.javabase.dispatcher.Terminator và com.cycastic.javabase.dispatcher.ExecutionLoop:**

package com.cycastic.javabase.dispatcher;

class Terminator extends Thread {

private final ExecutionLoop loop;

public Terminator(ExecutionLoop loop) { this.loop = loop; start(); }

@Override

public void run(){

loop.terminate();

}

}

@Deprecated

public class ExecutionLoop extends Thread {

private final CommandQueueCompat queue;

private long serverId;

private final SafeFlag exit;

public ExecutionLoop(){

queue = new CommandQueueCompat();

exit = new SafeFlag(false);

start();

}

@Override

public void run(){

serverId = Thread.currentThread().getId();

while (!exit.get()){

queue.waitAndRunOne();

}

queue.runAll();

}

public long getServerId() { return serverId; }

public void terminate(){

if (Thread.currentThread().getId() == serverId){

new Terminator(this);

return;

}

exit.set();

while (isAlive()) { continue; }

}

public void push(Command cmd, Object... params){

queue.push(cmd, params);

}

public void pushAndWait(Command cmd, Object... params){

queue.pushAndWait(cmd, params);

}

}

Lớp collection phụ trợ:

**(Legacy) com.cycastic.javabase.collection.ReferenceList:**

package com.cycastic.javabase.collection;

import java.util.Iterator;

import java.util.function.Consumer;

public class ReferencesList<T> implements Iterable<ReferencesList.Element<T>> {

private static class ReferencesHolder<T>{

public Element<T> first = null;

public Element<T> last = null;

public int \_size = 0;

public boolean erase(Element<T> elem){

if (elem.getHolder() != this) return false;

if (first == elem){

first = elem.next();

}

if (last == elem){

last = elem.prev();

}

if (elem.prev() != null) {

Element<T> e\_prev = elem.prev();

e\_prev.setNext(elem.next());

}

if (elem.next() != null){

Element<T> e\_next = elem.next();

e\_next.setPrev(elem.prev());

}

\_size -= 1;

return true;

}

}

public static class Element<T> implements Iterator<Element<T>> {

private T value;

private Element<T> \_next = null;

private Element<T> \_prev = null;

private final ReferencesHolder<T> holder;

public Element(ReferencesHolder<T> holder) { this.holder = holder; }

public ReferencesHolder<T> getHolder() { return holder; }

@Override

public boolean hasNext() {

return \_next != null;

}

public boolean hasPrev(){

return \_prev != null;

}

@Override

public Element<T> next() {

return \_next;

}

Element<T> prev(){

return \_prev;

}

public void setNext(Element<T> n) { \_next = n; }

public void setPrev(Element<T> p) { \_prev = p; }

public T getValue() { return value; }

public void setValue(T val) { value = val; }

@Override

public void remove() {

holder.erase(this);

}

}

private final ReferencesHolder<T> holder;

public ReferencesList(){

holder = new ReferencesHolder<>();

}

public int size() { return holder.\_size; }

public boolean remove(T value){

Element<T> iter = find(value);

if (iter == null) return false;

return holder.erase(iter);

}

public Element<T> find(T value){

Element<T> iter = holder.first;

while (iter != null) {

if (iter.getValue() == value) return iter;

iter = iter.next();

}

return null;

}

public boolean erase(Element<T> elem){

return holder.erase(elem);

}

public void add(T value) { pushBack(value); }

public Element<T> pushBack(T value){

Element<T> elem = new Element<>(holder);

elem.setValue(value);

if (holder.first == null){

holder.first = elem;

} else {

holder.last.setNext(elem);

elem.setPrev(holder.last);

}

holder.last = elem;

holder.\_size += 1;

return elem;

}

public Element<T> pushFront(T value){

Element<T> elem = new Element<>(holder);

elem.setValue(value);

if (holder.last == null){

holder.last = elem;

} else {

holder.first.setPrev(elem);

elem.setNext(holder.first);

}

holder.first = elem;

holder.\_size += 1;

return elem;

}

public Element<T> pushBefore(T value, Element<T> anchor){

if (anchor == null || anchor.getHolder() != holder) return null;

Element<T> elem = new Element<>(holder);

elem.setValue(value);

if (anchor == holder.first){

holder.first = elem;

} else {

elem.setPrev(anchor.prev());

elem.prev().setNext(elem);

}

anchor.setPrev(elem);

elem.setNext(anchor);

holder.\_size += 1;

return elem;

}

public Element<T> pushAfter(T value, Element<T> anchor){

if (anchor.getHolder() != holder) return null;

Element<T> elem = new Element<>(holder);

elem.setValue(value);

if (anchor == holder.last){

holder.last = elem;

} else {

elem.setNext(anchor.next());

elem.next().setPrev(elem);

}

anchor.setNext(elem);

elem.setPrev(anchor);

return elem;

}

@Override @Deprecated

public Iterator<Element<T>> iterator() {

return holder.first;

}

public Element<T> first() { return holder.first; }

public Element<T> last() { return holder.last; }

@Override

public void forEach(Consumer<? super Element<T>> action) {

Element<T> iter = holder.first;

while (iter != null) {

action.accept(iter);

iter = iter.next();

}

}

public String toString(){

StringBuilder builder = new StringBuilder("ReferencesList[ ");

Element<T> iter = holder.first;

while (iter != null) {

builder.append(iter.getValue().toString());

if (iter.hasNext()) builder.append(", ");

iter = iter.next();

}

builder.append(" ]");

return builder.toString();

}

}

**com.cycastic.javabase.collection.FIFOQueue:**

package com.cycastic.javabase.collection;

public class FIFOQueue<T> {

static class Node<T> {

public final T value;

public Node<T> next;

public Node(T value){

this.value = value;

next = null;

}

}

private Node<T> first;

private Node<T> last;

private int cachedSize = 0;

public int size(){

return cachedSize;

}

public boolean isEmpty(){

return size() == 0;

}

public void enqueue(T value){

Node<T> newNode = new Node<>(value);

if (first == null || last == null){

first = newNode;

} else {

last.next = newNode;

}

last = newNode;

cachedSize++;

}

public T dequeue(){

Node<T> iter = first;

if (iter == null) throw new IndexOutOfBoundsException();

T returnValue = iter.value;

first = iter.next;

if (first == null) last = null;

cachedSize--;

return returnValue;

}

}

Mã nguồn của module hỗ trợ kết nối qua HTTP (com.cycastic.javabase.network.\*)

**com.cycastic.javabase.network.HTTPRequestException:**

package com.cycastic.javabase.network;

public class HTTPRequestException extends RuntimeException {

public HTTPRequestException(){

super();

}

public HTTPRequestException(String msg){

super(msg);

}

public HTTPRequestException(String msg, Throwable thrown){

super(msg, thrown);

}

}

**com.cycastic.javabase.network.HTTPResponseListener:**

package com.cycastic.javabase.network;

import java.util.Map;

public abstract class HTTPResponseListener {

public abstract void request\_completed(int result, int response\_code, Map<String, String> headers, String rawResponse);

}

**com.cycastic.javabase.network.HTTPRequest và com.cycastic.javabase.network.RequestPackage:**

package com.cycastic.javabase.network;

import com.cycastic.javabase.dispatcher.AsyncEngine;

import com.cycastic.javabase.dispatcher.Command;

import com.cycastic.javabase.dispatcher.ExecutionLoop;

import com.cycastic.javabase.dispatcher.SafeFlag;

import java.io.\*;

import java.net.HttpURLConnection;

import java.net.URL;

import java.util.\*;

class RequestPackage {

private final int connectionTimeout;

private final String url;

private final Map<String, String> headers;

private final boolean sslOnly;

private final String method;

private final String customData;

private final HTTPResponseListener listener;

private void emit(HTTPResponseListener l, int result, int response\_code, Map<String, String> headers, String raw\_response){

if (l == null) return;

l.request\_completed(result, response\_code, headers, raw\_response);

}

public RequestPackage(int connectionTimeout, String url, Map<String, String> headers, boolean sslOnly, String method, String customData, HTTPResponseListener listener){

this.connectionTimeout = connectionTimeout;

this.url = url;

this.headers = headers;

this.sslOnly = sslOnly;

this.method = method;

this.customData = customData;

this.listener = listener;

}

public void activate(){

try {

URL realUrl = new URL(url);

HttpURLConnection con = (HttpURLConnection)realUrl.openConnection();

if (Objects.equals(method, "PATCH")){

con.setRequestProperty("X-HTTP-Method-Override", "PATCH");

con.setRequestMethod("POST");

} else con.setRequestMethod(method);

con.setConnectTimeout(connectionTimeout);

con.setReadTimeout(connectionTimeout);

for (Map.Entry<String, String> E : headers.entrySet()){

con.setRequestProperty(E.getKey(), E.getValue());

}

con.setDoOutput(true);

DataOutputStream out = new DataOutputStream(con.getOutputStream());

out.writeBytes(customData);

out.flush();

out.close();

int response\_status = con.getResponseCode();

Map<String, String> response\_headers = new HashMap<>();

con.getHeaderFields().entrySet().stream()

.filter(entry -> entry.getKey() != null)

.forEach(entry ->{

String response\_key = entry.getKey();

StringBuilder response\_value = new StringBuilder();

List<String> headers\_values = entry.getValue();

Iterator<String> it = headers\_values.iterator();

if (it.hasNext()){

response\_value.append(it.next());

while (it.hasNext()){

response\_value.append(", ").append(it.next());

}

}

response\_headers.put(response\_key, response\_value.toString());

});

Reader streamReader = null;

if (response\_status >= 300){

// Failed

streamReader = new InputStreamReader(con.getErrorStream());

} else {

// Succeeded

streamReader = new InputStreamReader(con.getInputStream());

}

BufferedReader in = new BufferedReader(streamReader);

String input\_line;

StringBuilder response\_data = new StringBuilder();

while ((input\_line = in.readLine()) != null){

response\_data.append(input\_line);

}

con.disconnect();

emit(listener, 1, response\_status, response\_headers, response\_data.toString());

} catch (IOException e){

emit(listener, 0, -1, new HashMap<>(), "");

}

}

}

public class HTTPRequest {

public int connectionTimeout = 5000;

public final SafeFlag oneshot;

private final AsyncEngine mainLoop;

public HTTPRequest(){

this(AsyncEngine.MODE\_HOT);

}

public HTTPRequest(int heatMode){

this.mainLoop = new AsyncEngine(heatMode);

this.oneshot = new SafeFlag(false);

}

public void terminate() { mainLoop.terminate(); }

public void request(String url, Map<String, String> headers, boolean sslOnly, String method, String customData, HTTPResponseListener listener) {

RequestPackage pck = new RequestPackage(connectionTimeout, url, headers, sslOnly, method, customData, listener);

mainLoop.dispatch(() -> {

pck.activate();

if (oneshot.get()) terminate();

});

}

}

Mã nguồn của module xác thực người dùng (com.cycastic.javabase.auth.\*) và module phụ trợ (com.cycastic.javabase.misc.\*)

**com.cycastic.javabase.auth.FirebaseAuthException:**

package com.cycastic.javabase.auth;

public class FirebaseAuthException extends RuntimeException {

public FirebaseAuthException(){

super();

}

public FirebaseAuthException(String msg){

super(msg);

}

public FirebaseAuthException(String msg, Throwable thrown){

super(msg, thrown);

}

}

**com.cycastic.javabase.auth.FirebaseAuthListener:**

package com.cycastic.javabase.auth;

import java.util.Map;

public abstract class FirebaseAuthListener {

public abstract void onConnectionFailed();

public abstract void onRequestFailed(int response\_code, Map<String, String> headers, String raw\_response);

public abstract void onAuthChanged(FirebaseAuthToken token);

}

**com.cycastic.javabase.auth.FirebaseAuthToken:**

package com.cycastic.javabase.auth;

import java.util.Map;

public class FirebaseAuthToken {

private final Map<String, String> key;

public FirebaseAuthToken(Map<String, String> cleanKey){

key = cleanKey;

}

public String getIdToken(){

return key.get("idtoken");

}

public String getLocalId(){

return key.get("localid");

}

}

**com.cycastic.javabase.auth.FirebaseAuthTokenWrapper:**

package com.cycastic.javabase.auth;

public class FirebaseAuthTokenWrapper {

private FirebaseAuthToken authToken;

public FirebaseAuthToken getAuthToken() {

return authToken;

}

public void setAuthToken(FirebaseAuthToken authToken) {

this.authToken = authToken;

}

}

**com.cycastic.javabase.auth.FirebaseAuth và com.cycastic.javabase.auth.JSONfy:**

package com.cycastic.javabase.auth;

import com.cycastic.javabase.network.HTTPRequest;

import com.cycastic.javabase.network.HTTPResponseListener;

import com.cycastic.javabase.misc.FirebaseConfig;

import org.json.JSONObject;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

class JSONfy {

public static String stringify(Map<String, Object> params){

StringBuilder builder = new StringBuilder();

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

builder.append("{");

params.entrySet().stream()

.filter(entry -> entry.getKey() != null)

.forEach(entry -> {

String key = "\"" + entry.getKey() + "\"";

Object value = entry.getValue();

String value\_str = "";

if (value.getClass() == String.class){

value\_str = "\"" + value + "\"";

} else value\_str = value.toString();

lines.add(key + ": " + value\_str);

});

for (int i = 0, s = lines.size(); i < s; i++){

builder.append(lines.get(i));

if (i != s - 1) {

builder.append(", ");

}

}

builder.append("}");

return builder.toString();

}

}

public class FirebaseAuth {

static final String API\_VERSION = "v1";

static final String BASE\_URL = String.format("https://identitytoolkit.googleapis.com/%s/", API\_VERSION);

static final String SIGNUP\_REQUEST\_URL = "accounts:signUp?key=%s";

static final String LOGIN\_REQUEST\_URL = "accounts:signInWithPassword?key=%s";

private final HTTPRequest httpRequest;

private final Map<String, String> headers;

private final FirebaseAuthTokenWrapper authWrapper;

private String apiKey = "";

private class AuthResponseHandler extends HTTPResponseListener {

private final FirebaseAuthListener activeListener;

public AuthResponseHandler(FirebaseAuthListener listener){

activeListener = listener;

}

private Map<String, String> cleanseKey(Map<String, Object> authResult){

Map<String, String> cleansed = new HashMap<>();

authResult.entrySet().stream()

.filter(entry -> entry.getKey() != null)

.forEach(entry -> {

String key = entry.getKey();

String value = entry.getValue().toString();

cleansed.put(key.replace("\_", "").toLowerCase(), value);

});

return cleansed;

}

@Override

public void request\_completed(int result, int response\_code, Map<String, String> headers, String rawResponse) {

if (result == 0){

if (activeListener != null) activeListener.onConnectionFailed();

return;

}

if (response\_code >= 300 || response\_code < 0 || rawResponse.isEmpty()){

if (activeListener != null) activeListener.onRequestFailed(response\_code, headers, rawResponse);

return;

}

JSONObject body = new JSONObject(rawResponse);

Map<String, Object> bodyAsMap = body.toMap();

FirebaseAuthToken authToken = new FirebaseAuthToken(cleanseKey(bodyAsMap));

authWrapper.setAuthToken(authToken);

if (activeListener != null) activeListener.onAuthChanged(authToken);

}

}

private String createLoginBody(String email,String password){

Map<String, Object> login\_request = new HashMap<>();

login\_request.put("email", email);

login\_request.put("password", password);

login\_request.put("returnSecureToken", true);

return JSONfy.stringify(login\_request);

}

public FirebaseAuth(){

headers = new HashMap<>();

headers.put("Content-Type", "application/json");

headers.put("Accept", "application/json");

httpRequest = new HTTPRequest();

authWrapper = new FirebaseAuthTokenWrapper();

}

public void terminate() { httpRequest.terminate(); }

public FirebaseAuthTokenWrapper getAuthWrapper() { return authWrapper; }

public void enrollConfig(FirebaseConfig config){

if (config != null)

apiKey = config.apiKey;

}

public void loginWithEmailAndPassword(String email, String password, FirebaseAuthListener listener){

if (apiKey.isEmpty()) throw new FirebaseAuthException("No API key found");

String body = createLoginBody(email, password);

httpRequest.request(BASE\_URL + LOGIN\_REQUEST\_URL.formatted(apiKey), headers,

true, "POST", body, new AuthResponseHandler(listener));

}

public void signupWithEmailAndPassword(String email, String password, FirebaseAuthListener listener){

if (apiKey.isEmpty()) throw new FirebaseAuthException("No API key found");

String body = createLoginBody(email, password);

httpRequest.request(BASE\_URL + SIGNUP\_REQUEST\_URL.formatted(apiKey), headers,

true, "POST", body, new AuthResponseHandler(listener));

}

public void loginWithEmailAndPassword(String email, String password) { loginWithEmailAndPassword(email, password, null); }

public void signupWithEmailAndPassword(String email, String password) { signupWithEmailAndPassword(email, password, null); }

public void logout() { authWrapper.setAuthToken(null); }

}

Mã nguồn của module Cloud Firestore (com.cycastic.javabase.firestore.\*)

**Các lớp Exception…:**

package com.cycastic.javabase.firestore;

public class FirestoreException extends RuntimeException {

public FirestoreException(){

super();

}

public FirestoreException(String msg){

super(msg);

}

public FirestoreException(String msg, Throwable thrown){

super(msg, thrown);

}

}

package com.cycastic.javabase.firestore;

public class FirestoreModelException extends FirestoreException {

public FirestoreModelException(){

super();

}

public FirestoreModelException(String msg){

super(msg);

}

public FirestoreModelException(String msg, Throwable thrown){

super(msg, thrown);

}

}

package com.cycastic.javabase.firestore;

public class FaultyConvertionException extends FirestoreModelException {

public FaultyConvertionException(){

super();

}

public FaultyConvertionException(String msg){

super(msg);

}

public FaultyConvertionException(String msg, Throwable thrown){

super(msg, thrown);

}

}

package com.cycastic.javabase.firestore;

public class FieldNotFoundException extends FirestoreModelException {

public FieldNotFoundException(){

super();

}

public FieldNotFoundException(String msg){

super(msg);

}

public FieldNotFoundException(String msg, Throwable thrown){

super(msg, thrown);

}

}

package com.cycastic.javabase.firestore;

public class IncorrectTypeException extends FirestoreModelException {

public IncorrectTypeException(){

super();

}

public IncorrectTypeException(String msg){

super(msg);

}

public IncorrectTypeException(String msg, Throwable thrown){

super(msg, thrown);

}

}

package com.cycastic.javabase.firestore;

public class NoConnectionException extends FirestoreModelException {

public NoConnectionException(){

super();

}

public NoConnectionException(String msg){

super(msg);

}

public NoConnectionException(String msg, Throwable thrown){

super(msg, thrown);

}

}

package com.cycastic.javabase.firestore;

public class QueryRefusedException extends FirestoreModelException {

public QueryRefusedException(){

super();

}

public QueryRefusedException(String msg){

super(msg);

}

public QueryRefusedException(String msg, Throwable thrown){

super(msg, thrown);

}

}

**com.cycastic.javabase.firestore.FirestoreTaskReceiver:**

package com.cycastic.javabase.firestore;

import java.util.Map;

public abstract class FirestoreTaskReceiver {

public abstract void connectionFailed();

public abstract void requestFailed(int response\_code, Map<String, String> headers, String rawResponse);

public abstract void queryCompleted(FirestoreTaskResult queryResult);

}

**com.cycastic.javabase.firestore.FirestoreQuery:**

package com.cycastic.javabase.firestore;

import org.json.JSONObject;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class FirestoreQuery {

private final QueryType type;

private final String httpMethod;

private boolean updated = false;

private String lastSerialization = "";

private String collectionName = "";

private String documentId = "";

private String databaseName = "(default)";

public enum Direction {

DIRECTION\_UNSPECIFIED,

ASCENDING,

DESCENDING;

public static String toString(Direction dir){

switch (dir){

case DIRECTION\_UNSPECIFIED -> {

return "DIRECTION\_UNSPECIFIED";

}

case ASCENDING -> {

return "ASCENDING";

}

case DESCENDING -> {

return "DESCENDING";

}

}

return "";

}

}

public enum Operator {

OPERATOR\_UNSPECIFIED,

LESS\_THAN,

LESS\_THAN\_OR\_EQUAL,

GREATER\_THAN,

GREATER\_THAN\_OR\_EQUAL,

EQUAL,

NOT\_EQUAL,

ARRAY\_CONTAINS,

ARRAY\_CONTAINS\_ANY,

IN,

NOT\_IN,

IS\_NAN,

IS\_NULL,

IS\_NOT\_NAN,

IS\_NOT\_NULL,

AND,

OR;

public static String toString(Operator op){

switch (op){

case OPERATOR\_UNSPECIFIED -> { return "OPERATOR\_UNSPECIFIED"; }

case LESS\_THAN -> { return "LESS\_THAN"; }

case LESS\_THAN\_OR\_EQUAL -> { return "LESS\_THAN\_OR\_EQUAL"; }

case GREATER\_THAN -> { return "GREATER\_THAN"; }

case GREATER\_THAN\_OR\_EQUAL -> { return "GREATER\_THAN\_OR\_EQUAL"; }

case EQUAL -> { return "EQUAL"; }

case NOT\_EQUAL -> { return "NOT\_EQUAL"; }

case ARRAY\_CONTAINS -> { return "ARRAY\_CONTAINS"; }

case ARRAY\_CONTAINS\_ANY -> { return "ARRAY\_CONTAINS\_ANY"; }

case IN -> { return "IN"; }

case NOT\_IN -> { return "NOT\_IN"; }

case IS\_NAN -> { return "IS\_NAN"; }

case IS\_NULL -> { return "IS\_NULL"; }

case IS\_NOT\_NAN -> { return "IS\_NOT\_NAN"; }

case IS\_NOT\_NULL -> { return "IS\_NOT\_NULL"; }

case AND -> { return "AND"; }

case OR -> { return "OR"; }

default -> { return ""; }

}

}

public static Operator fromString(String op){

switch (op){

case "OPERATOR\_UNSPECIFIED" -> { return OPERATOR\_UNSPECIFIED; }

case "LESS\_THAN" -> { return LESS\_THAN; }

case "LESS\_THAN\_OR\_EQUAL" -> { return LESS\_THAN\_OR\_EQUAL; }

case "GREATER\_THAN" -> { return GREATER\_THAN; }

case "GREATER\_THAN\_OR\_EQUAL" -> { return GREATER\_THAN\_OR\_EQUAL; }

// case "EQUAL" -> { return EQUAL; }

case "NOT\_EQUAL" -> { return NOT\_EQUAL; }

case "ARRAY\_CONTAINS" -> { return ARRAY\_CONTAINS; }

case "ARRAY\_CONTAINS\_ANY" -> { return ARRAY\_CONTAINS\_ANY; }

case "IN" -> { return IN; }

case "NOT\_IN" -> { return NOT\_IN; }

case "IS\_NAN" -> { return IS\_NAN; }

case "IS\_NULL" -> { return IS\_NULL; }

case "IS\_NOT\_NAN" -> { return IS\_NOT\_NAN; }

case "IS\_NOT\_NULL" -> { return IS\_NOT\_NULL; }

case "AND" -> { return AND; }

case "OR" -> { return OR; }

default -> { return EQUAL; }

}

}

}

public enum QueryType {

STRUCTURED\_QUERY,

CREATE\_DOCUMENT,

PATCH\_DOCUMENT,

DELETE\_DOCUMENT,

}

public QueryType getQueryType(){

return type;

}

public String getCollectionName(){

return collectionName;

}

public String getDocumentId(){

return documentId;

}

public String getDatabaseName() { return databaseName; }

public FirestoreQuery(QueryType t){

type = t;

if (type == QueryType.PATCH\_DOCUMENT) httpMethod = "PATCH";

else if (type == QueryType.DELETE\_DOCUMENT) httpMethod = "DELETE";

else httpMethod = "POST";

}

public FirestoreQuery(){

this(QueryType.STRUCTURED\_QUERY);

}

static Map<String, Object> orderObject(String field, Direction dir){

Map<String, Object> \_f = new HashMap<>();

Map<String, Object> \_ff = new HashMap<>();

\_ff.put("fieldPath", field);

\_f.put("field", \_ff);

\_f.put("direction", Direction.toString(dir));

return \_f;

}

public static String getTypeIndicator(Object value){

String typeIndicator = "";

if (value.getClass() == String.class) {

typeIndicator = "stringValue";

} else if (value.getClass() == Integer.class || value.getClass() == Long.class) {

typeIndicator = "integerValue";

} else if (value.getClass() == Double.class || value.getClass() == Float.class) {

typeIndicator = "doubleValue";

} else if (value.getClass() == Boolean.class){

typeIndicator = "booleanValue";

} else {

// Special orders

try {

List<Object> l = (List<Object>)value;

typeIndicator = "arrayValue";

} catch (ClassCastException ignored) {}

if (!typeIndicator.isEmpty()) return typeIndicator;

try {

Map<String, Object> m = (Map<String, Object>)value;

typeIndicator = "mapValue";

} catch (ClassCastException ignored) {}

if (!typeIndicator.isEmpty()) return typeIndicator;

}

if (typeIndicator.isEmpty()) throw new FirestoreException("Unsupported type");

return typeIndicator;

}

private Integer \_\_limit = 0;

private Boolean \_\_doc\_existed = true;

private Operator \_\_composite\_operation = Operator.AND;

private final List<Map<String, Object>> \_\_order\_by = new ArrayList<>();

private final List<Map<String, String>> \_\_from = new ArrayList<>();

private final List<Map<String, String>> \_\_select = new ArrayList<>();

private final List<Map<String, Object>> \_\_composite\_filters = new ArrayList<>();

private final List<String> \_\_update\_mask = new ArrayList<>();

private final List<String> \_\_mask = new ArrayList<>();

private Map<String, Object> \_\_create = new HashMap<>();

private String buildStructuredQuery(){

if (updated) return lastSerialization;

Map<String, Object> \_fake\_json = new HashMap<>();

JSONObject \_json = new JSONObject("{}");

if (\_\_limit > 0) \_fake\_json.put("limit", \_\_limit);

if (!\_\_order\_by.isEmpty()) \_fake\_json.put("orderBy", \_\_order\_by);

if (!\_\_from.isEmpty()) \_fake\_json.put("from", \_\_from);

if (!\_\_composite\_filters.isEmpty()){

Map<String, Object> real\_cf = new HashMap<>();

Map<String, Object> real\_filter = new HashMap<>();

real\_cf.put("op", Operator.toString(\_\_composite\_operation));

real\_cf.put("filters", \_\_composite\_filters);

real\_filter.put("compositeFilter", real\_cf);

\_fake\_json.put("where", real\_filter);

}

if (!\_\_select.isEmpty()) {

Map<String, List<Map<String, String>>> real\_select = new HashMap<>();

real\_select.put("fields", \_\_select);

\_fake\_json.put("select", real\_select);

}

\_json.put("structuredQuery", \_fake\_json);

lastSerialization = \_json.toString(4);

updated = true;

return lastSerialization;

}

public FirestoreQuery select(String field\_path){

if (type != QueryType.STRUCTURED\_QUERY) return this;

Map<String, String> \_field\_path = new HashMap<>();

updated = false;

\_field\_path.put("fieldPath", field\_path);

\_\_select.add(\_field\_path);

return this;

}

public FirestoreQuery from(String collection){

if (type != QueryType.STRUCTURED\_QUERY) return this;

Map<String, String> \_from = new HashMap<>();

updated = false;

\_from.put("collectionId", collection);

\_\_from.add(\_from);

return this;

}

public FirestoreQuery orderBy(String field, Direction dir){

if (type != QueryType.STRUCTURED\_QUERY) return this;

updated = false;

\_\_order\_by.add(orderObject(field, dir));

return this;

}

public FirestoreQuery limit(int by){

if (type != QueryType.STRUCTURED\_QUERY || by < 1) return this;

updated = false;

\_\_limit = by;

return this;

}

public FirestoreQuery relate(Operator op){

updated = false;

\_\_composite\_operation = op;

return this;

}

public FirestoreQuery where(String field, Operator op, Object value){

if (type != QueryType.STRUCTURED\_QUERY) return this;

updated = false;

String type\_indicator = getTypeIndicator(value);

Map<String, Object> \_field\_filter = new HashMap<>();

Map<String, Object> \_real\_field\_filter = new HashMap<>();

Map<String, Object> \_field\_path = new HashMap<>();

Map<String, Object> \_value = new HashMap<>();

\_value.put(type\_indicator, value);

\_field\_path.put("fieldPath", field);

\_real\_field\_filter.put("field", \_field\_path);

\_real\_field\_filter.put("op", Operator.toString(op));

\_real\_field\_filter.put("value", \_value);

\_field\_filter.put("fieldFilter", \_real\_field\_filter);

\_\_composite\_filters.add(\_field\_filter);

return this;

}

private Map<String, Object> serializeValue(Object target){

final String type = getTypeIndicator(target);

final Map<String, Object> \_value = new HashMap<>();

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

Map<String, Object> \_another\_value = new HashMap<>();

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

for (Object childObject : (List<Object>)target){

serializedArray.add(serializeValue(childObject));

}

\_another\_value.put("values", serializedArray);

\_value.put("arrayValue", \_another\_value);

} else if (type.equals("mapValue")){

Map<String, Object> \_another\_value = new HashMap<>();

\_another\_value.put("fields", target);

\_value.put("mapValue", \_another\_value);

} else {

\_value.put(type, target);

}

return \_value;

}

public FirestoreQuery create(Map<String, Object> document){

if (type != QueryType.CREATE\_DOCUMENT && type != QueryType.PATCH\_DOCUMENT) return this;

updated = false;

Map<String, Object> \_create = new HashMap<>();

for (Map.Entry<String, Object> E : document.entrySet()){

// BREAKING CHANGES

Map<String, Object> \_value = serializeValue(E.getValue());

\_create.put(E.getKey(), \_value);

}

\_\_create = \_create;

return this;

}

public FirestoreQuery update(Map<String, Object> document) { return create(document); }

public FirestoreQuery mask(String field\_mask) {

if (type != QueryType.PATCH\_DOCUMENT) return this;

updated = false;

\_\_mask.add(field\_mask);

return this;

}

public FirestoreQuery updateMask(String field\_mask) {

if (type != QueryType.PATCH\_DOCUMENT) return this;

updated = false;

\_\_update\_mask.add(field\_mask);

return this;

}

public FirestoreQuery documentExisted(boolean existence) {

if (type != QueryType.PATCH\_DOCUMENT) return this;

updated = false;

\_\_doc\_existed = existence;

return this;

}

public FirestoreQuery onCollection(String c\_name){

updated = false;

collectionName = c\_name;

return this;

}

public FirestoreQuery onDocument(String d\_name){

updated = false;

documentId = d\_name;

return this;

}

public FirestoreQuery onDatabase(String db\_name){

databaseName = db\_name;

return this;

}

public String getLastSerialization() { return lastSerialization; }

public String getHttpMethod() { return httpMethod; }

public String getSubUrl() {

switch (type){

case STRUCTURED\_QUERY -> { return ":runQuery"; }

case CREATE\_DOCUMENT -> {

String re = getCollectionName();

if (!getDocumentId().isEmpty()){

re += "?documentId=" + getDocumentId();

}

return re;

}

case PATCH\_DOCUMENT -> {

StringBuilder re = new StringBuilder(getCollectionName());

re.append("/").append(getDocumentId()).append("?");

re.append("currentDocument.exists=").append(\_\_doc\_existed.toString());

for (String mask : \_\_mask){

re.append("&mask.fieldPaths=").append(mask);

}

for (String update\_mask : \_\_update\_mask){

re.append("&updateMask.fieldPaths=").append(update\_mask);

}

return re.toString();

}

case DELETE\_DOCUMENT -> {

return getCollectionName() + "/" + getDocumentId() + "/?currentDocument.exists=true";

}

default -> { return ""; }

}

}

public String toHttpBody(){

switch (type){

case STRUCTURED\_QUERY -> {

return buildStructuredQuery();

}

case CREATE\_DOCUMENT, PATCH\_DOCUMENT -> {

lastSerialization = new JSONObject("{}").put("fields", \_\_create).toString(4);

updated = true;

return lastSerialization;

}

default -> { return "{}"; }

}

}

}

**com.cycastic.javabase.firestore.FirestoreDocument:**

package com.cycastic.javabase.firestore;

import org.json.JSONObject;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Vector;

public class FirestoreDocument {

String documentName = "";

String documentNameShort = "";

String readTime = "";

String createTime = "";

String updateTime = "";

Map<String, Object> fields = new HashMap<>();

private static String getDocName(String base\_name){

for (int i = base\_name.length() - 1; i >= 0; i--){

if (base\_name.charAt(i) == '/'){

return base\_name.substring(i + 1);

}

}

return "";

}

public String getDocumentPath() {

return documentName;

}

public String getDocumentName() {

return documentNameShort;

}

public String getReadTime() {

return readTime;

}

public String getCreateTime() {

return createTime;

}

public String getUpdateTime() {

return updateTime;

}

public Map<String, Object> getFields(){

return fields;

}

public FirestoreDocument(){}

Object parseSingleField(Map<String, Object> raw\_data){

Object re = null;

if (raw\_data.containsKey("stringValue")){

re = raw\_data.get("stringValue").toString();

} else if (raw\_data.containsKey("integerValue")) {

re = Long.valueOf(raw\_data.get("integerValue").toString());

} else if (raw\_data.containsKey("doubleValue")){

re = Double.valueOf(raw\_data.get("doubleValue").toString());

} else if (raw\_data.containsKey("booleanValue")){

re = Boolean.valueOf(raw\_data.get("booleanValue").toString());

} else if (raw\_data.containsKey("arrayValue")){

Map<String, Object> arr\_check = (Map<String, Object>)raw\_data.get("arrayValue");

if (arr\_check.isEmpty()) re = new Vector<>();

else {

List<Object> arr = (List<Object>)(arr\_check).get("values");

re = parseArrayValue(arr);

}

} else if (raw\_data.containsKey("mapValue")) {

Map<String, Object> map\_check = (Map<String, Object>)raw\_data.get("mapValue");

if (map\_check.isEmpty()) re = new HashMap<String, Object>();

else {

Map<String, Object> arr = (Map<String, Object>)(map\_check).get("fields");

re = parseMapValue(arr);

}

}

return re;

}

List<Object> parseArrayValue(List<Object> array){

List<Object> re = new Vector<>();

for (Object o : array) {

Object cleansed = parseSingleField((Map<String, Object>) o);

re.add(cleansed);

}

return re;

}

Map<String, Object> parseMapValue(Map<String, Object> raw\_data){

Map<String, Object> re = new HashMap<>();

for (Map.Entry<String, Object> E : raw\_data.entrySet()){

Map<String, Object> field = ((Map<String, Object>)E.getValue());

String field\_name = E.getKey();

Object value = parseSingleField(field);

re.put(field\_name, value);

}

return re;

}

public FirestoreDocument parse(JSONObject origin){

Map<String, Object> base\_map = origin.toMap();

// if (base\_map.containsKey("readTime"))

// read\_time = base\_map.get("readTime").toString();

if (!base\_map.containsKey("document")) return this;

return parseDocument((Map<String, Object>)base\_map.get("document"));

}

public FirestoreDocument parseDocument(Map<String, Object> doc){

documentName = doc.get("name").toString();

documentNameShort = getDocName(documentName);

createTime = doc.get("createTime").toString();

updateTime = doc.get("updateTime").toString();

if (!doc.containsKey("fields")) return this;

fields = parseMapValue((Map<String, Object>)doc.get("fields"));

return this;

}

}

**com.cycastic.javabase.firestore.FirestoreTaskResult:**

package com.cycastic.javabase.firestore;

import org.json.JSONArray;

import org.json.JSONException;

import org.json.JSONObject;

import java.util.Map;

import java.util.LinkedHashMap;

public class FirestoreTaskResult {

private final int responseCode;

private final Map<String, String> headers;

private final String content;

private final FirestoreQuery baseQuery;

private final Map<String, FirestoreDocument> documents;

public int getResponseCode() { return responseCode; }

public Map<String, String> getResponseHeaders() { return headers; }

public String getResponseContent() { return content; }

public FirestoreQuery getBaseQuery() { return baseQuery; }

public Map<String, FirestoreDocument> getDocuments() { return documents; }

public FirestoreTaskResult(int responseCode, Map<String, String> headers, String content, FirestoreQuery baseQuery){

this.responseCode = responseCode;

this.headers = headers;

this.content = content;

this.baseQuery = baseQuery;

this.documents = new LinkedHashMap<>();

if (responseCode == -1) return;

if (responseCode >= 300) return;

JSONArray task\_result;

try {

task\_result = new JSONArray(content);

if (task\_result.isEmpty()) return;

} catch (JSONException ignored){

try {

JSONObject stuff = new JSONObject(content);

if (stuff.isEmpty()) return;

FirestoreDocument doc = new FirestoreDocument().parseDocument(stuff.toMap());

if (!doc.getDocumentName().isEmpty())

documents.put(doc.getDocumentName(), doc);

} catch (JSONException ignored1) {

throw new FirestoreException("Failed to convert rawResponse into meaningful JSON");

}

return;

}

for (int i = 0, s = task\_result.length(); i < s; i++){

Object stuff = task\_result.get(i);

if (stuff == null) continue;

FirestoreDocument doc = new FirestoreDocument().parse((JSONObject) stuff);

if (doc.getDocumentName().isEmpty()) continue;

documents.put(doc.getDocumentName(), doc);

}

}

}

**com.cycastic.javabase.firestore.FirestoreModel:**

package com.cycastic.javabase.firestore;

import com.cycastic.javabase.dispatcher.SafeFlag;

public class FirestoreModel {

protected final Firestore host;

protected final boolean lazyEvaluation;

protected String documentName = "";

protected FirestoreModelException exception;

protected final SafeFlag ranOnce = new SafeFlag(false);

private final SafeFlag evaluationFlag = new SafeFlag(true);

public FirestoreModel(Firestore host, boolean lazyEvaluation){

this.host = host;

this.lazyEvaluation = lazyEvaluation;

if (!lazyEvaluation) evaluate();

}

public FirestoreModel(Firestore host){

this(host, true);

}

public String getDocumentName() {

return documentName;

}

public void setDocumentName(String documentName) {

this.documentName = documentName;

}

public void evaluate(){

synchronized (ranOnce) {

ranOnce.set();

}

synchronized (ranOnce){

if (isEvaluating()) return;

evaluationFlag.clear();

evaluatePrivate();

evaluationFlag.waitToFinish();

if (exception != null) throw exception;

}

}

public boolean isEvaluating() { return !evaluationFlag.get(); }

protected void setEvaluationFlag(){

evaluationFlag.set();

}

protected void evaluatePrivate(){

exception = new FirestoreModelException("Not implemented");

setEvaluationFlag();

}

}

**com.cycastic.javabase.firestore.FirestoreCollectionModel:**

package com.cycastic.javabase.firestore;

import java.util.Map;

import java.util.LinkedHashMap;

public class FirestoreCollectionModel<T extends FirestoreModel> extends FirestoreModel{

protected FirestoreQuery baseQuery;

protected final Map<String, T> cachedValue = new LinkedHashMap<>();

public FirestoreCollectionModel(Firestore host, boolean lazyEvaluation, FirestoreQuery baseQuery) {

super(host, lazyEvaluation);

this.baseQuery = baseQuery;

}

public FirestoreCollectionModel(Firestore host, boolean lazyEvaluation) {

super(host, lazyEvaluation);

this.baseQuery = null;

}

public FirestoreCollectionModel(Firestore host) {

super(host);

this.baseQuery = null;

}

public FirestoreCollectionModel<T> setBaseQuery(FirestoreQuery baseQuery) {

this.baseQuery = baseQuery;

return this;

}

public FirestoreQuery getBaseQuery() { return baseQuery; }

public Map<String, T> getDocuments(){

synchronized (ranOnce){

if (!ranOnce.get() && lazyEvaluation) evaluate();

}

while (isEvaluating()) { continue; }

return cachedValue;

}

private void evaluationFinalize(){

cachedValue.clear();

setEvaluationFlag();

}

@Override

protected void evaluatePrivate() {

if (baseQuery == null){

exception = new FirestoreModelException("No Query");

evaluationFinalize();

return;

}

if (baseQuery.getQueryType() == FirestoreQuery.QueryType.STRUCTURED\_QUERY)

host.query(baseQuery, new FirestoreTaskReceiver() {

@Override

public void connectionFailed() {

exception = new NoConnectionException();

evaluationFinalize();

}

@Override

public void requestFailed(int response\_code, Map<String, String> headers, String rawResponse) {

exception = new QueryRefusedException("Response code: %d. Raw response: %s".formatted(response\_code, rawResponse));

evaluationFinalize();

}

@Override

public void queryCompleted(FirestoreTaskResult queryResult) {

taskResultCleanse(queryResult);

setEvaluationFlag();

}

});

else if (baseQuery.getQueryType() == FirestoreQuery.QueryType.CREATE\_DOCUMENT ||

baseQuery.getQueryType() == FirestoreQuery.QueryType.PATCH\_DOCUMENT){

Map<String, Object> serialized = serialize();

baseQuery.update(serialized);

host.query(baseQuery, new FirestoreTaskReceiver() {

@Override

public void connectionFailed() {

exception = new NoConnectionException();

evaluationFinalize();

}

@Override

public void requestFailed(int response\_code, Map<String, String> headers, String rawResponse) {

exception = new QueryRefusedException("Response code: %d. Raw response: %s".formatted(response\_code, rawResponse));

evaluationFinalize();

}

@Override

public void queryCompleted(FirestoreTaskResult queryResult) {

setEvaluationFlag();

}

});

} else {

exception = new FirestoreModelException("Not implemented");

evaluationFinalize();

}

}

protected void taskResultCleanse(FirestoreTaskResult queryResult){

}

protected Map<String, Object> serialize(){

return null;

}

}

**com.cycastic.javabase.firestore.Firestore:**

package com.cycastic.javabase.firestore;

import com.cycastic.javabase.auth.FirebaseAuthTokenWrapper;

import com.cycastic.javabase.dispatcher.AsyncEngine;

import com.cycastic.javabase.misc.FirebaseConfig;

import com.cycastic.javabase.network.HTTPRequest;

import com.cycastic.javabase.network.HTTPResponseListener;

import java.util.HashMap;

import java.util.Map;

public class Firestore {

static final String API\_VERSION = "v1";

static final String BASE\_URL = String.format("https://firestore.googleapis.com/%s/projects/", API\_VERSION);

static final String BASE\_SUB\_URL = "/databases/%s/documents/";

private FirebaseAuthTokenWrapper authWrapper;

private String projectId;

private final HTTPRequest httpRequest;

private final boolean dispatched;

public Firestore(boolean dispatched){

this.dispatched = dispatched;

httpRequest = new HTTPRequest(dispatched ? AsyncEngine.MODE\_COLD : AsyncEngine.MODE\_HOT);

}

public Firestore(){

this(false);

}

public void enrollToken(FirebaseAuthTokenWrapper authToken){

this.authWrapper = authToken;

}

public void enrollConfig(FirebaseConfig config){

if (config != null)

projectId = config.projectId;

}

public void terminate() { httpRequest.terminate(); }

private void queryInternal(FirestoreQuery request, FirestoreTaskReceiver receiver){

if (projectId.isEmpty()) throw new FirestoreException("projectId not found");

Map<String, String> headers = new HashMap<>();

if (authWrapper != null && authWrapper.getAuthToken() != null) {

headers.put("Content-Type", "application/json");

headers.put("Accept", "application/json");

headers.put("Authorization", "Bearer %s".formatted(authWrapper.getAuthToken().getIdToken()));

}

String url = "";

url = BASE\_URL + projectId + BASE\_SUB\_URL.formatted(request.getDatabaseName());

url += request.getSubUrl();

httpRequest.request(url, headers, true, request.getHttpMethod(), request.toHttpBody(), new HTTPResponseListener() {

@Override

public void request\_completed(int result, int responseCode, Map<String, String> headers, String rawResponse) {

if (receiver == null) return;

if (result == 0) {

receiver.connectionFailed();

} else if (responseCode >= 300) {

receiver.requestFailed(responseCode, headers, rawResponse);

} else {

receiver.queryCompleted(new FirestoreTaskResult(responseCode, headers, rawResponse, request));

}

}

});

}

public void query(FirestoreQuery request){

query(request, null);

}

public void query(FirestoreQuery request, FirestoreTaskReceiver receiver){

if (dispatched) new Thread(() -> queryInternal(request, receiver)).start();

else queryInternal(request, receiver);

}

}

# 

# **Nguồn**

(1): <https://www.businessinsider.de/the-10-most-popular-programming-languages-according-to-github-2018-10?op=1>

(2): <https://github.com/openjdk/jdk/blob/master/LICENSE>

(3): <http://www.infoworld.com/d/developer-world/infoworld-review-top-java-programming-tools-191>

(4): <https://en.wikipedia.org/wiki/IntelliJ_IDEA>

(5): <https://en.wikipedia.org/wiki/Customer_identity_access_management>

(6): <https://firebase.google.com/docs/reference/rest/auth/>

(7): <https://en.wikipedia.org/wiki/Command_queue>

(8): <https://github.com/godotengine/godot/blob/master/core/templates/command_queue_mt.h>