Grupo 11
VICTOR BAZAGA VELASCO
ANGEL DONET GRANADO
RICARDO SANABRIA VEGA

RMI ACADÉMICO

INDICE

1. Introducción

Sistema Académico Distribuido con Java RMI

Este proyecto implementa un Sistema de Gestión Académica utilizando Java RMI, una tecnología que permite la comunicación entre objetos distribuidos en diferentes máquinas virtuales de Java.

Nuestro sistema está diseñado para administrar tres tipos de usuarios (administradores, profesores y alumnos), cada uno con permisos y funcionalidades específicas para interactuar con asignaturas, notificaciones y contenidos académicos.

La arquitectura sigue un modelo cliente-servidor, donde:

- El servidor centraliza la lógica y la persistencia de datos (base de datos SQL y sistema de archivos).
- Los clientes acceden a funcionalidades remotas mediante interfaces intuitivas basadas en roles.

En resumen, hemos desarrollado una plataforma académica básica para gestionar tanto a las personas como a las asignaturas y contenidos que componen una escuela o universidad.

2. Objetivos

En primer lugar, se buscó implementar un sistema distribuido utilizando Java RMI. Esto incluye la creación de objetos distribuidos como AdministradorImpl, ProfesorImpl y AlumnoImpl, archivos .java que pueden ser invocados remotamente desde los clientes. También, la gestión adecuada del registro RMI para permitir conexiones dinámicas.

Un segundo objetivo fundamental fue el diseño de un sistema de gestión de usuarios y permisos basado en una base de datos relacional. La implementación utiliza tablas SQL para almacenar información de administradores, profesores y alumnos. Este componente es crucial ya que permite diferenciar claramente los roles y sus capacidades dentro del sistema: mientras los administradores pueden crear y asignar asignaturas, los profesores gestionan contenidos y notificaciones, y los alumnos interactúan con estos recursos según sus matrículas.

Como tercer objetivo, facilitar la interacción entre los diferentes tipos de usuarios del sistema académico. Esto incluye funcionalidades como el sistema de notificaciones (que permite comunicación basada en publicador-suscriptor, en la que el profesor escribe y los alumnos leen las notificaciones), la gestión distribuida de contenidos académicos (con capacidad para subir y descargar archivos), y el proceso de matriculación donde los alumnos pueden inscribirse en asignaturas disponibles.

El último objetivo, el proyecto incorpora mecanismos para garantizar la consistencia de datos y un manejo adecuado de errores. Esto se logra mediante el uso de transacciones SQL para operaciones críticas, la implementación de excepciones personalizadas que encapsulan posibles fallos, y validaciones de datos que previenen inconsistencias. Esto es esencial para mantener la integridad del sistema ante operaciones concurrentes y posibles fallos de comunicación.

Además, hemos desarrollado un bash script con el que se puede iniciar y gestionar todo el proyecto fácilmente y de forma centralizada.

3. Herramientas utilizadas

Lenguaje de Programación y Plataforma

- Java SE (Standard Edition):
- Java RMI (Remote Method Invocation): Tecnología central para la comunicación remota entre el servidor y los clientes, permitiendo invocar métodos entre diferentes JVMs como si fueran locales.

Base de Datos y Persistencia

- PostgreSQL:
  - Sistema de gestión de bases de datos relacional para almacenar información estructurada (usuarios, asignaturas, matrículas, etc.).
  - Consultas mediante JDBC, que permite interactuar con la BD desde Java usando sentencias SQL.
- Sistema de archivos local:
  - Para almacenar contenidos subidos por profesores.

Comunicación y Red

- Protocolo RMI:
  - Registry de RMI: Servicio que permite a los clientes localizar métodos remotos. Usa el puerto 54355 (configurable) para registrar y buscar objetos remotos.

Interfaz de Usuario

- Consola/Terminal:
  - Interfaz basada en texto para clientes (Scanner para entrada de datos).
  - Swing (opcional en algunos casos): Para diálogos simples como JOptionPane.

4. Diseño del Sistema

El diseño del sistema se basa en una arquitectura cliente-servidor distribuida, donde la lógica principal reside en el servidor y los clientes acceden a ella de forma remota mediante Java RMI. El sistema está estructurado para garantizar escalabilidad, persistencia y consistencia en los datos, y una clara separación de responsabilidades entre los componentes. A continuación, detallaremos los aspectos clave del diseño.

4.1 Arquitectura General

El sistema sigue un modelo cliente-servidor con los siguientes flujos de comunicación:

- Cliente: Interactúa con el usuario mediante una interfaz de consola o gráfica, enviando solicitudes al servidor (ej: matricularse, enviar notificaciones, etc).
- Servidor: Procesa las peticiones, accede a la base de datos y devuelve respuestas. Usa RMI para exponer métodos remotos (ej: crearAsignatura, leerNotificaciones, etc).
- Base de datos: Almacena toda la información estructurada (usuarios, asignaturas, notificaciones) y se conecta al servidor mediante JDBC.

Cabe destacar que este proyecto utiliza dos patrones de diseño que son fundamentales para su estructuración: el patrón DAO (Data Access Object) y el patrón Fábrica. El patrón DAO se implementa en las clases Impl, donde cada método accede directamente a la base de datos mediante consultas SQL con PreparedStatement. Por otro lado, el patrón Fábrica se aplica en FabricaAcademicaImpl, que centraliza la creación de objetos remotos (Administrador, Profesor, Alumno) y actúa como punto único de acceso para los clientes, simplificando la gestión de instancias. Además, para las notificaciones se usa un patrón Publicador-Suscriptor, donde los clientes de tipo Alumno se suscriben automáticamente a las asignaturas donde se matricule y escuchará las notificaciones mandadas por los respectivos profesores.

4.2 Organización de clases

El sistema está organizado en las siguientes clases, que reflejan los roles y entidades principales del sistema:

- ❖ Clases de Implementación RMI:
  - ➢ ProfesorImpl, AlumnoImpl y AdministradorImpl que implementan interfaces remotas (Profesor, Alumno, Administrador) y contienen la lógica específica de cada rol (ej: un profesor puede enviarNotificacion).
  - ➢ FabricaAcademicaImpl que centraliza la creación de estos objetos.
  - ➢ NotificacionServiceImpl que centraliza la gestión de notificaciones
- ❖ Clases de Entidades:
  - ➢ AsignaturaImpl que modela asignaturas con atributos como id, nombre y dni_profesor.
  - ➢ ContenidoImpl que gestiona los archivos.
- ❖ Conexión a BBDD:
  - ➢ La clase DBConnection proporciona conexiones reutilizables a la base de datos mediante JDBC.
- ❖ Clientes:
  - ➢ clienteAcademico que engloba a los tres tipos de clientes (administrador, profesor y alumno).
  - ➢ clienteNotificationListener que se encarga del lado cliente de las notificaciones (usado por alumnos y profesores).
- ❖ Servidor:
  - ➢ ServidorAcademico que se encarga de iniciar los servicios de notificaciones y academico.

Algunos ejemplos de relaciones importantes para el sistema:

- ❖ Tanto profesor como alumno pueden tener múltiples asignaturas asignadas.
- ❖ Un alumno se relaciona con las asignaturas a través de matrículas.
- ❖ Un administrador gestiona la relación profesor-asignatura.

4.3 Estructura de la base de datos

La base de datos está compuesta por las siguientes tablas:

Tabla	Descripción	Campos Relevantes
Asignaturas	Almacena información de asignaturas.	id (PK), nombre, dni_profesor (FK a profesores)
Contenidos	Guarda rutas de archivos subidos por profesores.	id (PK), id_asignatura (FK), path (ruta del archivo)
Notificaciones	Registra mensajes entre usuarios con marca de tiempo.	id (PK), origen (alumno/profesor), mensaje, fecha
Matriculas	Relación entre alumnos y asignaturas.	dni_alumno (FK), id_asignatura (FK)

4.4 Estructura de Directorios

La estructura del proyecto se organiza así: en la raíz encontramos el script run.sh y la carpeta libs/ con la dependencia de PostgreSQL; dentro de interfaces/ residen todas las definiciones RMI (FabricaAcademica, Administrador, Profesor, Alumno, Asignatura, Contenido, NotificacionService y sus listeners); en servidor/ están las implementaciones remotas (FabricaAcademicaImpl, AdministradorImpl, ProfesorImpl, AlumnoImpl, NotificacionServiceImpl), la clase ServidorAcademico que arranca el RMI Registry y DBConnection para el acceso JDBC, junto al subdirectorio bbdd/ con el script SQL de creación de tablas; y en clientes/ se ubican ClienteAcademico.java (que recoge la funcionalidad principal y los menús para cada rol) y el listener de notificaciones (ClienteNotificationListener.java).

5. Desarrollo del proyecto

El desarrollo del sistema se centró en implementar las funcionalidades distribuidas mediante Java RMI, siguiendo un enfoque modular basado en interfaces remotas y sus implementaciones. A continuación, se detallan los aspectos clave del desarrollo.

5.1 Interfaces RMI

Las interfaces RMI definen los métodos que pueden ser invocados de forma remota por los clientes. Estas son:

- FabricaAcademica: Es la interfaz principal que actúa como punto de entrada para los clientes. Define métodos para verificar la existencia de usuarios, crear nuevos registros y obtener instancias remotas de los roles.
- Profesor: Define las operaciones exclusivas para profesores como ver las

asignaturas, enviar notificaciones o subir contenido.

- ❖ Administrador: Define las operaciones exclusivas para administradores como la gestión de asignaturas (crear, borrar, asignar profesores) y la consulta del listado completo de asignaturas.
- ❖ Alumno: Define las operaciones exclusivas para alumnos como la matriculación en asignaturas, la descarga de contenidos, el envío de consultas a profesores y la recepción de notificaciones.
- ❖ Asignatura: Modela una asignatura con métodos para acceder a sus propiedades.
- ❖ Contenido: Permite manejar archivos binarios mediante un array de bytes.
- ❖ NotificacionListener y NotificacionService: Permiten la gestión del servicio de notificaciones entre profesores y alumnos.

5.2 Implementación del servidor

En esta parte del proyecto se desarrolló el servidor, que es el encargado de manejar todas las funciones principales del sistema y de permitir que los usuarios puedan acceder a ellas de forma remota. Básicamente, el servidor implementa diferentes interfaces según el tipo de usuario: administrador, profesor o alumno.

Una de las clases que se creó fue ProfesorImpl, que implementa los métodos necesarios para el menú del profesor. Por ejemplo, se programaron métodos como subirContenido, enviarNotificacion y verAsignaturas, aunque algunos como el de subir contenido todavía están en desarrollo. Esta clase permite que el profesor pueda hacer acciones específicas dentro del sistema.

También se crearon otras clases similares como AdministradorImpl y AlumnoImpl, que permiten realizar operaciones como crear asignaturas, matricularse, ver datos personales, leer notificaciones, entre otras cosas, según lo que muestra cada menú.

En resumen, el servidor es como el “cerebro” de la aplicación, ya que desde ahí se controlan todas las funciones y se comunica con los clientes para que cada usuario pueda hacer lo que le corresponde dentro del sistema.

5.3 Cliente

El cliente fue hecho en un solo archivo y es el que se encarga de mostrar los menús a los diferentes usuarios: administrador, profesor y alumno. Dependiendo de quién inicie sesión, se muestra un menú distinto con las opciones que le corresponden.

El programa funciona por consola, así que el usuario solo tiene que escribir el número de la opción que quiere y el cliente se conecta al servidor usando RMI para hacer esa

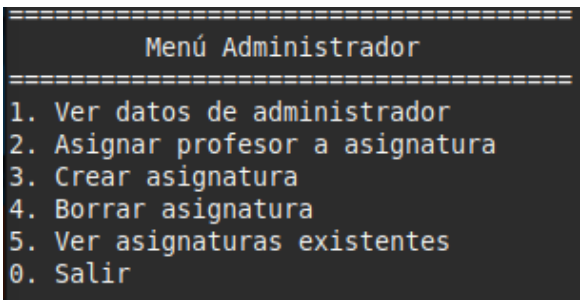
acción. Por ejemplo, si el usuario es un profesor y elige “ver asignaturas”, se llama al método remoto correspondiente en el servidor que devuelve la información.

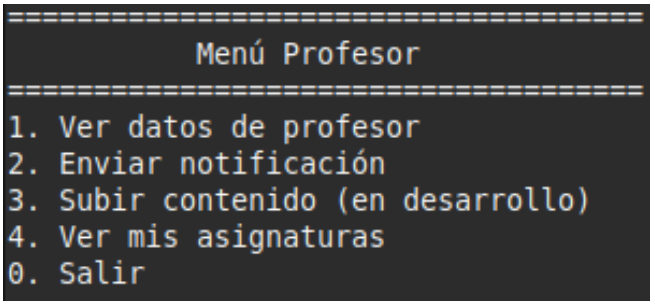
Cada menú está adaptado al tipo de usuario. El administrador puede crear y borrar asignaturas, o asignar profesores. El profesor puede enviar notificaciones o ver sus asignaturas. El alumno puede matricularse, ver sus materias o leer notificaciones. Algunas funciones todavía están en desarrollo, como subir contenido o notificar al profesor desde el lado del alumno, pero ya están puestas en el menú como indicativo.

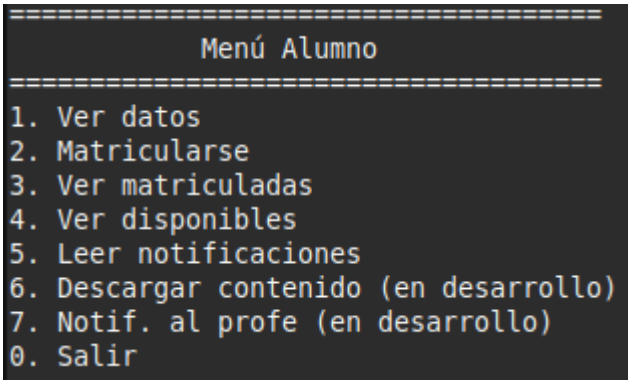
Además, existe un código cliente secundario para el sistema de notificaciones, que será usado por el programa cliente principal a demanda.

En resumen, el cliente es la parte que ve el usuario y que permite interactuar con el servidor de forma sencilla, usando menús claros y opciones por consola.

5.4 Capturas de los Menús por Rol de Usuario







6. Problemas encontrados y soluciones

Algunos problemas encontrados:

1. Duplicidad al Pedir el DNI
   Se pedía varias veces el DNI lo cual era molesto, en su lugar se pide el DNI al principio y se mantiene durante toda las sesión.

2. Problemas al Borrar Asignaturas
   Al eliminar una asignatura, no se desvincularon los profesores ni los alumnos de esa asignatura, lo que podía causar errores, para solucionarlo se han hecho consultas SQL para quitar a los alumnos de esa asignatura y dejar vacío el campo del profesor antes de borrarla.

3. Problemas con las Notificaciones
   Los alumnos no recibían notificaciones si se suscriben después de que el profesor enviará un mensaje, para solucionarlo los alumnos se suscriben automáticamente a las notificaciones al matricularse en una asignatura, asegurando que reciben los mensajes después

4. Problemas con la Interfaz Gráfica
   La interfaz gráfica de JOptionPane no siempre funcionaba bien en todos los entornos, Se optó por usar la consola para la mayor parte del programa y solo utilizar la interfaz gráfica para registrar nuevos usuarios.

5. Cambio en el tipo de contenido
   Antes se permitía subir contenido en texto, pero al ser similar a las notificaciones, se optó por permitir la subida de archivos PDF, lo cual resulta más útil para compartir material como apuntes o ejercicios, aunque esta función sigue aún en desarrollo.

7. Posibles mejoras

- Desarrollar en profundidad la opción de subir contenido para que el profesor pueda subir PDFs y los alumnos puedan descargarlos de forma remota. El código al que llegamos está comentado debido a que falta desarrollo.

- Desarrollar completamente la función de mandar mensajes al profesor de una asignatura. Esta opción también está comentada en el código debido a que falta desarrollo.
- Creación de un sistema de tutorías, de manera que los alumnos puedan reservar citas con los profesores.

8. Guia de Ejecución

Inicio

1. Iniciar sesión con usuario dit en la máquina virtual de la US. Se debe tener postgresql instalado con un usuario y contraseña dit (por defecto viene así en las máquinas virtuales de la US)

2. Descargar archivo trabajoRMI.zip en su directorio de trabajo
3. Descomprimir archivo

Ejecución Servidor:

1. Sitúese con una consola en el directorio trabajoRMI/
2. chmod +x run.sh
3. ./run.sh [--reset | -r]
   Observación: si usa la opción de reset la base de datos se formateará completamente.En caso contrario se mantendrán los datos persistidos.

Ejecucion clientes

1. Sitúese con una consola en el directorio trabajoRMI/
2. java -cp .:libs/postgresql-42.7.3.jar clientes.ClienteAcademico

9. Conclusiones

El desarrollo del sistema académico distribuido nos permitió aplicar conceptos clave de RMI, bases de datos y diseño orientado a objetos. A lo largo del proyecto, fuimos adaptando funcionalidades según las necesidades detectadas, como el cambio del sistema de notificaciones, la gestión de asignaturas o el manejo de contenido, pudimos ver cómo comunicar distintos roles del sistema (alumnos, profesores, administradores) a través de objetos remotos, lo cual facilitó la distribución de tareas entre cliente y servidor.

También aprendimos la importancia de validar bien cada parte del sistema para evitar errores, así como a mantener una estructura clara en el código para facilitar futuras modificaciones. Registrar e invocar objetos remotos correctamente, manejar excepciones en comunicaciones distribuidas y estructurar bien las interfaces para que fueran reutilizables desde el cliente. En general, el trabajo nos ayudó a reforzar tanto el desarrollo técnico como la capacidad de resolver problemas en sistemas distribuidos.