Escuela Técnica Superior de Ingeniería Informática

PAI5

MOBIFIRMA. DESARROLLO DE PILOTO PARA COMPRAS CON DISPOSITIVOS MÓVILES PARA UNA CORPORACIÓN HOTELERA INTERNACIONAL

Grupo 8:

- Carolina Cintra Fernandes Figueira
- Jule Ella Nogaj
- Scotti Manuel Scholter Marichal

MOBIFIRMA. DESARROLLO DE PILOTO PARA COMPRAS CON DISPOSITIVOS MÓVILES PARA UNA CORPORACIÓN HOTELERA INTERNACIONAL

Escuela Técnica Superior de Ingeniería Informática

Escuela Técnica Superior de Ingeniería Informática

Abstract

El objetivo de este proyecto es desarrollar una arquitectura de software en Android Studio para una cadena hotelera internacional que permita a los distintos centros hoteleros de todo el mundo realizar pedidos de material a través de dispositivos móviles. La atención se centra en garantizar que todas las solicitudes de adquisición de material hotelero a través de dispositivos móviles estén autenticadas, sean confidenciales y tengan integridad. Además, se presentarán informes mensuales sobre las solicitudes verificadas con éxito y las tendencias de los dos últimos meses al gobierno de seguridad de la información de la organización.

Los principales ejes del proyecto son el desarrollo y despliegue de una arquitectura cliente-servidor piloto para las compras electrónicas en el grupo. Se trata de implantar una arquitectura que permita autenticar los mensajes mediante firmas digitales en dispositivos Android. La transmisión a través de redes públicas debe ser autenticada, confidencial e íntegra.

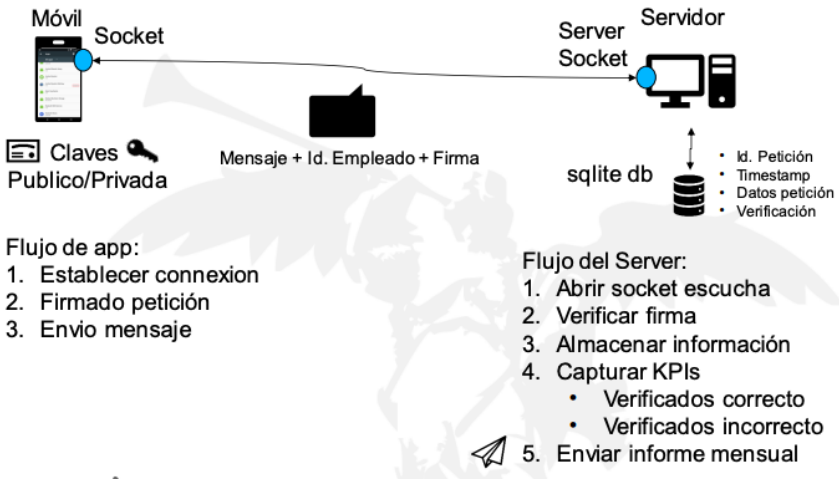
Además de las solicitudes de compra, el servidor también debe recoger información relevante para las directrices de seguridad de la organización. Esta información será utilizada por el gobierno de seguridad para controlar la autenticación incorrecta de los clientes.

La arquitectura recomendada incluye la implementación del servidor y el correspondiente gestor de base de datos, incluidas las tablas necesarias para almacenar los certificados digitales de los empleados y las solicitudes de material.

El **Manual para el uso fácil**, que recomendamos que se lea antes de que se intente ejecutar el proyecto, está colocado en el README de Github del PAI5.

Escuela Técnica Superior de Ingeniería Informática

Arquitectura



La arquitectura se compone de dos módulos: el Servidor y el Cliente. El Servidor empieza abriendo un socket que escucha hasta tener una conexión. El Cliente establece la conexión con el Servidor, si el segundo está disponible. Ambos se comunican a través de un socket, por el que el Cliente envía el mensaje, el ID del empleado y la firma cada vez que tiene una petición de compra. El Servidor, después de recibir por el socket las informaciones que necesita, verifica la firma con el uso de la clave pública que tiene almacenada en la base de datos y almacena todo el mensaje en la base de datos. El Servidor también es encargado de dar feedback sobre las verificaciones de firmas y compararlas con los últimos dos meses, de manera a tener valores para las tendencias del sistema - positiva si la ratio de los dos meses anteriores es menor o una es menor y otra igual al actual; negativa si alguna de las ratios de los dos meses anteriores es mayor al actual; nula si las ratios de los dos meses anteriores es igual al actual.

Ficheros y funciones

Módulo del Cliente:

MyApplication\app\src\main\java\com\example\myapplication\

- MainActivity.java
  - onCreate(): Esta función empieza la Activity del Cliente, genera la clave privada y llama la función responsable por enviar el mensaje

- Class StartClientTask: Clase asíncrona responsable por llamar la función que empieza la conexión con el Servidor
- Class StartMessageTask: Clase asíncrona responsable por llamar la función que envía el mensaje al Servidor
- startClient(): Crea el Socket y empieza la conexión con el Servidor
- sendStringToServer(): Envía el mensaje para el Servidor
- generatePrivateKey(): Torna la clave privada del Cliente (una String) en un objeto PrivateKey
- showDialog(): Función que trata de llamar las funciones que crean el mensaje y la envían, cuando el usuario hace click en el botón de enviar.
- extractData(): Función que hace la extracción de los datos input del cliente en la interfaz y la comprobación de los valores
- signData(message): Firma los datos del mensaje y hace encode con Base64

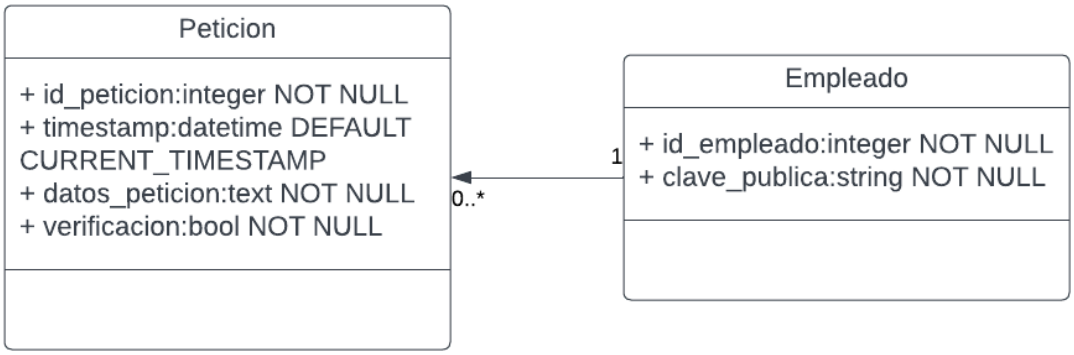
- Message.java
  - getMessage(): Retorna el mensaje en el formato '*camas* + "," + *mesas* + "," + *sillas* + "," + *sillones* + "," + *id del empleado*'

Módulo del Servidor:

Server\src\

- Main.java
  - main(): Esta función establece la conexión con el Cliente a través del socket y recibe el mensaje
  - parseMessage(message): Función responsable por dividir el mensaje de manera a tener el mensaje original, el mensaje firmado, el ID del empleado y la petición
  - generatePublicKey(): Torna la clave pública del Cliente (una String) en un objeto PublicKey
  - doVerify(message): Confirma si la firma está correcta y retorna true si sí y false si no.
- Database.java
  - Class Database: Crea la conexión entre el Servidor y la base de datos con las informaciones sobre las peticiones y empleados

La base de datos tiene la arquitectura abajo y fue desarrollado en SQLite como pedido por en el enunciado.



- ReportThread.java
  - Class ReportThread: Calcula las cuotas y tendencias de las peticiones cada minuto (mes) y facilita la creación del reporte en el fichero reporte.java.

Razonamiento

Entorno y estructura del proyecto

Optamos por trabajar, en un principio, con Android Studio para el cliente y Visual Studio Code a fin de desarrollar el servidor. Mientras que el primero fue sugerido por el enunciado, el segundo era importante para desarrollar el Servidor, que también fue realizado en lenguaje Java para obtener un proyecto cohesivo y uniforme. Más tarde descubrimos que sería necesario utilizar un IDE más complejo como IntelliJ para trabajar la conexión entre el Servidor y la base de datos. Además, era necesario adaptar el módulo Servidor para establecer dicha conexión. Java es un gran lenguaje, con mucho potencial pero su conexión con bases de datos como SQLite no es fácil. Adaptamos esa parte del proyecto para que fuera un proyecto Maven, una herramienta de software para gestionar y construir proyectos Java que requieran una arquitectura específica y dependencias que no vienen incluido en Java. Usamos la dependencia JDBC para establecer la conexión y hacer pedidos a la base de datos en “data.db”.

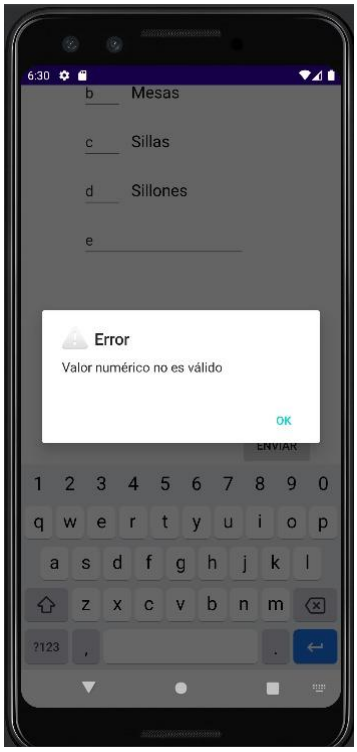
Escuela Técnica Superior de Ingeniería Informática

Usuario falso

Para crear una entrada de solicitud en la base de datos, es necesario identificar al usuario en la tabla de usuarios, de modo que las solicitudes siempre tengan un responsable. Pero en el caso de una petición con una firma falsa o un id que no existe en la base de datos, no es posible identificar (o verificar la identidad de) el usuario, pero es necesario almacenar esa petición. Para resolver este problema, creamos un usuario falso con el id 8, que es siempre el "responsable" de las solicitudes problemáticas, para que puedan almacenarse en la base de datos y utilizarse en los estudios de índices mensuales.

Validación de Entradas

El proceso de validación de los datos introducidos por el usuario de la aplicación Cliente se produce tras pulsar el botón Enviar. En este proceso se ve si los valores son numéricos y si están comprendidos entre 0 y 300. Si no lo son, se muestra un aviso al usuario con un mensaje de error y los datos no se envían al Servidor. He aquí un ejemplo del mensaje de error, que puede ser ignorado pero es propositalmente muy visible.



Pruebas

Se nos pidió crear, en el módulo Servidor, un fichero de texto generado automáticamente por el Servidor con información de la verificación de los pedidos realizados por los clientes. La idea de la propuesta era obtener el ratio de pedidos recibidos por el servidor cuya firma ha sido verificada con éxito/pedidos realizados y la tendencia mensual comparada con los dos meses anteriores. Para ello utilizamos un Thread que se encarga, cada minuto, de comparar los pedidos del último minuto con los realizados en el minuto anterior y en el minuto anterior. La recogida de datos se realiza mediante comparación de fechas en Java.

Elegimos esta estrategia para obtener los datos dentro del tiempo que tenemos para el desarrollo de la aplicación, ya que la petición era la comparación entre meses. Consideramos que demostrando que es posible comparar y tener los ratios entre minutos, lo mismo es posible para los meses, por lo que nuestra solución es suficientemente adecuada para el contexto. Abajo está un ejemplo de cómo el fichero se parece después de 10 'meses' de comunicación entre el Servidor y el Cliente.

```
Month: 1 | Year: 1 | Ratio current month: 0,333 | Tendency: 0
Month: 2 | Year: 1 | Ratio current month: 1,000 | Tendency: 0
Month: 3 | Year: 1 | Ratio current month: 0,000 | Tendency: -
Month: 4 | Year: 1 | Ratio current month: 0,800 | Tendency: -
Month: 5 | Year: 1 | Ratio current month: 1,000 | Tendency: +
Month: 6 | Year: 1 | Ratio current month: 0,667 | Tendency: -
Month: 7 | Year: 1 | Ratio current month: 0,750 | Tendency: -
Month: 8 | Year: 1 | Ratio current month: 0,000 | Tendency: -
Month: 9 | Year: 1 | Ratio current month: 0,000 | Tendency: -
Month: 10 | Year: 1 | Ratio current month: 0,000 | Tendency: 0
```

Con respecto al aspecto de la aplicación móvil, seguimos los mockups del enunciado y abajo puede encontrar un ejemplo de su uso. Además, para mejorar la experiencia del usuario de la aplicación móvil también creamos un aviso de que el mensaje será enviado, de manera que el usuario puede cambiar las informaciones antes del envío.

Escuela Técnica Superior de Ingeniería Informática

Conclusión

Se puede encontrar el proyecto en: https://github.com/caroucintra/SSII/tree/main/PAI5

Consideramos que el proyecto ha sido un éxito en llevar a cabo el proceso de comprobar que la firma que viene enviado con un mensaje de un cliente esté correcta. Todas las peticiones en las que falla el proceso de la firma, a causa de que por ejemplo sea un atacante no autorizado, estarán asignados a nuestro usuario falso (con el número de identificación 8). A través de este usuario falso podemos, como ya hemos mencionado antes, resumir cuántos peticiones han llegado o correctas o falsas a fin de cada mes. Y todo esto sin que surjan vulnerabilidades por qué no consideramos las peticiones asignadas a este usuario falso como relevantes para la empresa sino sólo para las estadísticas.

Por consiguiente, es necesario compartir la clave pública del cliente con el servidor, lo cual la guarda en su base de datos. Por otra parte y como estándar de seguridad, el cliente haría de guardar su clave privada de manera segura, como por ejemplo en un almacenamiento de claves de Java.
A propósito de evitar vulnerabilidades de inyección por medio del usuario / empleado, verificamos todas las posibles entradas.

En definitiva, reportamos las cuotas de peticiones válidas y exitosas frente a las peticiones incorrectas o maliciosas de cada mes. De esto derivamos las tendencias comparando siempre el mes actual con los dos meses anteriores según las reglas del enunciado.