Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación

Christopher Garcia (20541)
Marco Jurado (20308)
Yong Bum Park (20117)
Gabriel Vicente (20498)

Proyecto 1 – Comunicaciones Seguras

Documento de diseño

CC 3078 CIFRADO DE INFORMACIÓN
Sección 11
Catedrático: CANO FUENTES, LUDWING OTTONIEL

Índice

Arquitectura del proyecto

Al acceder al repositorio, nos encontramos con tres rutas principales claramente definidas: api, db y frontend, además del archivo docker-compose.yml ubicado en la raíz del proyecto. Este archivo .yml simplifica enormemente la creación y configuración de los contenedores necesarios para las tres partes fundamentales del desarrollo.

```yaml
services:
  db:
    build: ./db
    ports:
      - "5432:5432"
    volumes:
      - ./db/init.sql:/docker-entrypoint-initdb.d/init.sql
    environment:
      POSTGRES_USER: 'postgres'
      POSTGRES_PASSWORD: 'uvg2024'

  frontend:
    build: ./frontend
    ports:
      - '4200:4200'
    depends_on:
      - db

  api:
    build: ./api
    ports:
      - "3000:3000"
    depends_on:
      - db
```

Docker-compose

En este archivo se definen todos los pasos para la construcción de los contenedores, cada uno de ellos posee un dockerfile propio con instrucciones necesarias para que funcionen correctamente, adicional a esto también se incluyen los puertos, dependencias y variables de entorno necesarios para que la comunicación sea exitosa.

```dockerfile
FROM node:18.17.0-alpine AS builder
WORKDIR /app
COPY . .
RUN npm install
RUN npm install -g @angular/cli
COPY . .

EXPOSE 4200
CMD ["ng", "serve", "--host", "0.0.0.0", "--port", "4200"]
```

Dockerfile Frontend

```dockerfile
FROM postgres:13.0-alpine
COPY init.sql /docker-entrypoint-initdb.d/
```

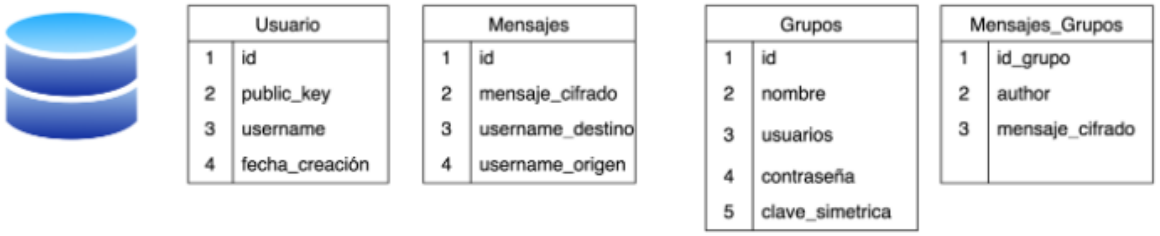
Dockerfile Base de datos

```
FROM node:18.17.0-alpine as builder
WORKDIR /app

COPY package*.json ./
RUN npm install
COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

Dockerfile Api/Backend

Estos 3 archivos son llamados desde el docker-compose para levantar sus respectivos contenedores. Para cada contenedor se manejo una estructura diferente:

- Base de datos

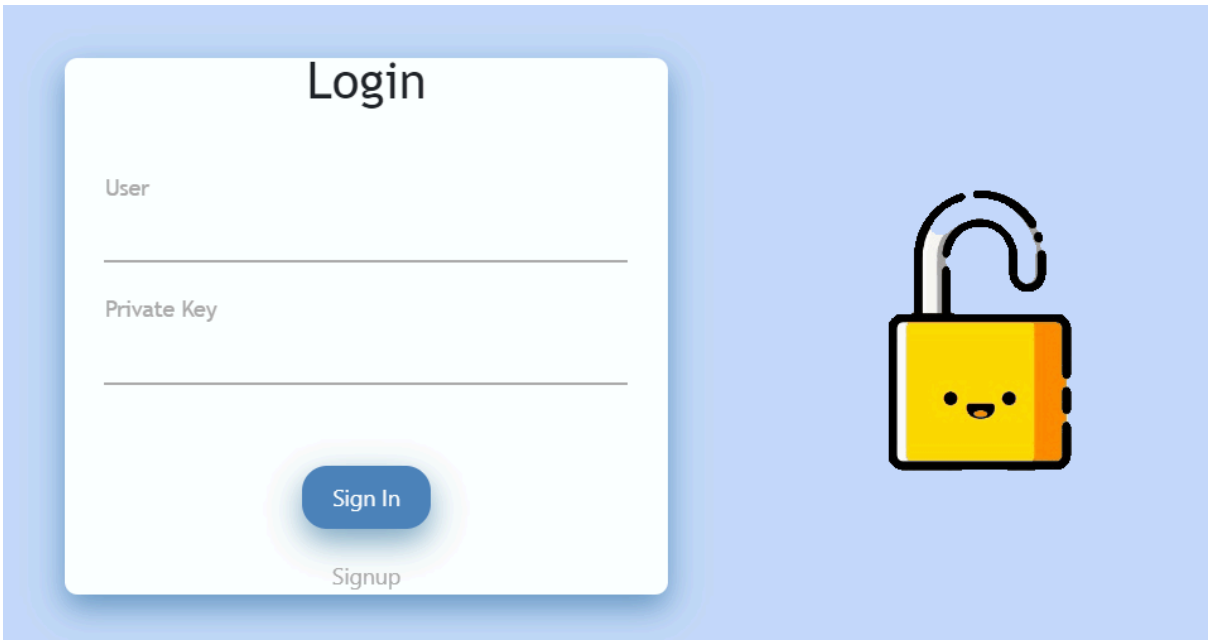


Se trabajó con la estructura proporcionada por el proyecto, al levantar el contenedor se creó una base de datos en PostgreSQL con 4 tablas (que se muestran en la imagen) y se exponía el puerto 5432 para realizar consultas a dicha base.

- Api

Para la api se trabajaron 12 métodos principales para el manejo de datos, esta api conectaba con la base de datos a través de http requests y se levantó con un proyecto de node y js. Esta api estaba compuesta de 3 archivos (index, connection y db). Connection era utilizado para ingresar los campos de contraseña, nombre de la base y host, haciendo de este proceso algo automático y que se mantenía durante toda la instancia. Db es una recopilación de métodos que operan con queries para obtener, agregar, actualizar o eliminar datos de la base de datos. Y finalmente index es el “main” donde se colocan todas las rutas de los cuales se obtendrán las respuestas de la base de datos.

- Frontend



Para el frontend se trabajó con Angular 17 y Node 18, el resultado final cuenta con 6 pantallas (login, signup, compose dm, inbox, groups, dms). Cada pantalla cuenta con sus propios componentes y métodos que conectan con el api mencionado anteriormente y se manejan los datos acorde a su tipo, por ejemplo, si son listas se busca la forma de desplegar la información de forma entendible.

Cifrado

En el desarrollo de la aplicación, se implementó un sistema de cifrado asimétrico utilizando la biblioteca node-forge. Este sistema se basa en la generación de un par de claves, una pública y una privada, utilizando el algoritmo RSA con una longitud de clave de 2048 bits y un exponente público de 0x10001. El código para generar este par de claves es let pair = Forge.pki.rsa.generateKeyPair(2048, 0x10001);. Una vez generadas las claves, se realizó un proceso de limpieza para eliminar los encabezados y los caracteres de nueva línea (\r y \n), dejando solo el texto de la clave como una cadena de caracteres.

La clave pública se almacena en la base de datos para su uso posterior, mientras que la clave privada se descarga y almacena localmente en el navegador del usuario. En particular, la clave privada se almacena en el almacenamiento local del navegador con la clave 'privateKey'. Cuando se envía un mensaje, se utiliza la clave pública del destinatario, que se recupera de la base de datos, para cifrar el contenido del mensaje. El cifrado se realiza utilizando el algoritmo RSA-OAEP, que proporciona un alto nivel de seguridad. El código para cifrar el mensaje es let encryptedMessage = publicKey.encrypt(this.messageContent, 'RSA-OAEP');.

Para desencriptar el mensaje, se utiliza la clave privada del destinatario, que se recupera del almacenamiento local del navegador. Al igual que con el cifrado, la desencriptación se realiza utilizando el algoritmo RSA-OAEP. Esto asegura que sólo el destinatario previsto, que posee la clave privada correspondiente, pueda leer el contenido del mensaje.

Para el tema de grupos, el proceso es ligeramente distinto. Para comenzar se trabajó con una llave simétrica generada con AES-128, como indica el documento del proyecto, la cual es utilizada tanto para encriptar como para desencriptar los mensajes.

Cuando se desea enviar un mensaje encriptado a un grupo, se utiliza la función encryptMessage(key, message). Esta función toma como entrada la clave AES-128 generada previamente y el mensaje en texto plano que se desea enviar. Internamente, se crea un cifrador AES en modo ECB (Electronic Codebook) utilizando la clave proporcionada. El modo ECB no utiliza un vector de inicialización (IV), por lo que se establece como una cadena vacía. Luego, el mensaje se cifra utilizando el cifrador creado y se devuelve en formato hexadecimal, este es enviado a la base de datos de forma encriptada.

Por otro lado, cuando se reciben mensajes encriptados del grupo, se utiliza la función decryptMessage(key, encryptedHex) para desencriptarlos. Esta función toma como entrada la misma clave AES-128 utilizada para cifrar el mensaje y el mensaje en formato hexadecimal. Se crea un descifrador AES en modo ECB utilizando la clave proporcionada y se inicializa con un IV vacío. Luego, el mensaje encriptado se convierte de hexadecimal a bytes y se desencripta utilizando el descifrador creado. El mensaje descifrado se devuelve como texto plano, el cual se muestra en el frontend como texto entendible (si se uso la llave correcta).