TEC.AZUAY
INSTITUTO UNIVERSITARIO

APLICACIONES SEGURAS

IMPLEMENTACIÓN Y ANÁLISIS DE TÉCNICAS DE ENCRIPTACIÓN

Criptografía de Flujo (Stream Cipher)

Prepared By :

- **Milton Avila**
- **Karla Mendez**
- **Jose Morocho**
- **Bryam Romero**
- **Luis Rey**
- **Oscar Zurita**

N6A

24/4/2025

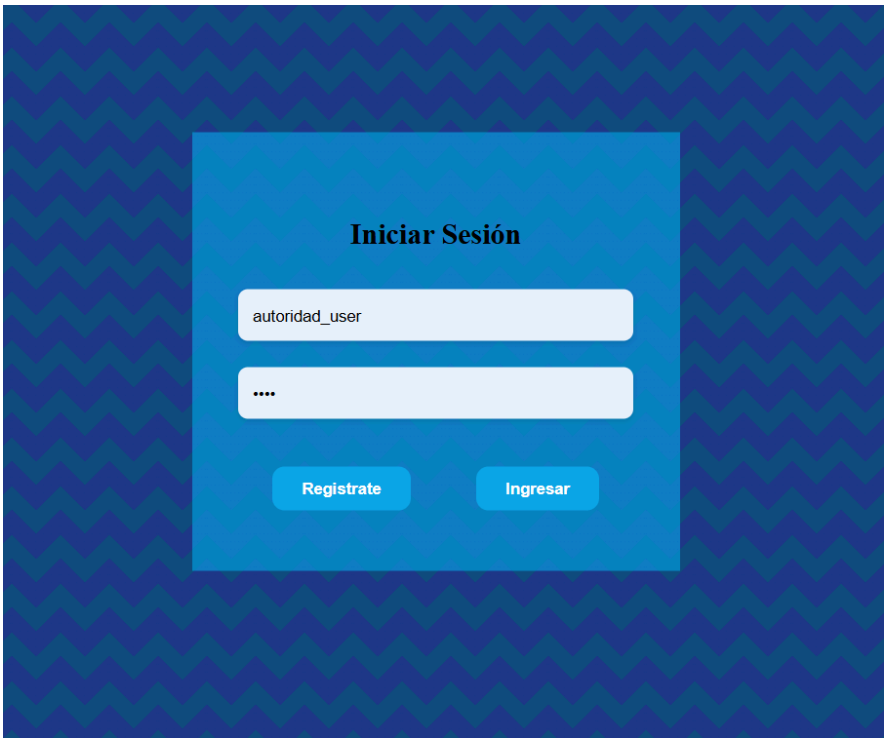
En este ejercicio implementamos una aplicación de mensajería segura utilizando el algoritmo de Criptografía de Flujo ChaCha20. Este tipo de cifrado se caracteriza por trabajar con datos en tiempo real, encriptando bit a bit o byte a byte, lo que lo hace ideal para proteger mensajes mientras son transmitidos.

El objetivo fue asegurar que los mensajes enviados entre usuarios estén cifrados antes de ser almacenados en la base de datos, y que solo puedan ser descifrados por el sistema cuando el receptor los consulta. Para lograrlo, se usó el algoritmo ChaCha20 en el backend, desarrollado con Node.js, Express y MongoDB.

Se construyó una API REST que permite:

- Registrar usuarios con autenticación mediante JWT.
- Enviar mensajes cifrados con ChaCha20.
- Consultar mensajes y descifrarlos correctamente.

Este ejercicio no solo demuestra la utilidad del cifrado de flujo en la protección de datos, sino que también refuerza buenas prácticas en la seguridad de aplicaciones, como la validación de entradas, autenticación y manejo de claves secretas.



```ts
import { Component } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { Router } from '@angular/router';  // Importar Router

@Component({
  selector: 'app-login',
  imports: [FormsModule, HttpClientModule],
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']  // Corregí 'styleUrl' a 'styleUrls'
})
export class LoginComponent {
    username: string= '';
    password: string ='';
    publicKey: string ='';

  constructor(private http: HttpClient, private router: Router) {}  // Inyectar Router

  login(): void {
    if (!this.username || !this.password) {
      console.error('El nombre de usuario y la contraseña son necesarios');
      return;
    }

    const payload = {
      username: this.username,
      password: this.password
    };

    this.http.post('http://localhost:3000/api/auth/login', payload).subscribe(
      response => {
        console.log('Inicio de sesión exitoso', response);
        this.router.navigate(['/mensaje']);   // Aqui envia al form del mensaje
      },
      error => {
        console.error('Error al ingresar', error);
```

```html
<div class="contenedor">
  <div class="contenido">
    <h2 class="iniciar_sesion">Iniciar Sesión</h2>

    <input [(ngModel)]="username" placeholder="Correo electrónico" />
    <input [(ngModel)]="password" type="password" placeholder="Contraseña" />

    <div class="botones-horizontal">
      <!--Boton de registrar-->
      <button
      (click)="registrar()"
      class="btn"
    >Registrate</button>
      <button
        (click)="login()"
        class="btn"
      >
        Ingresar
      </button>
    </div>

  </div>
</div>
```

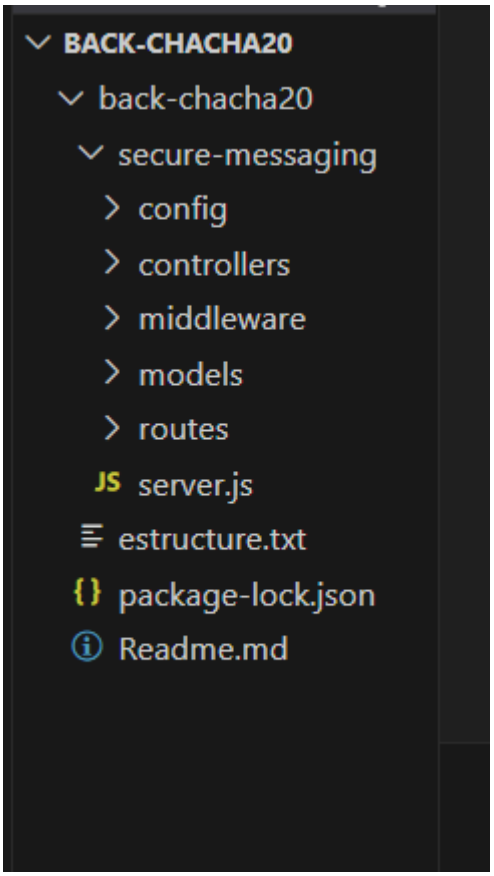
```
export class RegistroComponent {
  register(): void {
    if (!this.publicKey) {
      console.error('La clave pública es necesaria');
      return;
    }

    const payload = {
      username: this.username,
      password: this.password,
      publicKey: this.publicKey // Incluye la clave pública en el payload
    };

    this.http.post('http://localhost:3000/api/auth/register', payload).subscribe(
      response => {
        console.log('Usuario registrado', response);
        this.router.navigate(['/login']);  // Redirige al login después del registro
      },
      error => {
        console.error('Error en el registro', error);
      }
    );
  }
}
```

```typescript
import { Component, OnInit } from '@angular/core';
import { MessageService } from '../../services/message.service';
import { CommonModule } from '@angular/common';  // Importar CommonModule
import { FormsModule } from '@angular/forms';    // Importar FormsModule

@Component({
  selector: 'app-mensajes',
  standalone: true,  // Indicar que es un componente standalone
  imports: [CommonModule, FormsModule],  // Incluir CommonModule y FormsModule
  templateUrl: './mensaje.component.html',
  styleUrls: ['./mensaje.component.css']
})
export class MensajesComponent implements OnInit {
  mensaje: string = '';
  mensajes: any[] = [];

  constructor(private MessageService: MessageService) {}

  ngOnInit(): void {
    this.obtenerMensajes();
  }

  enviar(): void {
    if (!this.mensaje) return;
    this.MessageService.sendMessage(this.mensaje).subscribe(() => {
      this.mensaje = '';
      this.obtenerMensajes();
    });
  }

  obtenerMensajes(): void {
    this.MessageService.getMessages().subscribe((data: any) => {
      this.mensajes = data;
    });
  }
}
```

BACK END

Tecnologías utilizadas:

- Node.js + Express (servidor backend)
- MongoDB (almacenamiento de usuarios y mensajes)
- Mongoose (modelado de datos)
- @stablelib/chacha20poly1305 (cifrado seguro ChaCha20)
- JWT (autenticación de usuarios)
- bcryptjs (hashing de contraseñas)



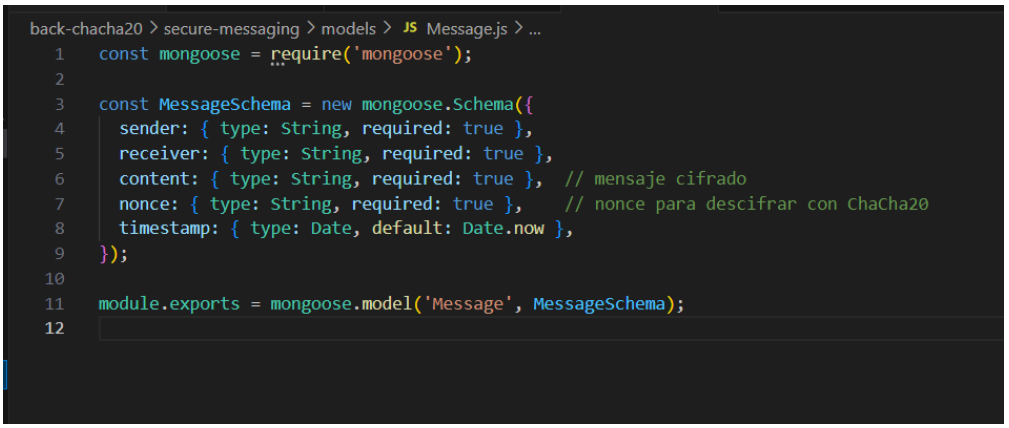
Componentes desarrollados:

1. Modelo de usuario (models/User.js)

- username: único
- password: encriptada con bcrypt
- publicKey: (campo opcional para firma/asimetría futura)

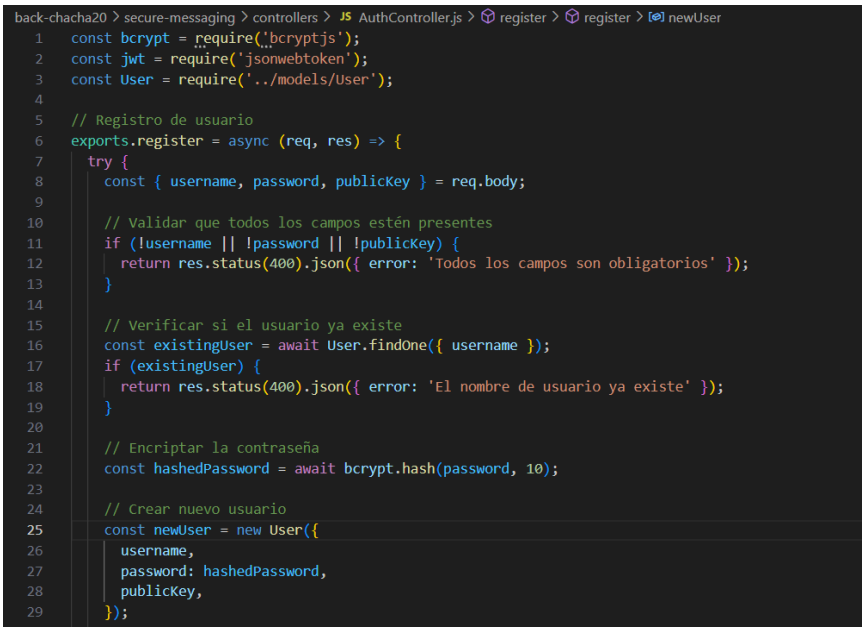
2. Modelo de mensaje (models/Message.js)

- sender: usuario emisor
- receiver: usuario receptor
- content: mensaje cifrado
- nonce: valor aleatorio necesario para descifrar
- timestamp: fecha de envío



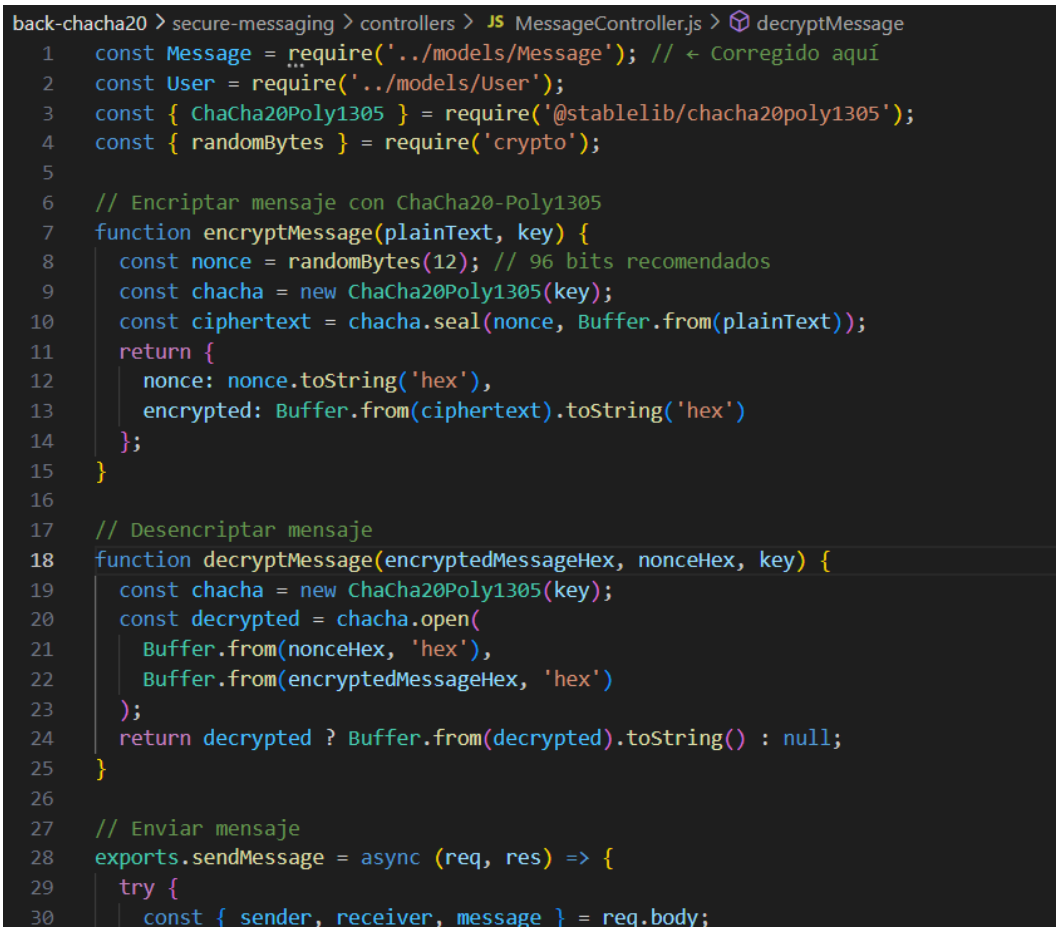
3. Controlador de autenticación (controllers/AuthController.js)

- Registro:
    - Valida que el nombre de usuario no exista
    - Cifra la contraseña con bcrypt
    - Guarda el usuario
- Login:
    - Verifica credenciales
    - Devuelve un token JWT válido si la contraseña coincide

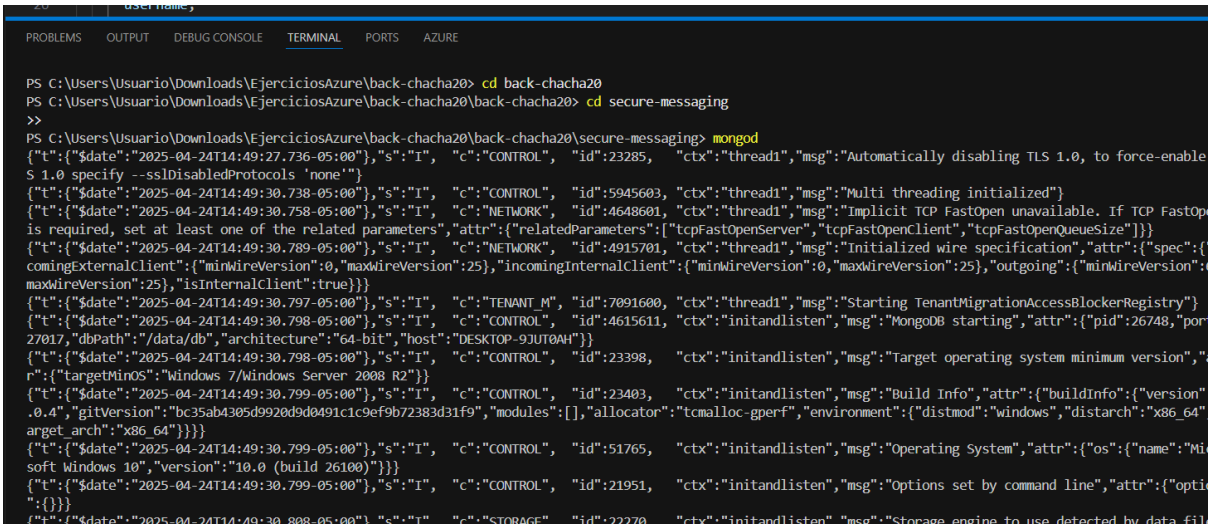


4. Controlador de mensajes (controllers/MessageController.js)

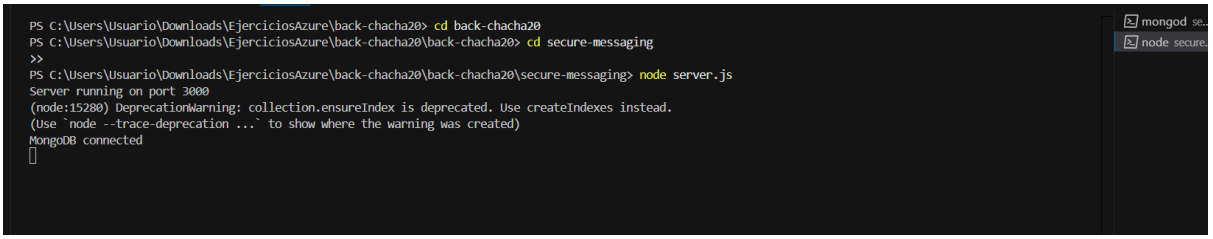
- sendMessage:
    - Verifica que el receptor exista
    - Genera nonce aleatorio
    - Cifra el mensaje con ChaCha20 usando key fija y el nonce
    - Guarda el mensaje cifrado y el nonce
- getMessages:
    - Recupera los mensajes de un usuario
    - Descifra cada mensaje con su nonce correspondiente
    - Devuelve los mensajes en texto plano al usuario autenticado



Iniciando Mongo

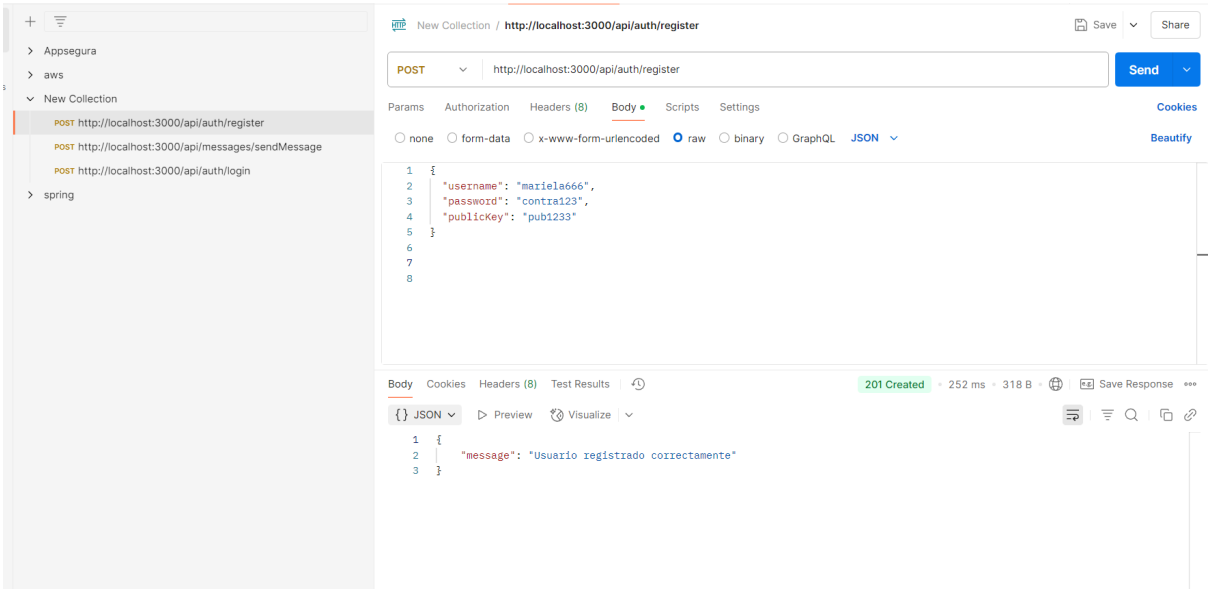


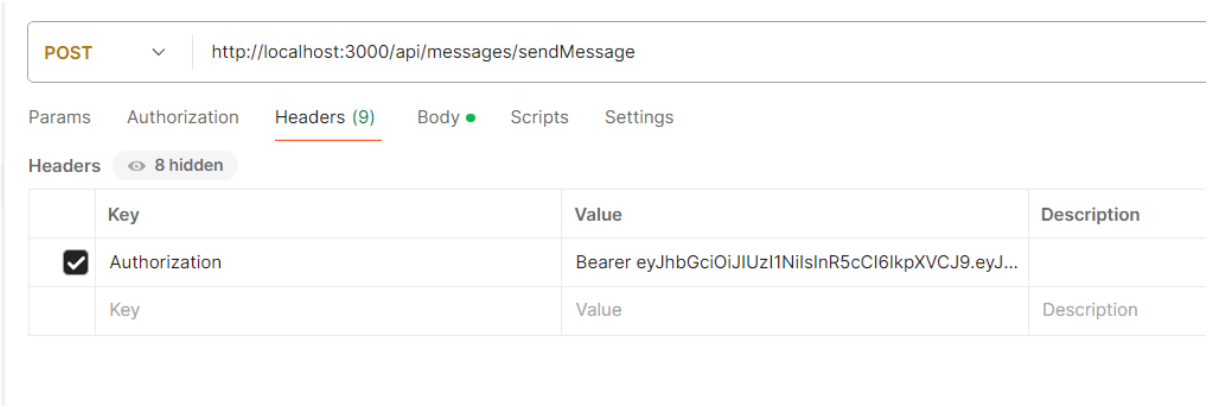
Iniciando node

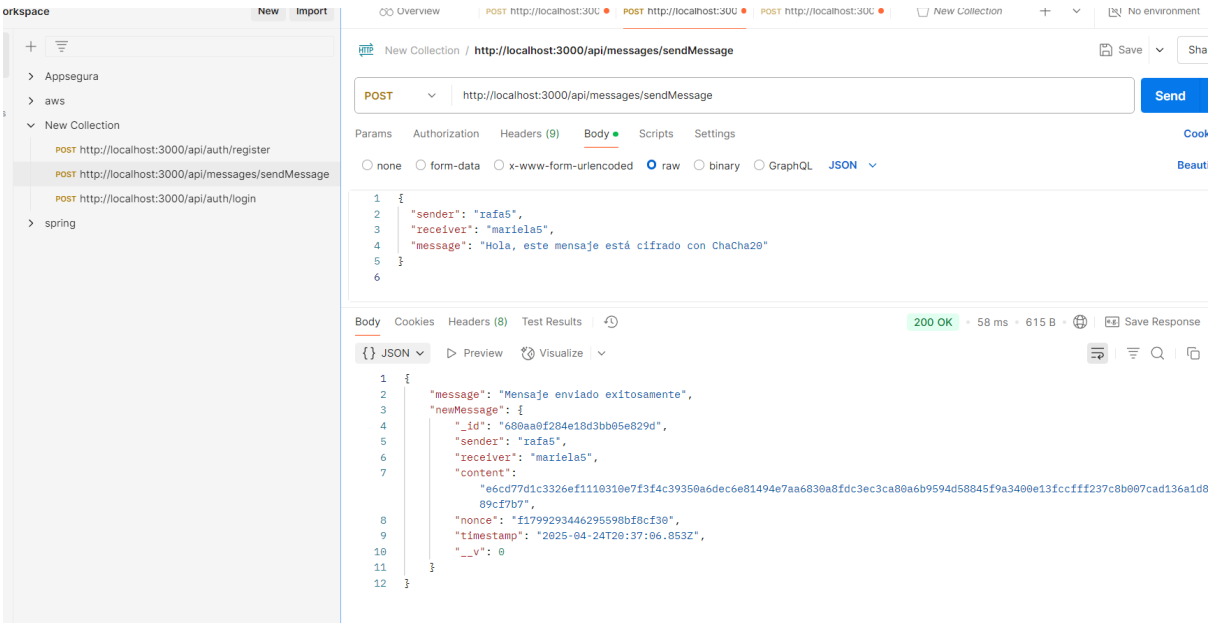


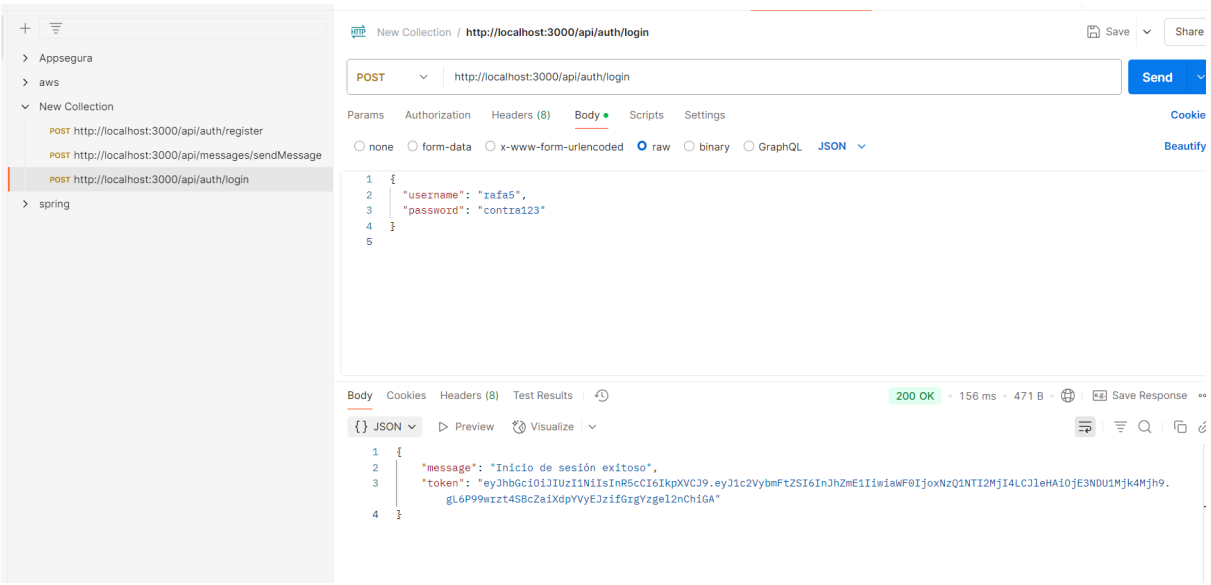
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  AZURE

PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20> cd back-chacha20
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20\back-chacha20> cd secure-messaging
>>
PS C:\Users\Usuario\Downloads\EjerciciosAzure\back-chacha20\back-chacha20\secure-messaging> node server.js
Server running on port 3000
(node:15280) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
(Use `node --trace-deprecation ...` to show where the warning was created)
MongoDB connected
```

PRUEBA DE ENDPOINTS EN POSTMAN - registro - login - mensajes




POST
http://localhost:3000/api/messages/sendMessage
Params
Authorization
Headers (9)
Body
Scripts
Settings
Headers
8 hidden
Key
Value
Description
Authorization
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...
Key
Value
Description


New Collection / http://localhost:3000/api/messages/sendMessage
Save
POST
http://localhost:3000/api/messages/sendMessage
Send
Params
Authorization
Headers (9)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"sender": "rafa5",
"receiver": "mariela5",
"message": "Hola, este mensaje está cifrado con ChaCha20"
}
Body
Cookies
Headers (8)
Test Results
200 OK
58 ms
615 B
Save Response
Preview
Visualize
{
"message": "Mensaje enviado exitosamente",
"newMessage": {
"_id": "680aa0f284e18d3bb05e829d",
"sender": "rafa5",
"receiver": "mariela5",
"content":
"e6cd77d1c3326ef1110310e7f3f4c39350a6dec6e81494e7aa6830a8fdc3ec3ca80a6b9594d58845f9a3400e13fccfff237c8b007cad136a1d8
89cf7b7",
"nonce": "f17992934462955 98bf8cf30",
"timestamp": "2025-04-24T20:37:06.853Z",
"__v": 0
}
}
Appsegura
aws
New Collection
POST http://localhost:3000/api/auth/register
POST http://localhost:3000/api/messages/sendMessage
POST http://localhost:3000/api/auth/login
spring


New Collection / http://localhost:3000/api/auth/login
Save
Share
POST
http://localhost:3000/api/auth/login
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"username": "rafa5",
"password": "contra123"
}
Body
Cookies
Headers (8)
Test Results
200 OK
156 ms
471 B
Save Response
Preview
Visualize
{
"message": "Inicio de sesión exitoso",
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InJhZmE1IiwiaWF0IjoxNzQ1NTI2MjI4LCJleHAiOjE3NDU1Mjk4Mjh9.
gL6P99wrzt4SBcZaiXdpYVyEJzifGrgYzgel2nChiGA"
}
Appsegura
aws
New Collection
POST http://localhost:3000/api/auth/register
POST http://localhost:3000/api/messages/sendMessage
POST http://localhost:3000/api/auth/login
spring

- Los mensajes no quedan en texto plano en la base de datos.
- Cada mensaje tiene su propia clave de flujo (nonce).
- Solo usuarios autenticados pueden enviar y leer mensajes.
- Técnica usada en protocolos como TLS 1.3.

LINK DE LOS REPOSITORIOS

https://github.com/RomaRomero/encript-chacha20.git

https://github.com/Oscar2876/frontappsegura.git