IPBeja
INSTITUTO POLITÉCNICO DE BEJA

INSTITUTO POLITÉCNICO DE BEJA
Escola Superior de Tecnologia e Gestão
Licenciatura em Engenharia Informática
Projeto Final de Curso

Desenvolvimento de um sistema de comunicações seguras

Martinho José Novo Caeiro - 23917
Paulo António Tavares Abade - 23919
Rafael Conceição Narciso - 24473

IPBeja Escola Superior de Tecnologia e Gestão

Beja, setembro de 2025

INSTITUTO POLITÉCNICO DE BEJA
Escola Superior de Tecnologia e Gestão
Licenciatura em Engenharia Informática
Projeto Final de Curso

Desenvolvimento de um sistema de comunicações seguras

Martinho José Novo Caeiro - 23917
Paulo António Tavares Abade - 23919
Rafael Conceição Narciso - 24473

Orientador: Professor Rui Miguel Soares Silva

Beja, setembro de 2025

Resumo

Neste relatório apresenta-se o processo de conceção e implementação de uma solução orientada para comunicações seguras, com enfoque na troca de mensagens de texto entre utilizadores, garantindo a proteção e confidencialidade da informação transmitida. O trabalho descrito foi desenvolvido no âmbito da Unidade Curricular de Estágio ou Projeto (IPBeja, 2025), tendo como objetivo principal a aplicação prática de conhecimentos adquiridos ao longo do curso, bem como a exploração de novas competências nas áreas de cibersegurança, redes e protocolos de comunicação.

Ao longo do relatório são descritos os principais desafios enfrentados, entre os quais a implementação de uma VPN com recurso ao WireGuard, a integração entre diferentes sistemas operativos (Android e Windows) e linguagens de programação (C# e Kotlin), bem como o desenvolvimento da camada de criptografia. São ainda analisadas as aprendizagens obtidas, tanto a nível técnico como no trabalho em equipa, apresentando-se no final uma reflexão sobre os resultados alcançados e as possíveis evoluções futuras da solução desenvolvida.

Keywords: aplicações, cibersegurança, comunicações, criptografia, c#, python, bash, almalinux, wireguard, sqlite, kotlin

Abstract

This report presents the process of designing and implementing a solution for secure communications, with a particular focus on enabling text message exchange between users while ensuring data protection and confidentiality. The work described was carried out within the scope of the Curricular Unit of Internship or Project (IPBeja, 2025), with the main objective of applying the knowledge acquired throughout the course, as well as exploring new skills in the areas of cybersecurity, networks, and communication protocols.

Throughout the report, the main challenges are discussed, including the implementation of a VPN using WireGuard, the integration across different operating systems (Android and Windows) and programming languages (C# and Kotlin), as well as the development of the encryption layer. The document also highlights the technical and collaborative skills strengthened during the project and concludes with a reflection on the results achieved, as well as potential future improvements and applications of the proposed solution.

Keywords: applications, cybersecurity, communications, cryptography, c#, python, bash, almalinux, wireguard, sqlite, kotlin

Índice

Índice de Figuras

1 Introdução

Para a realização do projeto foi necessário desenvolver um sistema de comunicações seguras, que permitisse a troca de mensagens de texto entre utilizadores. Para tal, tornou-se essencial implementar um sistema de autenticação de utilizadores, que possibilitasse a criação de contas e a respetiva validação de acesso. O sistema deveria garantir a confidencialidade, integridade e autenticidade das mensagens trocadas. As tecnologias utilizadas no desenvolvimento deste projeto foram: WireGuard (WireGuard, 2025), SQLite (SQLite, 2025), C# (Microsoft, 2025), Kotlin (JetBrains, 2025), Python (Foundation, 2025), Bash (Bash, 2025) e AlmaOS (AlmaOS, 2025).

No que respeita à criptografia, foram adotados os algoritmos Rijndael (vencedor do AES) e Serpent (segundo classificado no AES), ambos algoritmos de criptografia simétrica. Para a gestão do projeto, optou-se por uma adaptação da metodologia *SCRUM*, com reuniões semanais no início e, posteriormente, com maior espaçamento temporal, de forma a permitir períodos mais alargados dedicados ao desenvolvimento das tarefas. Os três membros da equipa foram distribuídos pelas principais áreas do projeto: desenvolvimento da aplicação Windows, desenvolvimento da aplicação Android e configuração da infraestrutura de rede. O cronograma do projeto foi estruturado em duas fases principais: Análise de Requisitos e Desenvolvimento, conforme ilustrado na figura 1. O relatório foi atualizado semanalmente, incorporando novas informações e registos de progresso.

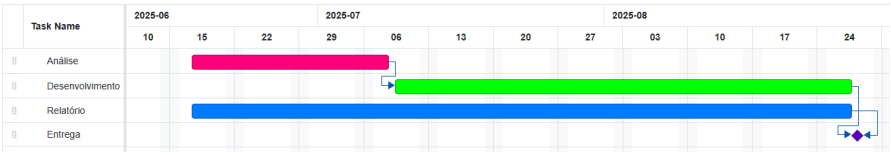


Figura 1: Cronograma do Projeto

O projeto foi armazenado nos repositórios GitHub da equipa, podendo ser consultado por outros alunos e docentes através dos seguintes endereços: https://github.com/MartinhoCaeiro/Projeto-Cripto & https://github.com/NarcisoUNK/ProjectFinal_ANDROID.

2 Análise de Requisitos

Para a realização deste projeto, a equipa iniciou o trabalho com a idealização da solução e a análise de outras aplicações de troca de mensagens, como o *WhatsApp* (WhatsApp Inc., 2025) e o *Telegram* (Telegram Messenger LLP, 2025), de forma a compreender o seu funcionamento. Constatou-se que estas plataformas recorrem a encriptação ponta a ponta para assegurar a proteção das mensagens. Neste projeto, será seguida uma abordagem semelhante, mas complementada com a integração de uma VPN. A ideia da utilização da VPN surgiu de uma discussão entre os membros da equipa, onde se considerou o impacto de encaminhar o tráfego da aplicação através desta tecnologia. Concluiu-se que tal integração acrescentaria uma camada adicional de segurança. Deste modo, foram definidos os seguintes requisitos funcionais e não funcionais:

- **Requisitos Funcionais:**
  - O sistema deve permitir a criação de contas de utilizador.
  - O sistema deve permitir a autenticação de utilizadores.
  - O sistema deve permitir o envio e a receção de mensagens entre utilizadores.
  - O sistema deve garantir a confidencialidade, integridade e autenticidade das mensagens trocadas.
- **Requisitos Não Funcionais:**
  - O sistema deve ser seguro e resistente a ataques.
  - O sistema deve ser de fácil utilização e intuitivo.
  - O sistema deve ser escalável e capaz de suportar um elevado número de utilizadores.

3 Diagrama de Casos de Uso

O diagrama de casos de uso apresentado na Figura 2 fornece uma visão global das funcionalidades principais do sistema e das interações entre os utilizadores e o servidor. Nele, é possível identificar os atores envolvidos — nomeadamente os utilizadores finais e o servidor — bem como os casos de uso associados a cada ator.

Entre os casos de uso representados, destacam-se:

- Criação e autenticação de contas de utilizador;
- Envio e receção de mensagens de texto, garantindo a confidencialidade através de encriptação;
- Gestão de chats, incluindo a criação de novas conversas e a adição de participantes;
- Configuração do serviço WireGuard, incluindo o download do ficheiro de configuração necessário para o estabelecimento da VPN;

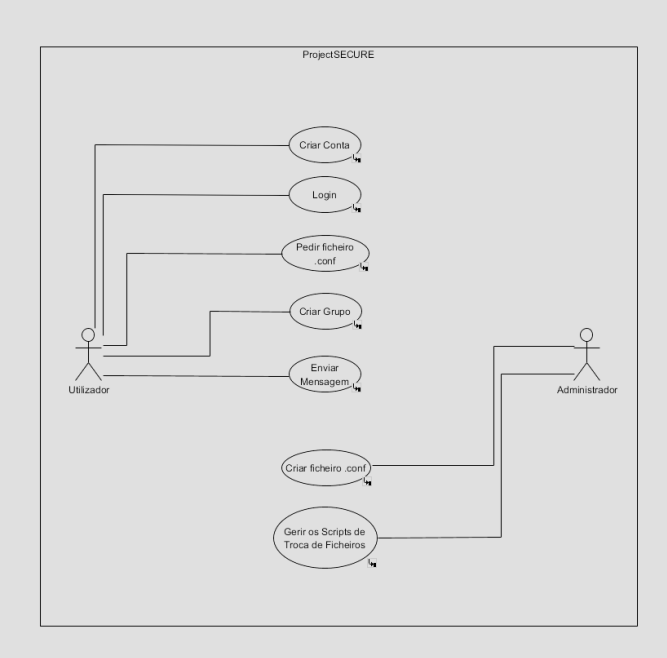


Figura 2: Diagrama de Casos de Uso

Este diagrama serve como um ponto de referência para compreender o comportamento esperado do sistema e a forma como os utilizadores interagem com as diferentes funcionalidades, complementando as descrições apresentadas na secção de requisitos funcionais e não funcionais.

4 Diagramas de Sequência

Nesta secção serão mostrados os diagramas de sequência que ilustram o fluxo de interações para os principais casos de uso do sistema.

4.1 Criação de Conta

O diagrama de sequência da Figura 3 descreve o processo de criação de uma nova conta de utilizador, onde o utilizador fornece as suas credenciais e solicita a criação da conta. O servidor valida as informações e, em caso de sucesso, confirma a criação da conta. É importante notar que a autenticação só é possível quando a ligação ao WireGuard está ativa.

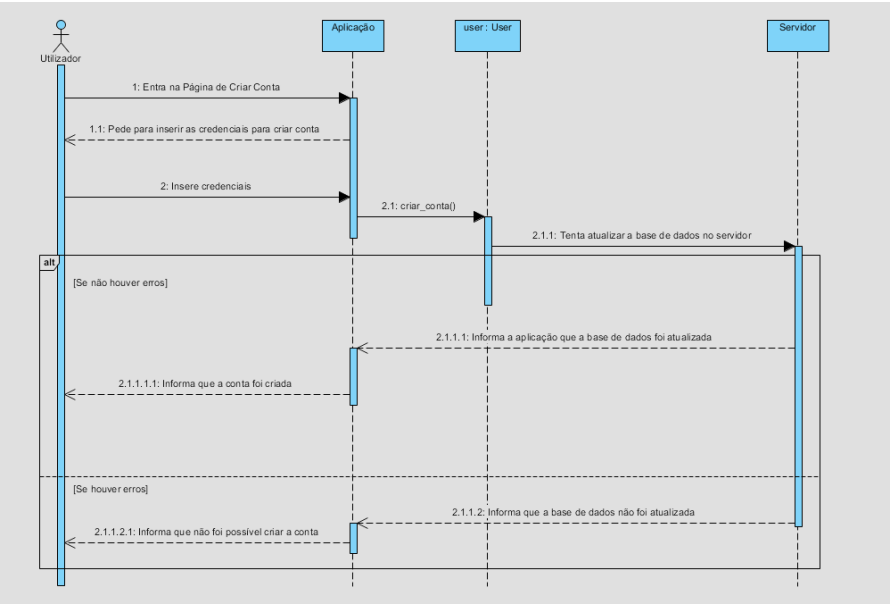


Figura 3: Diagrama de Sequência - Criação de Conta

4.2 Autenticação de Utilizador

O diagrama de sequência da Figura 4 ilustra o processo de autenticação de um utilizador existente, onde o utilizador fornece as suas credenciais e solicita a autenticação. O servidor valida as informações e, em caso de sucesso, confirma a autenticação do utilizador. É importante notar que a autenticação só é possível quando a ligação ao WireGuard está ativa.

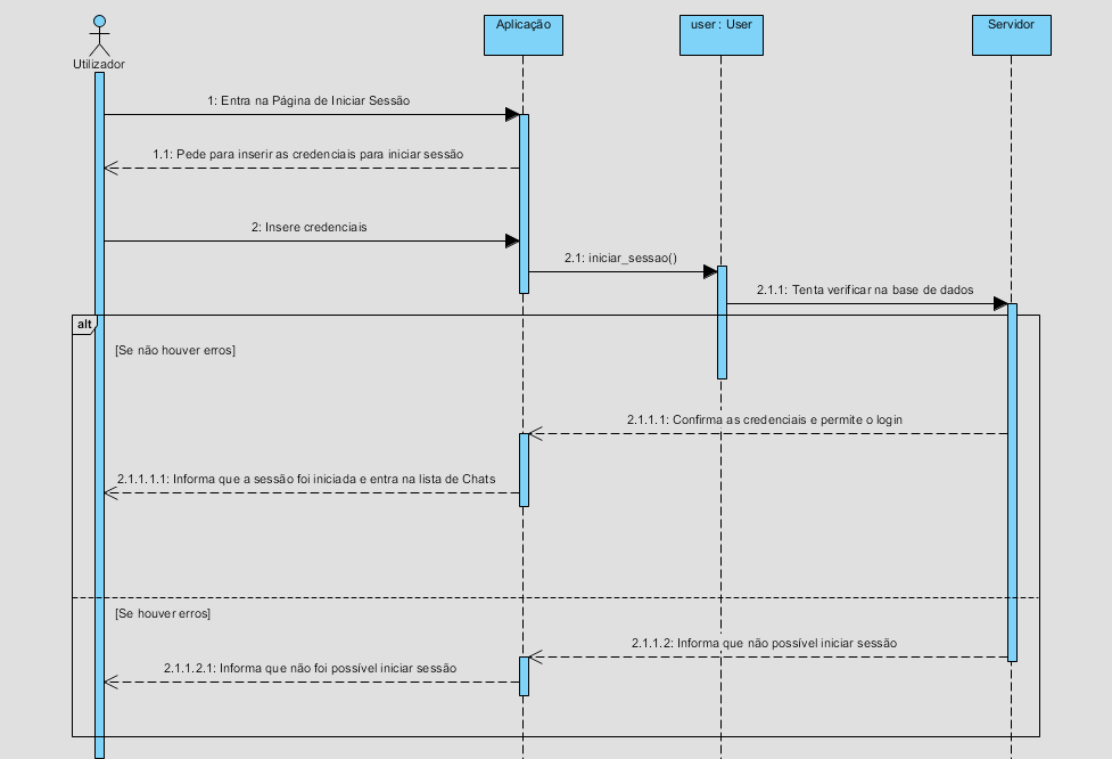


Figura 4: Diagrama de Sequência - Autenticação de Utilizador

4.3 Pedido do ficheiro de Configuração do WireGuard

O diagrama de sequência da Figura 5 descreve o processo de pedido do ficheiro de configuração do WireGuard, onde o utilizador solicita o ficheiro e o servidor responde com o conteúdo do mesmo. É importante notar que o pedido do ficheiro de configuração só é possível quando a ligação ao WireGuard está ativa e a sessão foi iniciada.

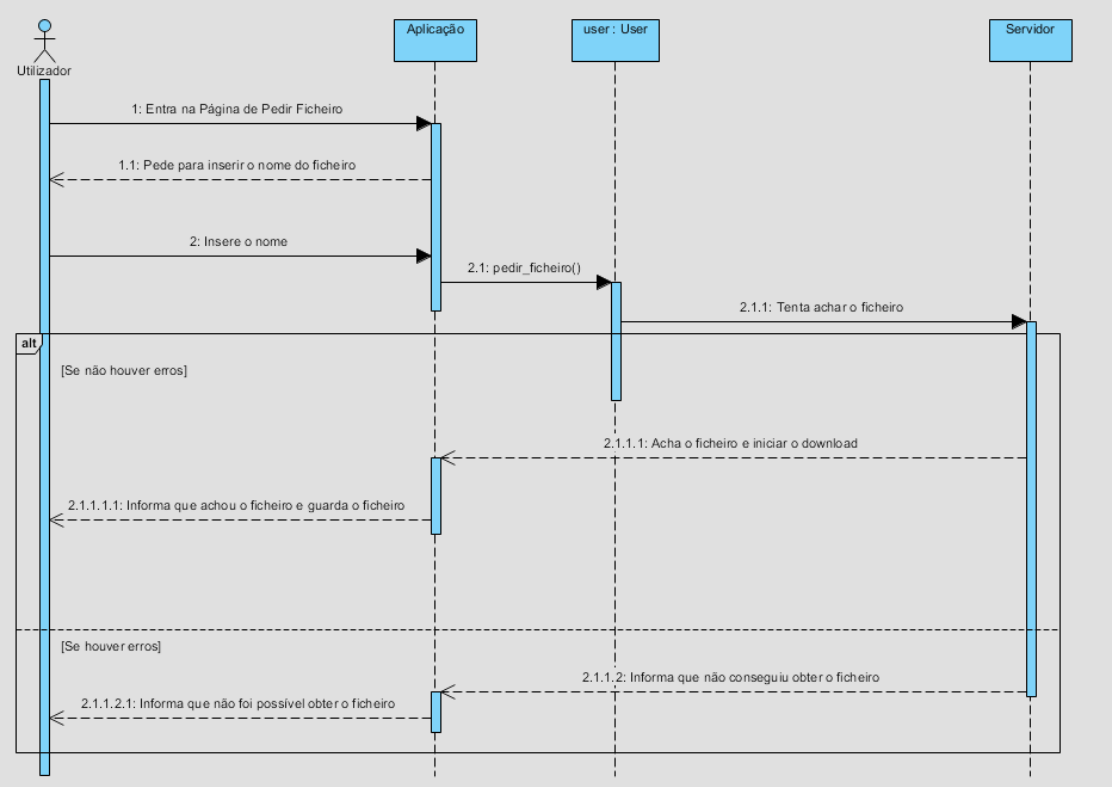


Figura 5: Diagrama de Sequência - Pedido do ficheiro de Configuração do WireGuard

4.4 Criação do Chat

O diagrama de sequência da Figura 6 descreve o processo de criação de um novo chat, onde o utilizador fornece o nome do chat e os participantes, e solicita a criação do chat. O servidor valida as informações e, em caso de sucesso, confirma a criação do chat. É importante notar que a criação do chat só é possível quando a ligação ao WireGuard está ativa e a sessão foi iniciada.

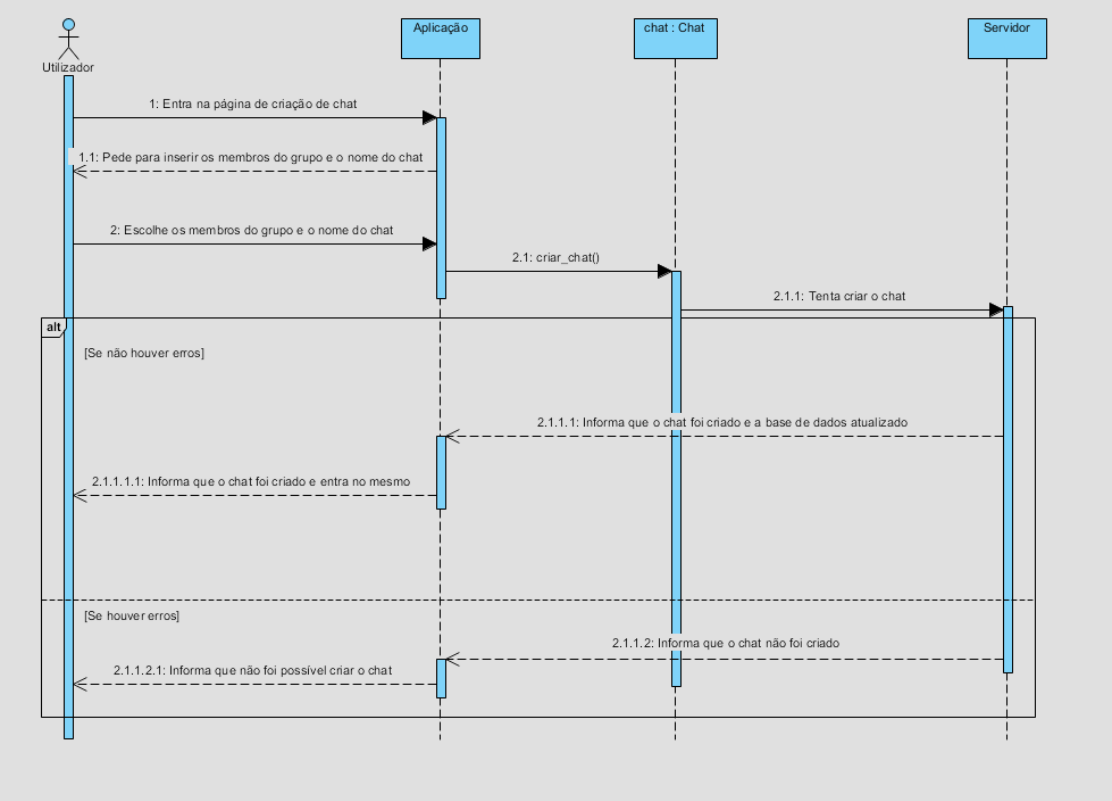


Figura 6: Diagrama de Sequência - Criação do Chat

4.5 Envio de Mensagem

O diagrama de sequência da Figura 7 ilustra o processo de envio de uma mensagem, onde o utilizador fornece o conteúdo da mensagem e solicita o envio. A mensagem é encriptada e enviada ao servidor, que a encaminha para o destinatário. É importante notar que o envio de mensagens só é possível quando a ligação ao WireGuard está ativa e a sessão foi iniciada.

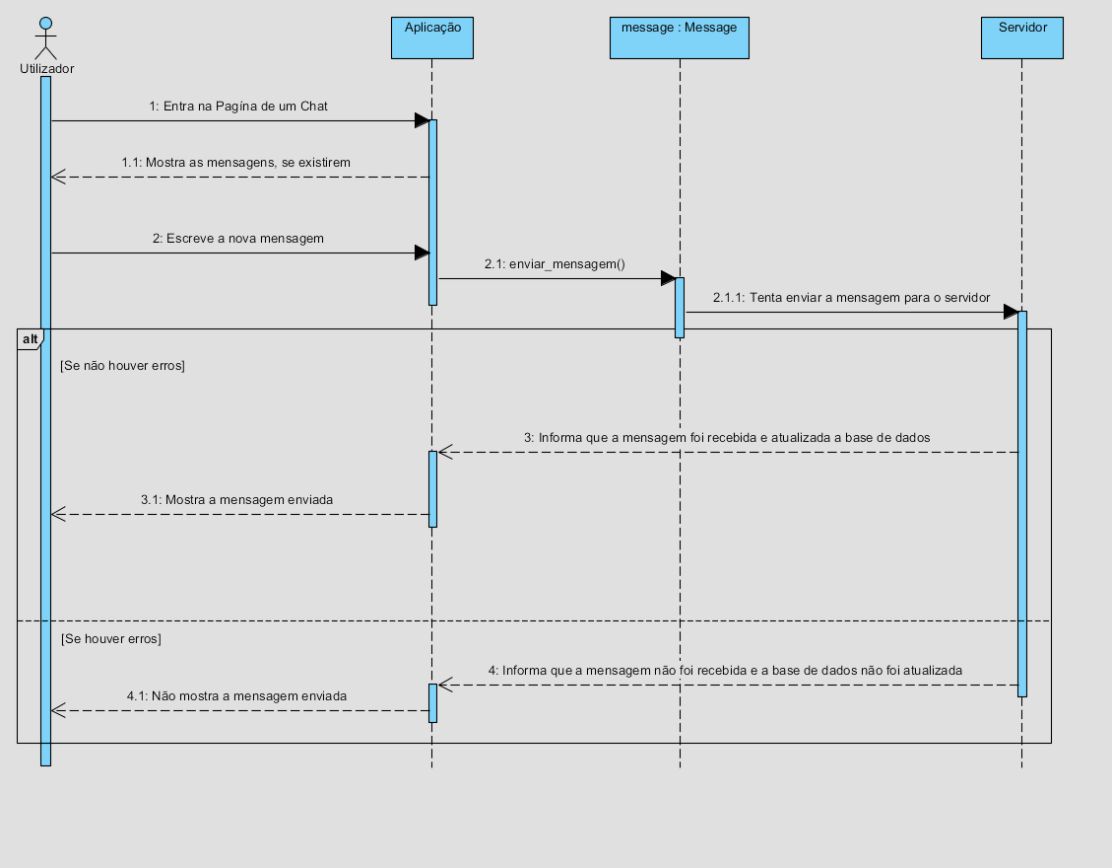


Figura 7: Diagrama de Sequência - Envio de Mensagem

4.6 Criação do Ficheiro de Configuração do WireGuard

O diagrama de sequência da Figura 8 descreve o processo de criação do ficheiro de configuração do WireGuard, onde o administrador do sistema, com acesso ao servidor, executa um script em Bash que gera o ficheiro de configuração. Este é bastante simples, porém é importante para facilitar a configuração do WireGuard para novos utilizadores. Esta ação é realizada diretamente no servidor, sem o auxílio da aplicação desenvolvida, mas através do *PuTTy*, que permite a ligação remota ao servidor.

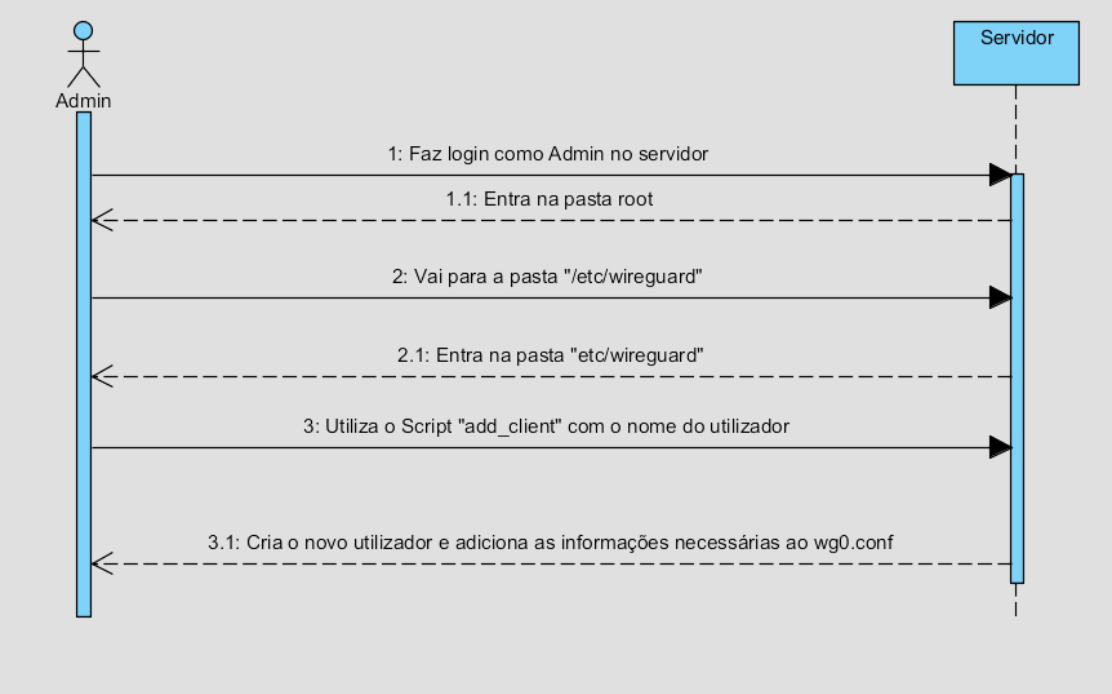


Figura 8: Diagrama de Sequência - Criação do Ficheiro de Configuração do WireGuard

4.7 Gerir os Scripts de Trocas de Ficheiros

O diagrama de sequência da Figura 9 descreve o processo de gestão dos scripts de trocas de ficheiros, onde o administrador do sistema, com acesso ao servidor, executa dois scripts feitos em *Python* que permite o download e upload de ficheiros, no caso da troca de mensagens, enquanto que o outro e relacionado com a figura 8, onde é este que permite o download do ficheiro de configuração pelo utilizador. Esta ação é realizada também diretamente no servidor, sem o auxílio da aplicação desenvolvida, mas através do *PuTTy*.

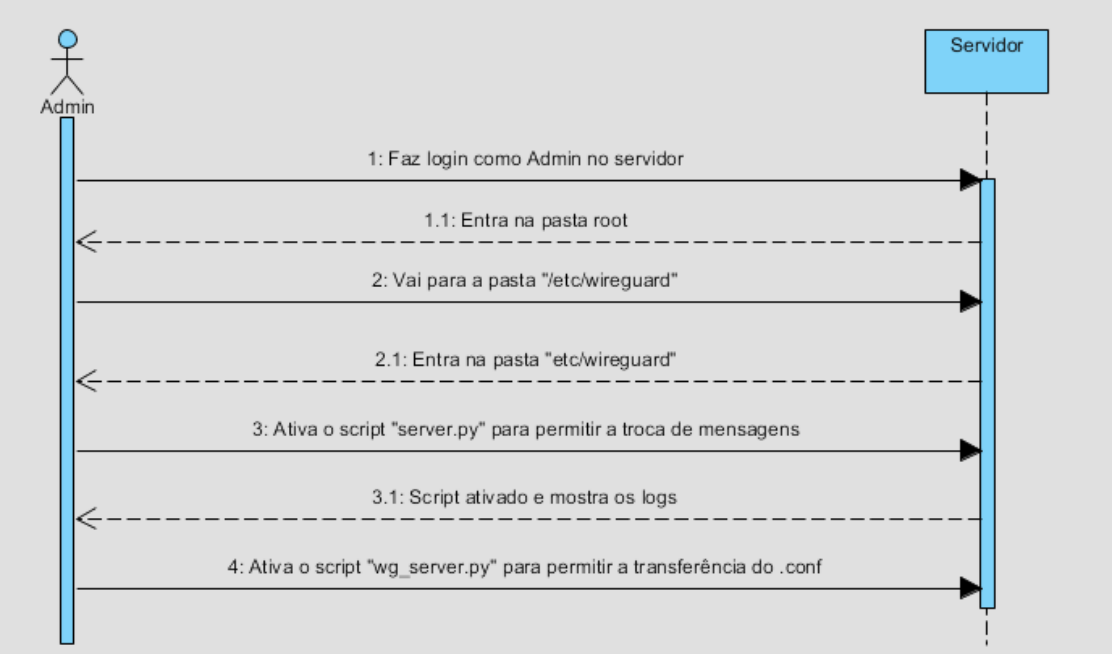


Figura 9: Diagrama de Sequência - Gerir os Scripts de Trocas de Ficheiros

5 Tecnologias Utilizadas

Para o desenvolvimento deste projeto, foram utilizadas as seguintes tecnologias:

- **Linguagens de Programação:** C#, Kotlin, Python, Bash
  - **C#:** Utilizada para o desenvolvimento da aplicação de desktop (Windows). Embora tivesse sido considerada a possibilidade de recorrer a Java, optou-se por C# por já ter sido utilizada recentemente noutra unidade curricular.
  - **Kotlin:** Utilizada para o desenvolvimento da aplicação Android. Apesar de também ter sido ponderado o uso de Java, escolheu-se Kotlin por se tratar de uma linguagem mais moderna e concisa, além de ter sido igualmente abordada noutra unidade curricular.
  - **Python:** Utilizada para a implementação de scripts de apoio, como a criação automática de ficheiros de configuração do WireGuard, sendo uma linguagem que já tem um vasto histórico de utilização pela comunidade para estes fins.
  - **Bash:** Utilizada para a automação de tarefas no sistema operativo, como a configuração de ficheiros do WireGuard no servidor, sendo a melhor opção para isso.
- **Frameworks:** .NET, WireGuard
  - **.NET:** Utilizada para o desenvolvimento da aplicação de desktop (Windows), tendo sido selecionada por já ter sido aplicada previamente noutra unidade curricular.
  - **WireGuard:** Utilizada para a implementação da VPN, por se tratar de uma solução moderna, leve e de elevado desempenho.
- **Base de Dados:** SQLite
  - Selecionada por ser um sistema de gestão de base de dados relacional leve, rápido e de fácil utilização.

- **Criptografia:** Rijndael (vencedor do AES), Serpent (segundo classificado no AES)
  - Optou-se por estes algoritmos de criptografia simétrica por serem considerados seguros e eficientes, além de serem amplamente utilizados em aplicações de segurança. A escolha foi ainda reforçada pela recomendação do professor orientador.
- **Ferramentas de Desenvolvimento:** Git, Visual Studio Code, Android Studio, GitHub
  - Utilizadas por todos os membros da equipa, por se tratarem de ferramentas modernas, eficientes e de fácil utilização.

6 Arquitetura do Sistema

O sistema é composto por uma aplicação que funcionará como um chat, onde os utilizadores registados na VPN poderão comunicar entre si. É importante notar que cada mensagem enviada corresponde ao ficheiro completo da base de dados SQLite, encapsulado e encriptado, e não apenas às mensagens recentes ou alterações individuais. Esta abordagem garante a consistência dos dados entre cliente e servidor, embora aumente o volume de dados. Dentro da aplicação, é possível identificar quais os utilizadores associados a um determinado endereço IP da VPN e, dessa forma, estabelecer comunicação através do envio de mensagens. Quando uma mensagem é enviada, a aplicação contacta o servidor da VPN para obter o caminho até ao destinatário e proceder à entrega. Antes da transmissão, a mensagem é encriptada, garantindo assim a sua confidencialidade e integridade. A encriptação utilizada varia de acordo com uma lógica pré-definida na aplicação, recorrendo-se aos algoritmos AES (Rijndael) e Serpent.

6.1 Estrutura

A estrutura do sistema centra-se na troca de informação através da VPN, podendo ser representada pela figura 10:

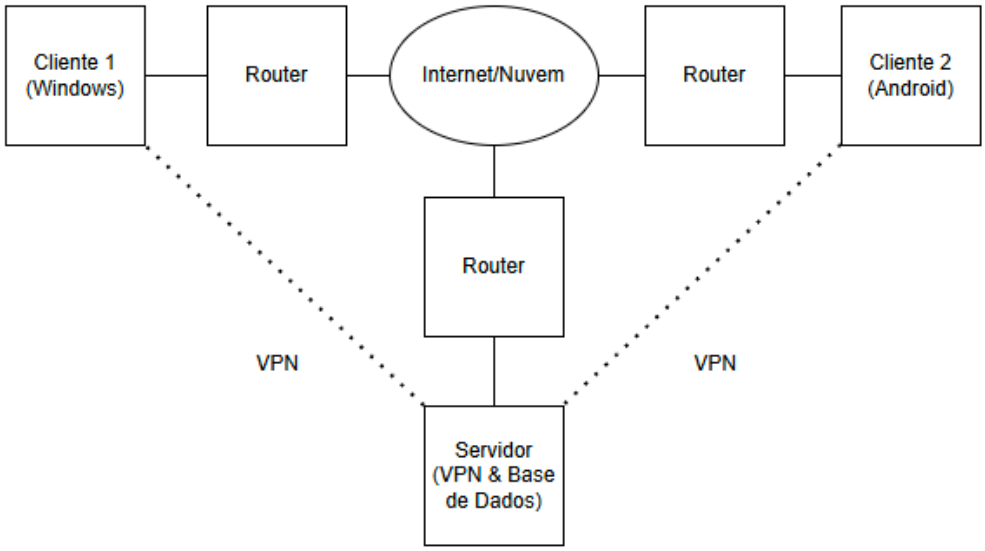


Figura 10: Arquitetura do Sistema

Os utilizadores comunicam entre si através da ligação VPN estabelecida com o servidor WireGuard, o que garante uma camada adicional de segurança, além da criptografia. No entanto, na primeira vez que o utilizador acede À aplicação, é necessário fazer o download do ficheiro de configuração por fora da VPN, uma vez que a ligação ainda não está estabelecida. Após o download, o utilizador pode iniciar a ligação VPN e, posteriormente, autenticar-se na aplicação.

6.2 Encriptação

O processo de encriptação das mensagens encontra-se ilustrado na figura 11. A mensagem é encriptada com base num número pseudo-aleatório gerado pela aplicação, o qual determina o algoritmo de encriptação a utilizar nesse momento, podendo ser AES-256 ou Serpent.

Cada operação de encriptação corresponde à transmissão do ficheiro completo da base de dados SQLite. Mesmo que apenas pequenas alterações sejam realizadas na BD, o sistema envia o ficheiro integral, garantindo a sincronização total e evitando inconsistências entre clientes e servidor.

Para a encriptação, é igualmente necessária uma chave, que nesta fase corresponde a "***Spartacus***". Importa salientar que o seletor se encontra ofuscado, de modo a dificultar a sua identificação, recorrendo a uma máscara gerada a partir da chave (*salt*) e do número pseudo-aleatório.

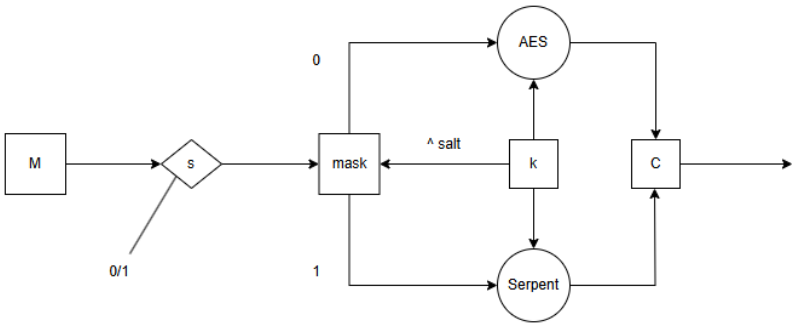


Figura 11: Processo de Encriptação das Mensagens

O processo de desencriptação corresponde à operação inversa. Para tal, o número pseudo-aleatório é anexado à mensagem encriptada, permitindo identificar qual o algoritmo de encriptação utilizado. Este processo encontra-se representado na figura 12.

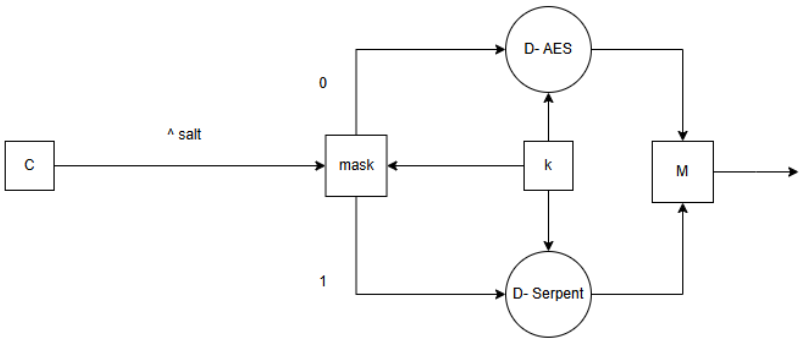


Figura 12: Processo de Decriptação das Mensagens

6.3 Implementação da Criptografia

O projeto inclui um módulo denominado `DbCrypto` (implementado em Kotlin/Java), responsável por encapsular os processos de encriptação e desencriptação dos conteúdos armazenados. Apresenta-se de seguida um resumo explicativo com os pontos-chave, acompanhado de um excerto do código mais relevante.

Resumo da lógica

- Para cada encriptação, é gerado um **salt** e um **nonce** aleatórios.
- A partir da *passphrase*/*masterKey*, deriva-se uma chave simétrica de 256 bits utilizando PBKDF2-SHA256 (100k iterações).
- É calculada uma máscara (*mask*) para ofuscar o algoritmo utilizado (AES ou Serpent), sendo o valor ofuscado armazenado no cabeçalho.
- Recorre-se a AES-GCM ou Serpent-GCM (via BouncyCastle), com AAD a incluir o cabeçalho (magic, versão, algoritmo ofuscado, número de iterações, salt e nonce).

- O ficheiro resultante apresenta a seguinte estrutura: cabeçalho || *ciphertext* || *tag*.

Trecho de código

```
private fun encryptInternal(data: ByteArray, masterKey: ByteArray, salt:
    ByteArray, nonce: ByteArray, alg: Alg): ByteArray {
    val key = deriveKey(masterKey, salt, ITERATIONS)
    val mask = computeAlgMask(masterKey, salt)
    val algXor = (alg.code.toInt() xor (mask.toInt() and 0xFF)).toByte()
    val aad = buildHeader(algXor, ITERATIONS, salt, nonce) // full header is
        AAD
    val ctWithTag = if (alg == Alg.AES) {
        val cipher = Cipher.getInstance("AES/GCM/NoPadding")
        cipher.init(Cipher.ENCRYPT_MODE, SecretKeySpec(key, "AES"),
            GCMParameterSpec(TAG_SIZE * 8, nonce))
        cipher.updateAAD(aad)
        cipher.doFinal(data)
    } else {
        val gcm = GCMBlockCipher(SerpentEngine())
        gcm.init(true, AEADParameters(KeyParameter(key), TAG_SIZE * 8, nonce,
             aad))
        val out = ByteArray(gcm.getOutputSize(data.size))
        var len = gcm.processBytes(data, 0, data.size, out, 0)
        len += gcm.doFinal(out, len)
        out.copyOf(len)
    }
    return envelope(algXor, salt, nonce, ctWithTag)
}
```

Listing 1: Excerto de `DbCrypto` — função de encriptação

Exemplo de validação Para validar o comportamento, foi utilizada a função de desencriptação que procede à verificação do cabeçalho, desfaz a máscara do algoritmo, re-deriva a chave com os mesmos parâmetros e tenta a desencriptação. A execução correta deste processo deverá

devolver o conteúdo SQLite original, validado pela presença do identificador "magic" do SQLite.

7 Módulos do Sistema

O sistema foi concebido de forma modular, permitindo separar claramente as responsabilidades de cada componente e facilitar a manutenção futura. Cada módulo desempenha uma função específica, mas todos interagem de forma integrada para assegurar o correto funcionamento da aplicação. Entre os principais módulos destacam-se a interface de utilizador, responsável pela interação direta com o sistema, os módulos de comunicação, criptografia e gestão de dados.

7.1 Interface

A interface para computador foi desenvolvida em C#, uma linguagem que possibilita o desenvolvimento eficiente de aplicações desktop. Já a interface para dispositivos Android foi implementada em Kotlin, uma linguagem moderna e concisa, amplamente utilizada no desenvolvimento de aplicações móveis.

A interface é composta por quatro ecrãs principais:

- **Ecrã de Login:** permite ao utilizador autenticar-se na aplicação através das suas credenciais, sendo também possível visualizar o estado da ligação WireGuard.
- **Ecrã de Lista de Chats:** apresenta a lista de conversas existentes e possibilita a criação de novos chats.
- **Ecrã de Chat:** permite ao utilizador enviar e receber mensagens de outros utilizadores.
- **Ecrã de Configuração WireGuard:** possibilita a criação de um ficheiro de configuração para o WireGuard.

7.2 Módulo da VPN

A VPN encontra-se alojada num servidor com o sistema operativo AlmaOS 8.10 (AlmaOS, 2025) e é responsável pela gestão das ligações dos utilizadores, bem como pela sua autenticação. A solução selecionada foi o WireGuard, uma tecnologia de VPN de código aberto, leve e de elevado desempenho, que recorre a criptografia moderna para garantir a segurança das comunicações. Para que o utilizador possa estabelecer ligação à VPN, é necessário que o WireGuard esteja previamente configurado no respetivo dispositivo.

7.3 Base de Dados

A base de dados é utilizada para armazenar as informações dos utilizadores, incluindo as credenciais de acesso, bem como as mensagens trocadas entre eles. Para a sua implementação foi escolhido o SQLite, por se tratar de um sistema de gestão de base de dados leve, relacional e amplamente utilizado em aplicações móveis e de pequena escala.

A estrutura da base de dados é apresentada na figura 13:

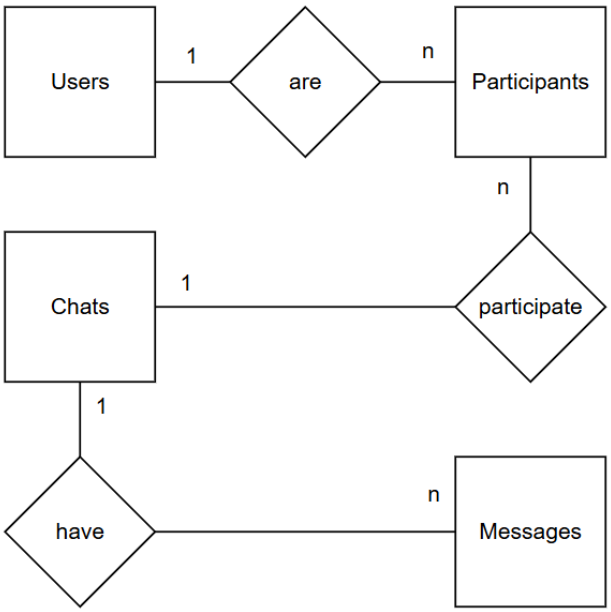


Figura 13: Estrutura da Base de Dados

A base de dados é composta por quatro tabelas principais, cada uma com as suas respetivas colunas:

- **User:**
  - UserID (chave primária)
  - Username (texto)
  - Password (texto)
- **Chat:**
  - ChatID (chave primária)
  - Name (texto)
  - AdminID (chave estrangeira, referência à tabela User)
- **Participant:**
  - ParticipantID (chave primária)
  - ChatID (chave estrangeira, referência à tabela Chat)
  - UserID (chave estrangeira, referência à tabela User)
- **Message:**
  - MessageID (chave primária)
  - ParticipantID (chave estrangeira, referência à tabela Participant)
  - Content (texto)
  - Date (data e hora da mensagem)
  - SenderUserID (chave estrangeira, referência à tabela User)

8 Desenvolvimento

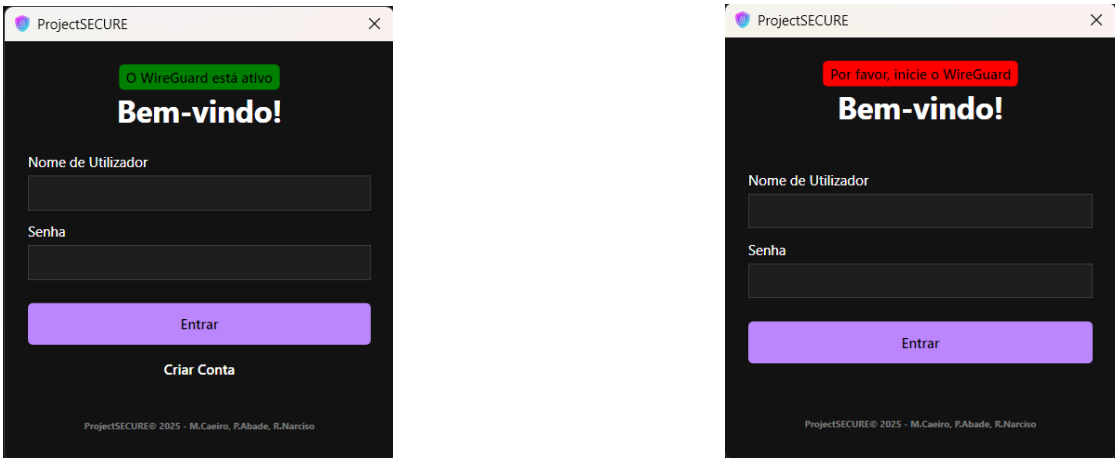
Esta secção descreve os principais módulos do sistema, com foco na implementação das interfaces, funcionalidades de comunicação, gestão de mensagens e configuração da VPN. Serão apresentados os detalhes técnicos das aplicações para computador e Android, bem como os processos de autenticação, encriptação e gestão de dados.

8.1 Interface para Computador

Para o desenvolvimento da interface para computador foi utilizado o Visual Studio Code, um ambiente de desenvolvimento integrado (IDE) da Microsoft, que oferece suporte a diversas linguagens e ferramentas, permitindo um processo de implementação ágil e organizado.

8.1.1 Ecrã de Login

O ecrã de login é composto por um formulário onde o utilizador pode introduzir as suas credenciais, bem como um botão para efetuar a autenticação na aplicação. Caso o utilizador não possua uma conta, pode criar uma diretamente no mesmo formulário, sendo apenas necessário preencher os campos e selecionar a opção de criação de conta. Esta ação apenas é possível quando a ligação ao WireGuard está ativa. Adicionalmente, o ecrã apresenta o estado da ligação ao WireGuard, indicando se esta se encontra ativa ou inativa.

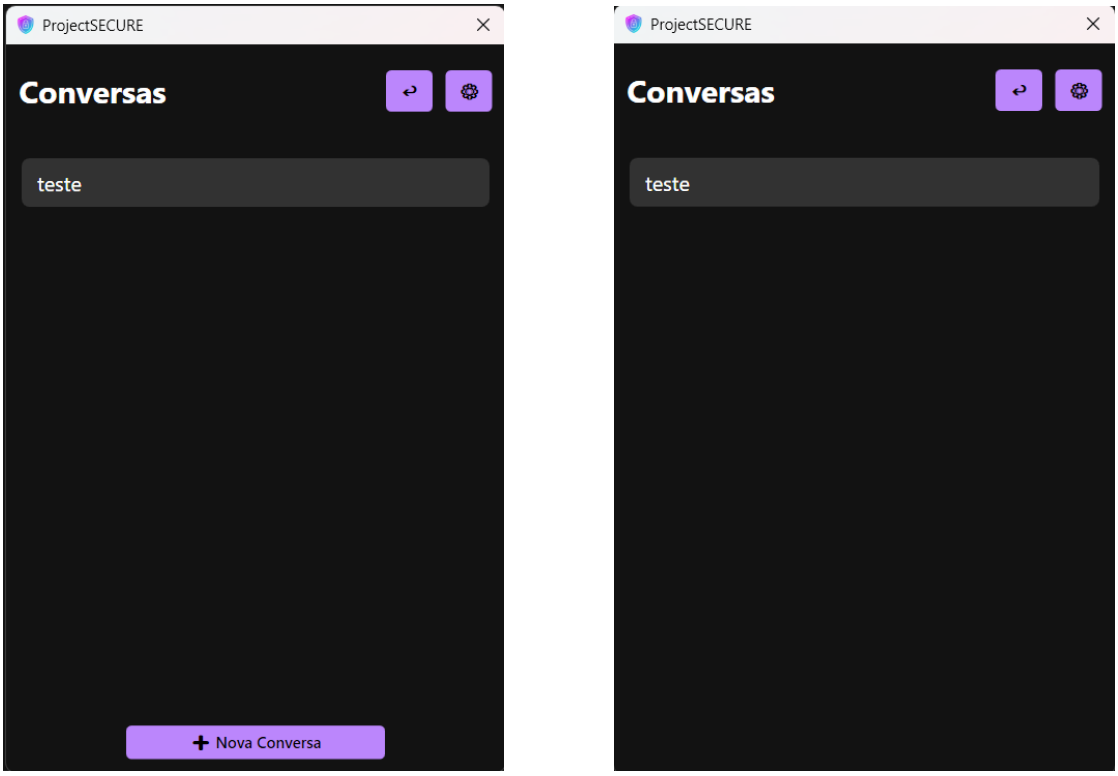


(a) WireGuard Ativado (b) WireGuard Desativado

Figura 14: Ecrã de Login — Computador

8.1.2 Ecrã de Lista de Chats

O ecrã de lista de chats apresenta os chats existentes e inclui um botão para a criação de novos chats, que apenas pode ser utilizado quando a ligação ao WireGuard se encontra ativa. No canto superior direito do ecrã, encontra-se um botão para aceder às definições da aplicação, permitindo ao utilizador selecionar o modo de visualização, claro ou escuro. À sua esquerda, existe um botão para terminar a sessão, possibilitando a troca de utilizador de forma rápida e segura.

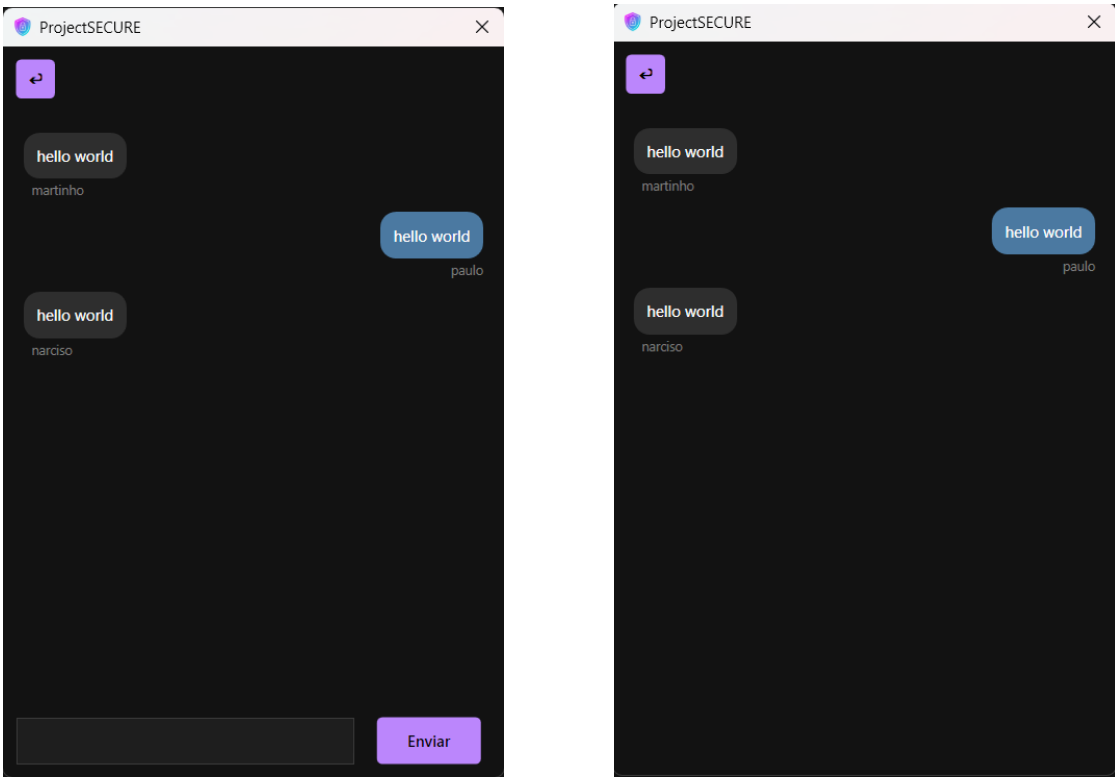


(a) WireGuard Ativado (b) WireGuard Desativado

Figura 15: Ecrã de Lista de Chats — Computador

8.1.3 Ecrã de Chat

O ecrã de chat apresenta a lista de mensagens trocadas entre os utilizadores, sendo que as mensagens recebidas aparecem no lado esquerdo, enquanto as mensagens enviadas são exibidas no lado direito. O envio de mensagens apenas é possível quando a ligação ao WireGuard se encontra ativa. Para enviar uma mensagem, o utilizador deve introduzir o texto no campo apropriado e selecionar o botão de envio.



(a) WireGuard Ativado (b) Wireguard Desativado

Figura 16: Ecrã de Chat - Computador

8.1.4 Ecrã de Configuração WireGuard

O ecrã de configuração do WireGuard apresenta um formulário onde o utilizador deve introduzir o seu nome. Após preenchimento, é disponibilizado um botão que permite descarregar o ficheiro de configuração do WireGuard, necessário para estabelecer a ligação à VPN.

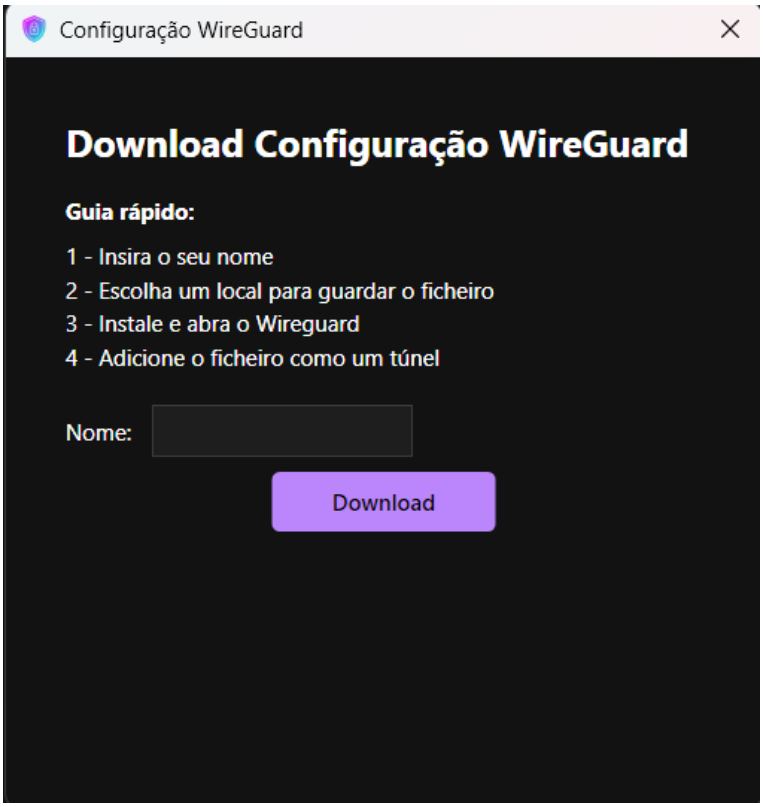


Figura 17: Ecrã de Configuração WireGuard - Computador

8.2 Interface para Android

Para o desenvolvimento da interface para Android foi utilizado o Android Studio, o ambiente de desenvolvimento integrado (IDE) oficial para o sistema operativo Android, que oferece suporte a ferramentas e bibliotecas nativas, permitindo um desenvolvimento eficiente e estruturado das aplicações móveis.

8.2.1 Ecrã de Login

O ecrã de login apresenta um formulário onde o utilizador pode introduzir as suas credenciais e um botão para efetuar a autenticação na aplicação. O utilizador pode iniciar sessão ou criar uma nova conta caso ainda não possua uma. A interface oferece também a possibilidade de facilitar a configuração da comunicação, permitindo ao utilizador descarregar o ficheiro de configuração do WireGuard diretamente a partir do ecrã de login.

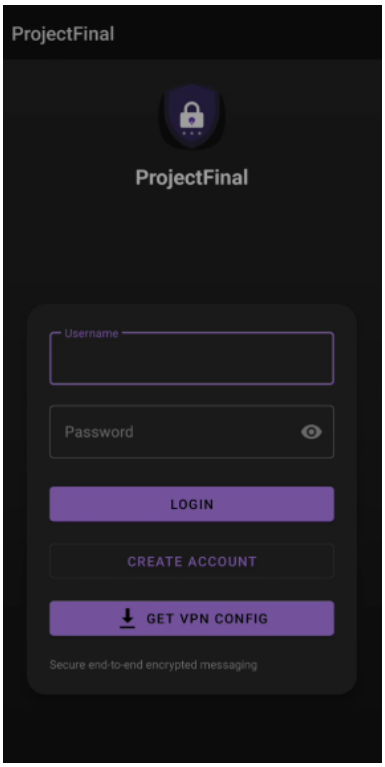


Figura 18: Ecrã de Login - Android

8.2.2 Ecrã de configuração do WireGuard

Neste ecrã, o utilizador deve introduzir um nome associado à sua ligação WireGuard. Após o preenchimento e envio do pedido, é disponibilizado o ficheiro de configuração do WireGuard, que pode ser importado diretamente na aplicação do WireGuard no dispositivo Android, permitindo a configuração rápida e segura da ligação à VPN.

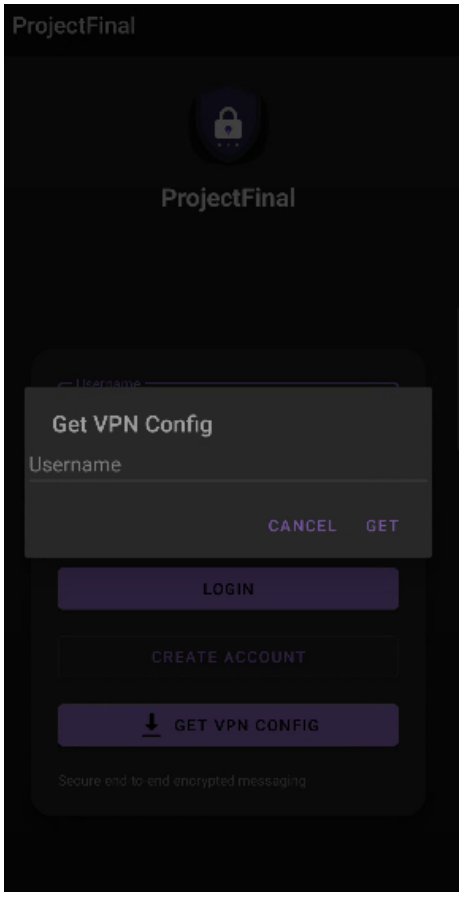


Figura 19: Ecrã de Configuração do WireGuard - Android

8.2.3 Ecrã de Lista de Chats

O ecrã de lista de chats apresenta uma relação das conversas anteriores, permitindo ao utilizador selecionar qualquer uma delas para visualizar o respetivo conteúdo. Adicionalmente, é possível iniciar uma nova conversa caso o utilizador assim o deseje, promovendo uma gestão simples e intuitiva das interações.

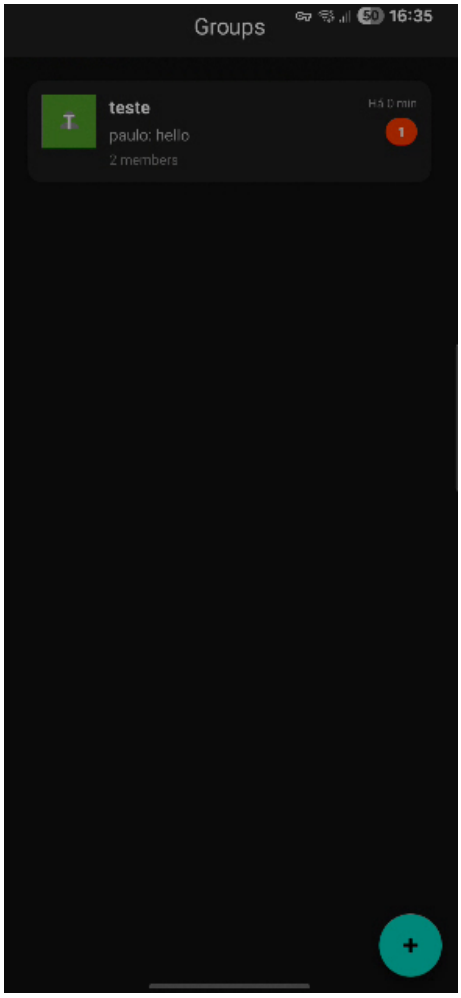


Figura 20: Lista de Chats - Android

8.2.4 Ecrã de Chat

O ecrã de chat apresenta a lista de mensagens trocadas entre os utilizadores, sendo que as mensagens recebidas são exibidas no lado esquerdo, enquanto as mensagens enviadas aparecem no lado direito. O envio de mensagens apenas é possível quando a ligação ao WireGuard se encontra ativa. Para enviar uma mensagem, o utilizador deve introduzir o texto no campo apropriado e selecionar o botão de envio.

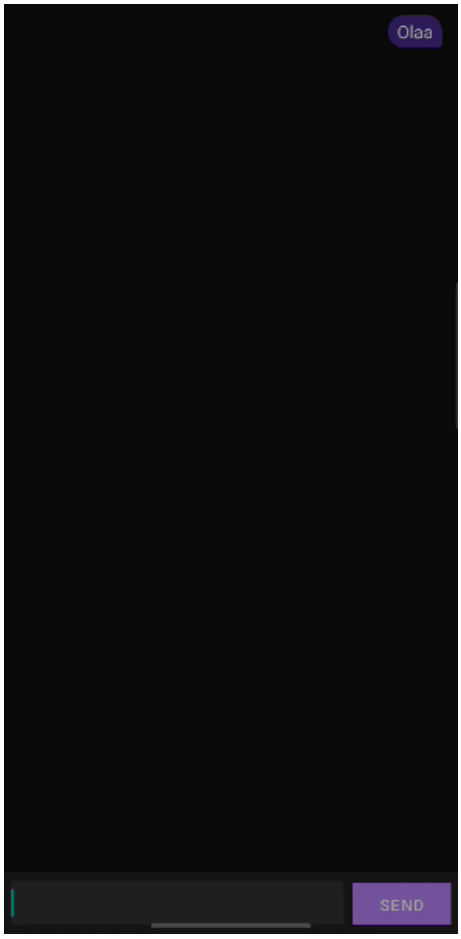


Figura 21: Ecrã de Chat - Android

8.3 AlmaOS - Serviço de VPN

Para configurar corretamente o serviço de VPN, é necessário instalar o WireGuard no servidor AlmaOS 8.10. Para tal, devem ser seguidos os seguintes passos:

1. Adicionar o repositório EPEL: `sudo dnf install epel-release`
2. Adicionar o repositório ELRepo: `sudo dnf install https://www.elrepo.org/elrepo-release-8.`
3. Ativar o CodeReady Builder: `sudo /usr/bin/crb enable`
4. Atualizar os metadados: `sudo dnf makecache`
5. Instalar o módulo WireGuard: `sudo dnf -enablerepo=elrepo install kmod-wireguard -y`
6. Instalar as ferramentas WireGuard: `sudo dnf install wireguard-tools -y`

Para verificar se o WireGuard foi instalado corretamente, pode-se utilizar o comando:

```
sudo modprobe wireguard
```

Se o comando não apresentar qualquer output, significa que o WireGuard se encontra instalado corretamente.

De seguida, para configurar o WireGuard, é necessário gerar as chaves de criptografia. Para tal, aceda à pasta `/etc/wireguard` e execute o seguinte comando:

```
wg genkey | tee server_private.key | wg pubkey > server_public.key
chmod 600 server_private.key
```

Este procedimento gera duas chaves: uma privada e uma pública, que serão utilizadas para autenticar os utilizadores na VPN.

Após a criação das chaves de criptografia, é necessário gerar o ficheiro de configuração do WireGuard, onde serão definidas as configurações da VPN, incluindo o endereço IP da rede, a porta de escuta, as chaves de criptografia, entre outros parâmetros.

Para tal, deve ser criado o ficheiro **wg0.conf** na pasta **/etc/wireguard**, contendo o seguinte:

```
[Interface]
PrivateKey = <Chave Privada do Servidor>
Address = 10.0.0.1/24
ListenPort = 51820
SaveConfig = true

[Peer]
PublicKey = <Chave Pública do Cliente>
AllowedIPs = 10.0.0.2/32
```

Para assegurar o correto funcionamento da VPN, é necessário ativar o encaminhamento de endereços IP, permitindo o tráfego de rede. Para tal, execute os seguintes comandos:

```
# Ativar o encaminhamento de IP

echo "net.ipv4.ip_forward = 1" | sudo tee -a /etc/sysctl.conf
sudo sysctl -p

# Configurar firewall e NAT para permitir o tráfego pela porta 51820

sudo firewall-cmd --add-masquerade --permanent
sudo firewall-cmd --add-port=51820/udp --permanent
sudo iptables -t nat -A POSTROUTING -s 10.0.0.0/24 -o enp0s3 -j MASQUERADE
sudo firewall-cmd --reload
```

Para permitir o encaminhamento do tráfego de rede, é necessário adicionar uma regra no router que redirecione todos os pacotes que cheguem à porta 51820 para o servidor WireGuard.

De seguida, o serviço do WireGuard deve ser iniciado, executando o seguinte comando:

```
sudo systemctl enable --now wg-quick@wg0
```

Para possibilitar a troca de ficheiros entre o servidor e os clientes, foi implementado um *script* em Python que gere a sincronização da base de dados. Este *script* utiliza a biblioteca Flask para criar uma API REST, permitindo a comunicação segura entre cliente e servidor. No servidor, foi necessário permitir a porta 8000 para que a comunicação seja efetuada corretamente:

```
sudo firewall-cmd --add-port=8000/tcp --permanent
sudo firewall-cmd --reload
```

No lado de cada cliente, é necessário configurar o WireGuard de forma a que apenas a informação proveniente da aplicação desenvolvida seja enviada através da VPN, poupando assim recursos no dispositivo que atua como servidor. Para facilitar este processo, foi criado um *script* que gera automaticamente o ficheiro de configuração do WireGuard para o cliente. O utilizador apenas precisa de adicionar este ficheiro na aplicação WireGuard.

O mesmo *script* automatiza a adição do cliente nas configurações do servidor. Foi ainda desenvolvido um *script* complementar em Python para disponibilizar o ficheiro de configuração de forma simples para o utilizador/cliente. Este segundo *script* funciona de forma semelhante à sincronização da base de dados, com a diferença de que utiliza o endereço IP local do servidor em vez da VPN.

Para obter o ficheiro de configuração, o utilizador deve estabelecer ligação com o servidor no endereço *192.168.1.106:9595/nome*, onde *nome* corresponde ao nome do utilizador que pretende aceder à VPN. Após estabelecer a ligação, o ficheiro de configuração criado pelo administrador estará disponível para transferência. O administrador gera este ficheiro utilizando o *script* anterior, permitindo limitar o acesso às comunicações caso a VPN seja comprometida.

9 Problemas Encontrados

Durante o desenvolvimento do projeto foram identificados alguns problemas, bem como as respetivas soluções. Nas secções seguintes apresenta-se um conjunto de problemas e soluções selecionados, em função da sua relevância para o projeto.

9.1 Problemas de Conexão

Logo no início da implementação da VPN, foi necessário encontrar uma solução para a configuração do encaminhamento de pacotes, uma vez que o servidor Linux não possui um endereço IP público próprio. Para contornar este problema, procedeu-se à configuração do encaminhamento de pacotes no router, de forma a que todos os pacotes recebidos na porta 51820 fossem direcionados para o servidor Linux, o qual se encarregaria de encaminhar o restante do tráfego através da VPN.

Outro problema identificado foi a dificuldade em estabelecer a conexão inicial, ou seja, enviar o ficheiro de configuração da VPN para o dispositivo do utilizador de modo a permitir a ligação ao servidor. Para resolver esta situação, foi necessário criar um *script* que gerasse automaticamente o ficheiro de configuração do WireGuard e possibilitasse o seu envio para o dispositivo do utilizador.

9.2 Problemas na Aplicação Windows/Android

Durante o desenvolvimento da aplicação, foram identificados diversos problemas, entre os quais se destacam:

- **Problemas de compatibilidade entre tipos de base de dados (SQLite e Schemas):**
  - Incompatibilidade de nomes entre tabelas, sendo necessário uniformizar as tabelas em letra minúscula em ambas as plataformas. A solução consistiu em padronizar a definição das entidades. Exemplo:

```kotlin
@Entity(tableName = "users")
data class UserEntity(
@PrimaryKey
@ColumnInfo(name = "UserId")
val userId: String,
@ColumnInfo(name = "Name")
val name: String,
@ColumnInfo(name = "Password")
val password: String
)
```

Listing 2: Definição da tabela `users` em Kotlin

  - Configurações por defeito diferentes entre as plataformas, sendo necessário explicitar em ambos os lados as configurações essenciais, como por exemplo `NOT NULL`.

- **Permissões no Android:** Foi necessário adicionar todas as permissões obrigatórias no ficheiro `AndroidManifest.xml` para que a aplicação funcionasse corretamente. A solução passou pela inclusão explícita das permissões:

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
    /> <uses-permission android:name="android.permission.INTERNET" />
```

Listing 3: Permissões no AndroidManifest.xml

- **Visualização e sincronização incorreta de informações:** Existia também a impossibilidade de modificar a base de dados enquanto a aplicação estava a correr. Para contornar esta limitação, foi criada uma base de dados temporária (*snapshot*) através da instrução `VACUUM INTO`, garantindo consistência sem bloquear a base de dados principal:

```
1
2
3  private fun createDatabaseSnapshot(context: Context): File? {
4  val dbFile = context.getDatabasePath(DB_NAME)
5  if (!dbFile.exists()) return null
6  return try {
7  val sqlDb = SQLiteDatabase.openDatabase(dbFile.path, null,
       SQLiteDatabase.OPEN_READWRITE)
8  try {
9  sqlDb.rawQuery("PRAGMA wal_checkpoint(PASSIVE);", null).close()
10 val snapshot = File.createTempFile("upload_snapshot", ".db", context.
       cacheDir)
11 val escaped = snapshot.absolutePath.replace("'", "''")
12 sqlDb.execSQL("VACUUM INTO '$escaped'")
13 snapshot
14 } finally { sqlDb.close() }
15 } catch (e: Exception) { null }
16 }
```

Listing 4: Criação de *snapshot* temporário da base de dados

9.3 Depuração da Camada Criptográfica e Protocolo de *Framing*

Contexto A comunicação Cliente Android (Kotlin) ↔ Servidor PC (Python) exige confidencialidade e integridade ponta-a-ponta. Foi adotado Serpent-256 em modo AEAD, com AAD autenticando metadados. Durante os testes iniciais surgiram falhas de decriptação (`BAD_DECRYPT`) mesmo com parâmetros aparentemente corretos.

Sintomas

- Erros `BAD_DECRYPT` antes de obter qualquer *plaintext*.
- Chaves válidas, mas divergências em nonce, tag ou AAD.
- Envio rápido de mensagens consecutivas provocava bloqueio ou exceções.

Abordagem de Depuração

- Instrumentação com *logs* detalhados (chave, AAD, nonce, tag).
- Comparação hex/base64 entre cliente e servidor.
- Testes isolados em *scripts* Python com alterações de 1 byte.
- Validação com ferramentas externas (`openssl`, `hexdump`).

Soluções Implementadas

Normalização do Envelope e *Framing*

- Contrato binário canónico com prefixo de 4 bytes (**big-endian**).
- Estrutura: LEN, MAGIC, VER, ALG, FLAGS, SEQ, TIMESTAMP, NONCE, TAG, AAD_LEN, CT_LEN, AAD, CIPHERTEXT.
- Regras claras de endianidade e tamanhos.
- Envelope de encriptação com AAD autenticado, garantindo consistência cliente-servidor.

```kotlin
private fun encryptInternal(
    data: ByteArray,
    masterKey: ByteArray,
    salt: ByteArray,
    nonce: ByteArray,
    alg: Alg
): ByteArray {
    val key = deriveKey(masterKey, salt, ITERATIONS)
    val mask = computeAlgMask(masterKey, salt)
    val algXor = (alg.code.toInt() xor (mask.toInt() and 0xFF)).toByte()
    val aad = buildHeader(algXor, ITERATIONS, salt, nonce) // full header is
        AAD

    val ctWithTag = if (alg == Alg.AES) {
        val cipher = Cipher.getInstance("AES/GCM/NoPadding")
        cipher.init(
            Cipher.ENCRYPT_MODE,
            SecretKeySpec(key, "AES"),
            GCMParameterSpec(TAG_SIZE * 8, nonce)
        )
        cipher.updateAAD(aad)
        cipher.doFinal(data)
    } else {
        val gcm = GCMBlockCipher(SerpentEngine())
        gcm.init(true, AEADParameters(KeyParameter(key), TAG_SIZE * 8, nonce,
            aad))
        val out = ByteArray(gcm.getOutputSize(data.size))
        var len = gcm.processBytes(data, 0, data.size, out, 0)
        len += gcm.doFinal(out, len)
        out.copyOf(len)
    }
```

```
30      return envelope(algXor, salt, nonce, ctWithTag)
31  }
```

Listing 5: Função de encriptação com AAD e envelope binário

Robustez na Receção e Persistência

- Uso de *BufferAssembler* para processar apenas *frames* completos.
- Rejeição de SEQ duplicados ou regressivos (anti-replay).
- Escrita na base de dados via transações atómicas, com acesso serializado.

Resultados

- Eliminação de `BAD_DECRYPT` após ráfagas curtas.
- Envios consecutivos (30–250 ms) sem falhas ou corrupção da BD.
- Critérios de fecho: 0 falhas em campanhas prolongadas; 100 envios duplos consecutivos sem *crash*; latência média < 150 ms.

Diagnóstico e Medidas

- Condições de corrida no parsing TCP causavam concatenação ou fragmentação de *frames*.
- Reenvio isolado decifra corretamente; problema ocorre apenas em ráfagas rápidas.
- Medidas imediatas: processamento sequencial, ACK/backpressure, transações atómicas na BD.

Recomendações

- *Length-prefix* (4 bytes big-endian) + *BufferAssembler* robusto.
- Serialização de escrita na BD via *worker* ou fila dedicada.
- Validação de comprimentos de AAD, nonce e tag antes da decriptação.
- NONCE único por mensagem (prefixo da chave + SEQ).

Lições Aprendidas

- Estabilidade depende do contrato binário formal: envelope + AAD + *framing*.
- Unicidade de nonce derivada do SEQ e instrumentação detalhada são essenciais.

Fundamentos Criptográficos e Práticas

- AEAD garante confidencialidade e integridade; falha se qualquer bit for alterado.
- AAD autenticada mas não cifrada; divergências provocam `BAD_DECRYPT`.
- Nonces únicos por chave (GCM 96 bits).
- HKDF para derivação de chaves de sessão; separação por função (encrypt, mac, nonce).
- TCP pode fragmentar ou coalescer envios; usar length-prefix e validar LEN.
- Proteções adicionais: SEQ monotónico, anti-replay, TIMESTAMP, comparação de TAG em tempo constante.

Testes e Verificação

- Vetores determinísticos no arranque (*self-test*).
- *Fuzzing* do *BufferAssembler* e desserializador.
- Testes negativos: nonce repetido, TAG truncada, AAD alterada; *stress tests* com ráfagas e latências variáveis.

10 Testes Realizados

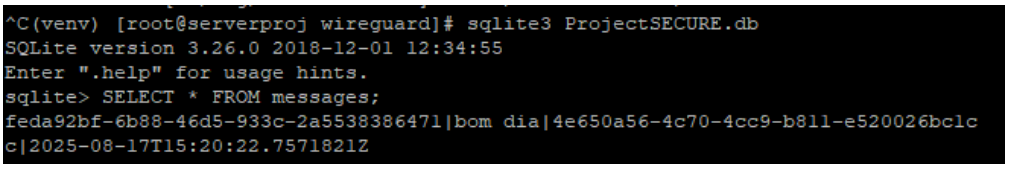
Nesta secção descrevem-se os testes efetuados ao longo do desenvolvimento, incluindo verificação da integridade da base de dados, monitorização do tráfego de rede e validação da camada criptográfica. Os testes foram realizados em diferentes cenários para garantir que o sistema cumpre os requisitos de confidencialidade, integridade e autenticidade das mensagens trocadas.

10.1 Servidor

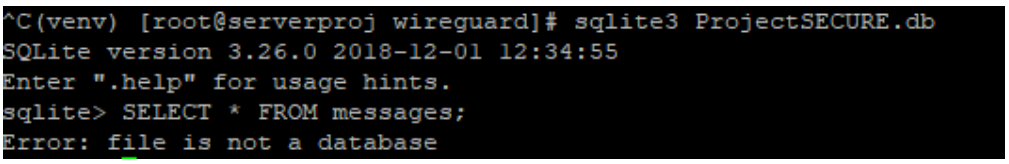
Os testes no servidor focaram-se na validação da base de dados e na monitorização das comunicações da aplicação.

Para verificar a base de dados, foram realizados ensaios de leitura do ficheiro armazenado. Nas figuras 22a e 22b é possível observar o seguinte:

- Ao ler o ficheiro sem criptografia, o conteúdo da base de dados é visualizável e reconhecível pelo sistema.
- Ao ler o ficheiro com criptografia, o conteúdo não é legível, nem reconhecido como uma base de dados válida, evidenciando que a camada de encriptação está a funcionar corretamente.



(a) Sem Criptografia

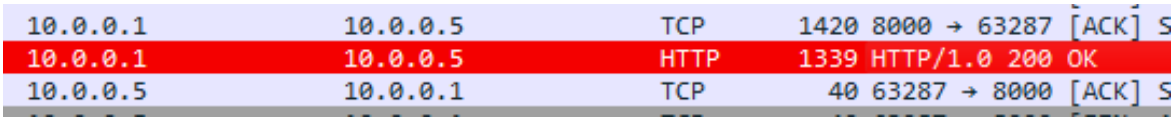


(b) Com Criptografia

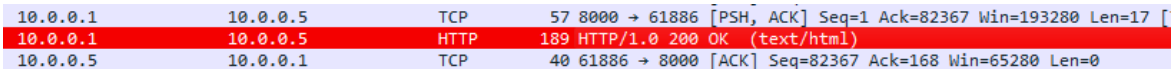
Figura 22: Visualização da Base de Dados no Servidor

10.2 Comunicações

Para monitorizar as comunicações, foi utilizado o Wireshark, permitindo a captura dos pacotes enviados e recebidos pelo servidor. Nas figuras 23a e 23b apresentam-se exemplos destes testes, comparando tráfego sem e com criptografia.



(a) Sem Criptografia



(b) Com Criptografia

Figura 23: Monitorização do tráfego no servidor com Wireshark

Para obter estas capturas, foi necessário selecionar no Wireshark o adaptador associado à VPN, garantindo que apenas o tráfego da aplicação era analisado. Ao observar as imagens, não se detectam diferenças óbvias, exceto pelo aparecimento do indicador *(text/html)* nas mensagens criptografadas.

Para uma análise mais detalhada, foi utilizado o recurso *Follow → TCP Stream* no Wireshark, permitindo visualizar o conteúdo das mensagens de forma linear. Os resultados obtidos encontram-se nas figuras 24 e 25.

Na figura 24, o tráfego sem criptografia exibe a estrutura da base de dados no inicio da mensagem, permitindo visualizar informações sensíveis, como utilizadores e mensagens.

Por outro lado, a figura 25 mostra o tráfego com criptografia AES256/Serpent. O conteúdo da base de dados torna-se ilegível para um observador externo, garantindo que apenas os sistemas com a chave correta podem aceder às informações. Esta abordagem assegura a confidencialidade das comunicações, mesmo que o tráfego seja interceptado por terceiros ou caso a VPN seja comprometida.

Figura 24: Conteúdo da base de dados sem criptografia

Figura 25: Conteúdo da base de dados com criptografia

11 Melhorias Futuras

Embora o projeto tenha alcançado os objetivos definidos, existem diversas oportunidades para expansão e refinamento da solução. Entre as melhorias possíveis, destacam-se:

- **Novas funcionalidades de comunicação:** implementação de mensagens de voz, imagens e chamadas criptografadas, de forma a expandir o leque de opções de comunicação segura entre os utilizadores, mantendo a confidencialidade e integridade dos dados.
- **Aperfeiçoamento da interface de utilizador:** melhoria da experiência do utilizador (UX) e da estética da interface, incluindo suporte a temas personalizáveis, notificações avançadas e layouts responsivos para diferentes dispositivos.
- **Fortalecimento da segurança das comunicações:** utilização de protocolos seguros de transporte, como HTTPS/TLS para a API e troca de ficheiros, proteção adicional contra ataques de rede, validação reforçada de certificados e rotinas de rotação de chaves.
- **Escalabilidade e gestão de utilizadores:** otimização da infraestrutura da VPN e do servidor, para suportar um maior número de utilizadores simultâneos, incluindo mecanismos de balanceamento de carga e monitorização centralizada.
- **Automatização e manutenção:** implementação de scripts automáticos de backup e atualização da base de dados, monitorização de logs e alertas de segurança, visando reduzir a intervenção manual e aumentar a fiabilidade do sistema.
- **Otimização da sincronização de dados:** de forma a enviar apenas as alterações recentes ou mensagens individuais, reduzindo o volume de tráfego e melhorando a eficiência da comunicação, sem comprometer a integridade da base de dados.

Estas melhorias futuras podem não só reforçar a segurança e a robustez do sistema, como também proporcionar uma experiência de utilização mais completa e profissional, servindo de base para a evolução do projeto em contextos reais.

12 Conclusão

Conclui-se, assim, que o projeto atingiu os objetivos definidos, tendo sido desenvolvido um sistema de comunicações seguras que possibilita a troca de mensagens de texto entre utilizadores, assegurando a proteção contra acessos indevidos por terceiros. De forma geral, o projeto proporcionou uma aprendizagem significativa no domínio da cibersegurança e das comunicações seguras. A implementação de uma VPN com recurso ao WireGuard revelou-se um desafio relevante, permitindo aprofundar o conhecimento sobre o funcionamento destas tecnologias. Acresce ainda o processo de implementação da camada de criptografia, que exigiu garantir a sua correta integração entre dois sistemas operativos distintos (Android e Windows), recorrendo a linguagens de programação diferentes (C# e Kotlin).

Foi igualmente necessário adquirir conhecimentos sobre criptografia, servidores, redes e protocolos de comunicação. O trabalho desenvolvido reforçou também competências relacionadas com a colaboração em equipa, nomeadamente na gestão e distribuição equilibrada de tarefas, de forma a cumprir os prazos estabelecidos. Neste sentido, o projeto não só contribuiu para a consolidação de competências técnicas e de cooperação, como também resultou numa solução que poderá servir de base a futuros desenvolvimentos e aplicações em contextos práticos.

13 Agradecimentos

Os autores agradecem ao professor Rui Miguel Soares Silva pela disponibilidade demonstrada para fornecer orientação e sugestões ao longo do projeto, bem como aos colegas de turma pelo apoio e colaboração prestados.

Bibliografia

AlmaOS. (2025). *AlmaOS: A Secure Operating System* [Página oficial do AlmaOS]. Obtido agosto 16, 2025, de https://almalinux.org

Bash. (2025). *Bash: The GNU Project's Command-Line Interpreter* [Página oficial do Bash]. Obtido agosto 16, 2025, de https://www.gnu.org/software/bash/

Foundation, P. S. (2025). *Python: A Programming Language* [Página oficial do Python]. Obtido agosto 16, 2025, de https://www.python.org/

IPBeja. (2025). *Disciplina: Estágio ou Projeto | IPBeja* [Página EP]. Obtido junho 16, 2025, de https://cms.ipbeja.pt/course/view.php?id=238

JetBrains. (2025). *Kotlin: A Modern Programming Language* [Página oficial do Kotlin]. Obtido agosto 16, 2025, de https://kotlinlang.org/

Microsoft. (2025). *C#: A Modern Programming Language for .NET* [Página oficial do C#]. Obtido agosto 16, 2025, de https://learn.microsoft.com/en-us/dotnet/csharp/

SQLite. (2025). *SQLite: A Self-Contained, High-Performance, Embedded SQL Database Engine* [Página oficial do SQLite]. Obtido agosto 16, 2025, de https://www.sqlite.org/

Telegram Messenger LLP. (2025). *Telegram: A Cloud-Based Instant Messaging App* [Página oficial do Telegram]. Obtido agosto 16, 2025, de https://telegram.org/

WhatsApp Inc. (2025). *WhatsApp: A Messaging App* [Página oficial do WhatsApp]. Obtido agosto 16, 2025, de https://www.whatsapp.com/

WireGuard. (2025). *WireGuard: Next Generation Kernel Network Tunnel* [Página oficial do WireGuard]. Obtido agosto 16, 2025, de https://www.wireguard.com/