# 安全即时通讯系统

一个全栈、实时的聊天应用，它演示了如何为文本消息和文件传输实现健壮的端到端加密（E2EE）。本项目旨在作为一份综合指南，指导如何使用现代Web技术构建安全的通信系统，其中服务器仅作为加密数据的盲中继，从而确保绝对的用户隐私。

## 🚀 核心功能

- **端到端加密消息**：所有文本消息在发送前都在客户端加密，并且只能由预期的接收者解密。
- **安全文件传输**：使用混合加密方案（RSA + AES-GCM）对文件进行加密，确保大文件的安全性和效率。
- **实时通信**：利用WebSockets实现即时消息传递和用户的实时在线状态更新。
- **自动化密钥管理**：首次登录时，RSA密钥对会在浏览器中自动生成。私钥永远不会离开用户设备，而公钥则发布到服务器以供发现。
- **现代化技术栈**：后端采用FastAPI，前端采用Remix（React），提供高性能、开发者友好的体验。
- **安全会话处理**：使用签名的、`httpOnly`的Cookie进行健壮的会话管理，有效缓解XSS风险。

---

## 🛠️ 技术栈

| 类别 | 技术 | 描述 |
| --- | --- | --- |
| **后端** | Python 3, FastAPI, Uvicorn | 一个用于构建API的高性能、现代化的Web框架。 |
| **前端** | TypeScript, Remix, React, Vite | 一个用于打造快速、有弹性的用户体验的全栈Web框架。 |
| **样式** | Tailwind CSS, Shadcn/ui, Lucide Icons | 一个实用程序优先的CSS框架，用于快速UI开发。 |
| **数据库** | MongoDB 与 Beanie ODM | 一个灵活的NoSQL数据库，用于存储用户数据和公钥。 |
| **安全** | Web Crypto API (浏览器原生) | 我们E2EE实现的核心，提供加密原语。 |
| **实时** | WebSockets | 实现客户端和服务器之间的双向实时通信。 |

---

## 🔐 工作原理：端到端加密流程

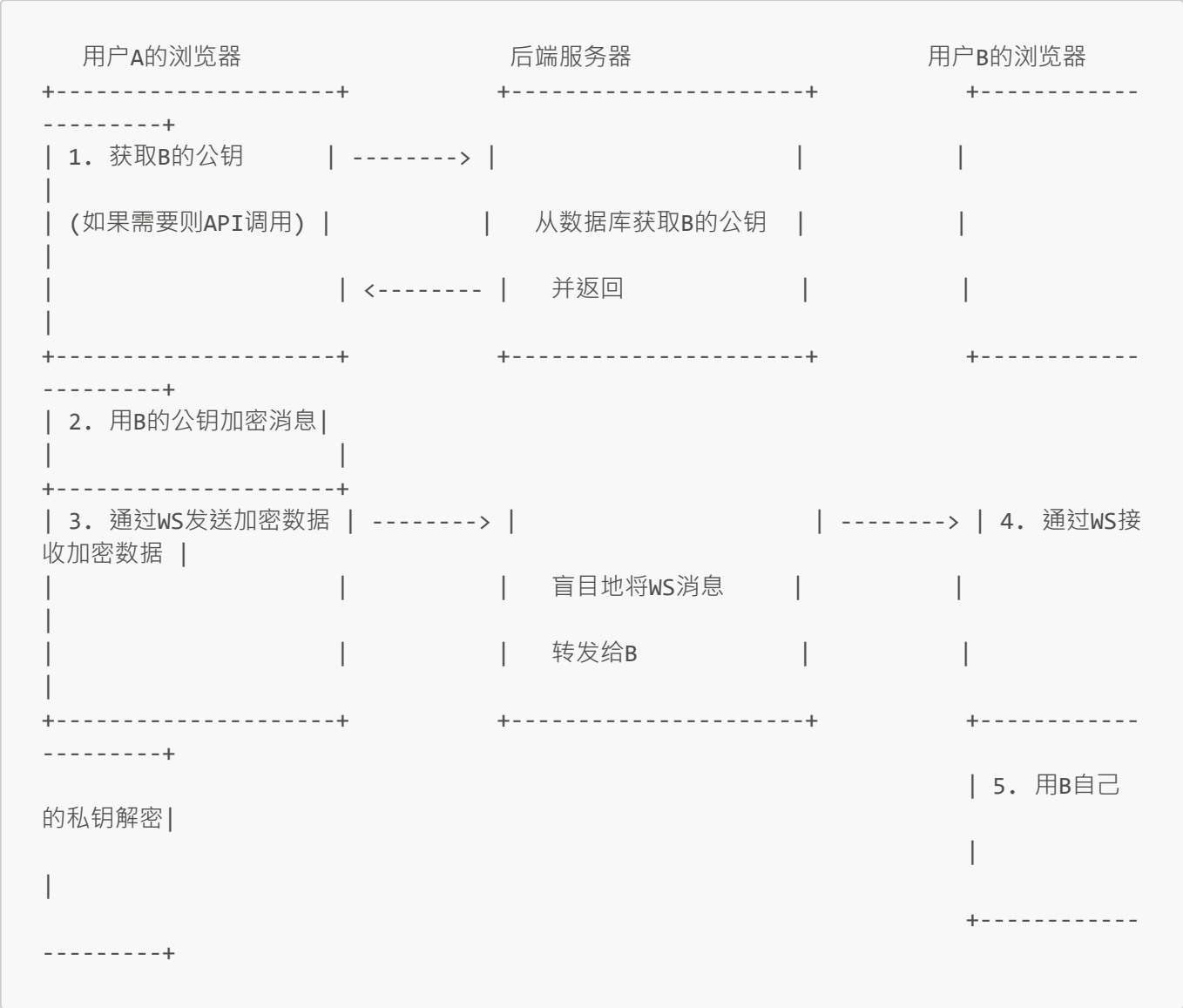
The security of this application hinges on a carefully designed E2EE protocol. Here's a detailed breakdown of the process.

### 阶段一：密钥生成与管理

1. **首次登录**：当用户首次登录时，前端会检查服务器上是否存有其公钥 (`hasPublicKey: false`)。
2. **密钥对生成**：如果密钥不存在，将调用浏览器的 **Web Crypto API** 来生成一个新的`RSA-OAEP` 4096位密钥对。
3. **安全存储：**
    - **私钥**立即存储在浏览器的`localStorage`中。**它永远不会离开用户的设备。**
    - **公钥**被导出并发送到服务器。

4. **公钥发布**：服务器将用户的公钥存储在数据库中，使其可供其他用户获取。

## 阶段二：加密并发送文本消息（用户A -> 用户B）




1. **获取接收方密钥**：用户A发起与用户B的聊天。应用首先从服务器请求用户B的公钥。该密钥随后在本地缓存以供会话期间使用。
2. **加密**：用户A输入一条消息。明文使用**用户B的公钥**通过RSA-OAEP算法进行加密。
3. **传输**：生成的密文（作为Base64字符串）被发送到WebSocket服务器。
4. **中继**：服务器只能看到一团不透明的加密数据。它识别接收者并将消息转发，无法读取其内容。
5. **解密**：用户B的浏览器接收到密文。它使用其**自己的私钥**（从`localStorage`中检索）来解密消息，从而显示原始明文。

## 阶段三：加密并发送文件（混合加密）

传输大文件需要一种比单独使用非对称加密更高效的方法。我们使用**混合加密**方案（AES-GCM + RSA-OAEP）。


1. **生成对称密钥**：当用户A选择要发送的文件时，浏览器会生成一个强壮的、随机的256位**AES-GCM对称密钥**。此密钥仅用于这一次文件传输。
2. **加密文件**：文件内容使用这个新生成的AES密钥进行高效加密。
3. **加密对称密钥**：然后，AES密钥本身使用**用户B的公钥RSA密钥**进行加密。
4. **传输数据包**：加密后的文件和加密后的AES密钥被打包在一起，作为Base64字符串发送到服务器。
5. **中继**：服务器再次盲目地将这个数据包转发给用户B。
6. **解密对称密钥**：用户B的浏览器接收到数据包。它首先使用其**自己的私钥RSA密钥**来解密被加密的AES密钥。
7. **解密文件**：随着对称AES密钥的解密，它被用来快速解密大文件内容。解密后的文件作为一个可下载的Blob呈现给用户B。

---

## 项目结构

```
/
├── backend_server/         # FastAPI 后端
│   ├── server/
│   │   ├── auth.py         # 认证逻辑 (哈希, token)
│   │   ├── db.py           # 数据库初始化
│   │   ├── main.py         # FastAPI主应用, API端点, WebSocket逻辑
│   │   └── models.py       # Beanie ODM 模型 (用于MongoDB)
│   └── requirements.txt    # Python 依赖
│
└── my-secure-chat-app/     # Remix 前端
    ├── app/
```

```
│   ├── components/     # 可复用的React组件 (ChatInterface等)
│   ├── routes/         # Remix路由文件 (auth.tsx, chat.tsx)
│   ├── styles/         # CSS 文件
│   └── utils/          # 工具函数
│       ├── crypto.client.ts # 所有Web Crypto API逻辑 (密钥生成, 加密, 解密)
│       └── session.server.ts# 服务器端会话管理
├── public/             # 静态资源
├── package.json        # Node.js 依赖
└── README.md           # 本文件
```

---

## 🚀 开始使用

请按照以下说明在您的本地计算机上运行此项目。

### 先决条件

- Node.js (v18 或更高版本)
- Python (v3.10 或更高版本)
- 正在运行的 MongoDB 实例 (或像MongoDB Atlas这样的云实例)。

### 1. 后端设置

```
# 1. 导航到后端目录
cd backend_server

# 2. 创建并激活虚拟环境
python -m venv venv
# Windows:
# venv\Scripts\activate
# macOS/Linux:
# source venv/bin/activate

# 3. 安装Python依赖
pip install -r requirements.txt

# 4. 运行后端服务器
# 它将连接到MongoDB并在 http://127.0.0.1:8000 上运行
uvicorn server.main:app --reload
```

### 2. 前端设置

```
# 1. 打开一个新终端并导航到前端目录
cd my-secure-chat-app

# 2. 安装Node.js依赖
npm install
```

```
# 3. 运行前端开发服务器
# 它将在 http://localhost:5173 上可用
npm run dev
```

### 3. 使用方法

1. 在两个独立的浏览器窗口或配置文件中打开 http://localhost:5173，以模拟两个不同的用户。
2. 为每个用户注册一个新帐户（例如，`user-a` 和 `user-b`）。
3. 用两个帐户登录。您应该会看到每个用户出现在对方的"在线好友"列表中。
4. 开始在他们之间发送加密消息和文件吧！
5. **显示消息**:
    - 客户端的 React 组件状态被更新，将解密后的明文消息渲染到聊天窗口中。

至此，一次完整的、端到端加密的通信流程宣告完成。

---

## 开发与部署

### 启动后端服务

```
# 进入后端目录
cd d:/security_inst_comm_system/backend_server

# 安装依赖
pip install -r requirements.txt

# 启动服务器
uvicorn server.main:app --host 127.0.0.1 --port 8000 --reload
```

### 启动前端服务

```
# 进入前端目录
cd d:/security_inst_comm_system/my-secure-chat-app

# 安装依赖
npm install

# 启动开发服务器
npm run dev
```