CODE STREET

Тестове завдання

Mini Video Chat (SwiftUI + WebRTC + Socket.IO)

Загальні вимоги

- UI: SwiftUI — **обов'язково**
- Архітектура: MVVM + Coordinator (рекомендовано Stinsen, але можна власну реалізацію)
- Логіка сигналінгу: Socket.IO
- WebRTC:
  - на симуляторі — тестове відео webRtcTestVideo.mp4 (Можна використати будь яке інше)
  - на девайсі — локальна камера

Логіка async / потоків даних

У всіх сервісах (Socket.IO, WebRTC, Coordinator interaction, VM → Service) потрібно використовувати один підхід до асинхронності:
- або повністю на Swift Concurrency (async/await, AsyncStream)
- або повністю на Combine (Publisher/Subscriber)

Змішувати ці підходи у сервісному шарі **заборонено**.
Можна змікувати лише у SwiftUI-в'юшках, якщо цього вимагають UI API (наприклад, .task {} + Combine subscription).

Тобто:
- сервіс WebRTC → має працювати або повністю через async/await, або через Combine
- сервіс SocketIO → теж один підхід
- логіка координатора → один підхід

Головне правило:

У всіх non-UI шарах використовувати єдиний асинк-патерн без міксування.

Готовий проект необхідно розмістити у відкритому GitHub-репозиторії.

Екран 1 — Login

- UI:
  - TextField: username
  - TextField: roomId (int)
  - Button: Продовжити
- Логіка:
  - username і roomId зберігаються через UserDefaults / Keychain
  - після натискання переходити на екран комнаты

Екран 2 — Room State

Екран управління з'єднанням із signaling server.

UI:
- Label: username
- Label: roomId
- Switch / Toggle: isCaller

Після підключення до сокета має замінюватися на статичний текст з роллю
- Button:
- "Під'єднатись" → коли socket не підключений
- "Від'єднатись" → коли socket підключений
- Стан peer'а:
- показати, чи інший peer знаходиться в кімнаті
- peer доступний тоді, коли його роль протилежна до isCaller
- Button "Почати дзвінок"
- активна тільки коли peer доступний
- відкриває екран відеочату
- Button "Logout"
- очищує збережені username/roomId
- повертає на Login

Еміти / події для цього екрана

Клієнт отримує стани peer'а з сервера:

Підключення peer'а
- event: room_user_joined
- payload:

```
{
  "username": "Bob",
  "roomId": 1,
  "isCaller": false
}
```

Відключення peer'а
- event: room_user_left
- payload:

```
{
  "username": "Bob"
}
```

Клієнт має оновлювати UI відповідно.

Екран 3 — Video Chat

Головний WebRTC екран.

UI:
- Remote Video View
- Local Video View
- Текст: ім'я піра, з яким іде дзвінок
- Кнопка End Call

Основна логіка:
- В залежності від isCaller:

Caller:
1. Створює offer
2. Емітить offer
3. Чекає answer
4. Далі — обмін candidate

Callee:
1. Чекає offer
2. Встановлює remoteDescription
3. Створює answer
4. Емітить answer
5. Далі — candidate

Додатково (обов'язково):
- Якщо приходить room_user_left → дзвінок завершується автоматично
→ повернення на екран Room State
→ WebRTC peer connection закривається
- Для симулятора — локальний відеотрек обов'язково використовує webRtcTestVideo

Еміти WebRTC (важливо)

Усі події сигналінгу не типізовані — сервер передає payload "як є".

Тобто:
- якщо один peer надіслав:

```
{ "type": "offer", "sdp": "..." }
```

інший peer отримує саме це, без змін.

Еміти для клієнта:

offer

```
{
  "type": "offer",
  "sdp": "STRING"
}
```

answer

```
{
  "type": "answer",
  "sdp": "STRING"
}
```

candidate

```
{
  "candidate": "candidate:...",
  "sdpMid": "0",
  "sdpMLineIndex": 0
}
```

Socket Server

Тобі видасться:
- код сервера (index.js)
- інструкція запуску (SIGNALING_SERVER_SETUP.md)
- документація по емітам (SIGNALING_PROTOCOL.md)

Обов'язково:
Перед запуском застосунку необхідно:
npm install
node index.js

Що очікується в результаті

Посилання на GitHub репозиторій із готовим застосунком
- MVVM + Coordinator
- Робочий WebRTC дзвінок Caller ↔ Callee
- Робота в симуляторі (через тестове відео)
- Завершення дзвінка при відключенні peer'а
- Правильне керування станами та переходами між екранами
- Окремі сервіси для:
- Socket.IO
- WebRTC

Буде плюсом винесення їх в окремий Swift Package