Go Temporary Chat Room Service Application

Overview:

- A lightweight, highly concurrent chat application built in Go, utilizing **WebSockets** (gorilla/websocket) for real-time communication

تطبيق دردشة خفيف وسريع، مبني بلغة **Go** ، وبيستخدم **WebSockets مكتبة (gorilla/websocket)** بحيث التواصل يكون لحظي

- The system supports multiple isolated **Rooms** that are automatically destroyed after a configurable **lifetime** (e.g., 5 minutes).

بيدعم وجود كذا **غرفة (Room)** معزولة عن بعضها، وكل غرفة بتتقفل وتتمسح لوحدها بعد **مدة زمنية** محددة (5 دقايق).

Key Technologies:

- **Go , WebSockets , Goroutines , Channels**

1-Core Architecture (Hub, Room, Client)

- **The Hub (Central Manager)**

The central structure (hub.Hub) responsible for managing the lifecycle of all active chat rooms. It uses a **Read/Write Mutex (sync.RWMutex)** to ensure thread-safe access to the rooms map.

هو الهيكل المركزي **(`hub.Hub`)** المسؤول عن إدارة حياة كل غرف الدردشة الموجودة. بيستخدم **قفل Read/Write Mutex** بحيث يضمن أن التعامل مع خريطة الغرف **(Map)** يكون آمن ومتزامن.**(Thread-safe)**

- **The Room**

An isolated process that manages its own set of connected users (clients) and handles message broadcasting within the room. It operates via a dedicated **Goroutine** (r.run()).

عملية معزولة بتدير مجموعة العملاء المتصلين بيها **(`clients`)** وبتتولى عملية بث الرسائل لكل العملاء جواها. بتشتغل عن طريق **Goroutine** خاصة بيها**(`r.run()`)**

- **The Client**

Represents one **WebSocket** connection (*websocket.Conn) and is responsible for all I/O operations through two dedicated **Goroutines**: ReadPump and WritePum

بيمثل اتصال WebSocket واحد **(`*websocket.Conn`)** ومسؤول عن كل عمليات الإدخال والإخراج عن طريق اثنين **Goroutines** مخصصتين : **`ReadPump`** و **`WritePump`**.

2-Concurrency - Goroutines (The Lightweight Threads)

Concurrency is achieved using Goroutines , Go's lightweight thread-like functions.

- **Key Goroutines launched per connection**

1. client.ReadPump():

Reads incoming WebSocket messages from the client, wraps the text into a structured Message, and sends it to the **Room's Broadcast Channel**.

بتقرأ الرسائل اللي جاية من العميل الـ **(WebSocket)**، بتحول النص لـ **`Message`** منظم، وبتبعتها على **قناة** **`Broadcast`** الخاصة بالغرفة

2. client.WritePump():

Listens to the **Client's send Channel** for outgoing messages and sends periodic **Ping messages** to keep the connection alive.

بتستنى الرسائل اللي هتتبعت على ال**channel** الخاصة بال **client** ، وكمان بتبعت رسائل **Ping** دورية بحيث تحافظ على الاتصال شغال.

Point

Room Goroutine: Each Room runs its own r.run() Goroutine, which is the heart of the room's message processing.

كل غرفة بتشغل الـ **Goroutine** الخاصة بها **(`r.run()`)** ، ودي اساس عملية معالجة الرسائل في اي Room.

3-Concurrency - Channels (Safe Communication)

Channels are the key mechanism for safe communication between the Room Goroutine and the various Client Goroutines.

هي الطريقة الرئيسية للتواصل **الآمن** ما بين الـ **Goroutine** الخاصة بالRooms والـ **Goroutines** المختلفة الخاصة بال Clients.

⟹ Channels used by the Room (`r.run` loop):

الchannels اللي Room بتستخدمها في `r.run`

- **Register / Unregister:** Used by the WSHandler and ReadPump to safely add or remove clients from the Room's internal map.

بيتم استخدامهم بواسطة الـ **`WSHandler`** و الـ **`ReadPump`** بحيث يتم إضافة أو إزالة العملاء بأمان من خريطة العملاء الداخلية للغرفة.

- **Broadcast:** Where all incoming messages from clients are sent for processing.

الchannel اللي بيتم إرسال كل الرسائل اللي جاية من العملاء عليها عشان تتجهز.

- **closed:** A signal channel that, when closed, triggers the room shutdown.

قناة إشارة، لما بتتقفل بتدي إشارة لبدء عملية إغلاق الغرفة

.

4-The Broadcast Mechanism (Non-Blocking Send)

How the Room sends messages (The `r.run` loop) **إزاي الغرفة بتبعت الرسائل**

When a message arrives on the Broadcast channel, the r.run loop iterates over all connected clients.

لما بتيجي رسالة على قناة Broadcast، حلقة r.run بتلف على كل العملاء المتصلين.

Critical Safety Feature: Non-Blocking Send:

- The `select` statement is used with a `default` case to attempt sending the message to the client's `c.send` channel.
- If the send blocks (the `c.send` channel is full), the client is identified as a **"Slow Client"**.
- The **Slow Client** is immediately removed from the room and disconnected, preventing it from slowing down or blocking the entire room process.

- بيتم استخدام جملة `select` مع حالة `default` عشان تحاول تبعت الرسالة على قناة `c.send` الخاصة بالعميل.
- لو الإرسال اتحظر (يعني قناة `c.send` مليانة)، العميل ده بيعتبر **"عميل بطيء" (Slow Client).**
- **العميل البطيء** بيتم إزالته من الغرفة وفصله على طول، وده بيمنعه من إنه يبطّئ أو يحظر عملية الغرفة بالكامل.

5-Lifetime Management & Closure

Room Lifetime

When a room is created, `time.AfterFunc` schedules the `CloseRoom()` function to execute after the configured duration (e.g., 5 minutes).

لما الغرفة بتتعمل، دالة `time.AfterFunc` بتبرمج دالة `CloseRoom()` إنها تشتغل بعد المدة المحددة (زي 5 دقايق).

Closure Process (`r.closed`): عملية الإغلاق (`r.closed`)

1. `CloseRoom()` is called, which simply closes the `r.closed` channel.
2. The `r.run()` Goroutine receives the signal on the closed channel.
3. It broadcasts a "room closed" system message to all clients, forcefully closes all client
   connections, and terminates the Room Goroutine.
4. Finally, it calls the Hub API to delete the room entry from the central map, freeing up resources.

1. دالة `CloseRoom()` بتشتغل، وبتعمل إغلاق لقناة `r.closed`.
2. الـ Goroutine الخاصة بـ `r.run()` بتستقبل الإشارة دي
3. بتبعت رسالة نظام "الغرفة اتقفلت" لكل العملاء، وبتقفل كل اتصالات العملاء بالقوة، وبتنهي الـ Goroutine الخاصة بالغرفة.
4. وأخيراً، بتنادي الـ Hub عشان يمسح الغرفة من الـ Map المركزية، وبكده الموارد بتفضى.

مقارنة بين Goroutines و Channels في مشروعنا

1. Goroutines

الـ **Goroutines** هي زي الـ **Threads** (الخيوط) بس أخف بكتير وأسرع في التشغيل، وهي اللي بتمكّن البرنامج إنه يعمل كذا حاجة في نفس الوقت (التوازي/التزامن).

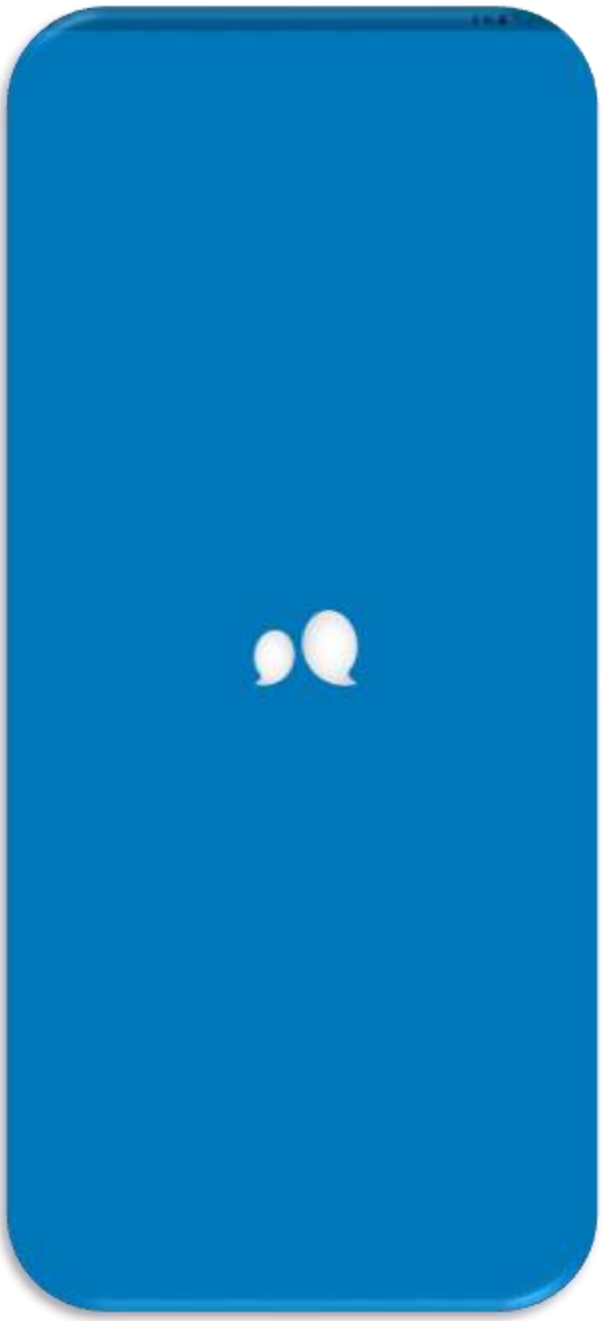
المفهوم	الوصف	استخدامها في المشروع
الـGoroutines	**"المُنفِّذ"** أو **"العامل"**. هي وظيفة بتشتغل في الخلفية بشكل متزامن مع بقية البرنامج.	* **لكل عميل متصل (Client):** بنطلق 2 Goroutines واحدة للقراءة (`ReadPump`) وواحدة للكتابة (`WritePump`).
وظيفتها الأساسية	تشغيل العمليات بشكل متوازٍ لضمان عدم حظر أي عملية للعمليات الأخرى (مثلاً، قراءة رسالة من عميل مايوقفش إرسال رسالة لعميل تاني).	* **لكل غرفة (Room):** بنطلق Goroutine واحدة (`r.run()`) للتحكم في كل عمليات الغرفة (التسجيل والبث والإغلاق).

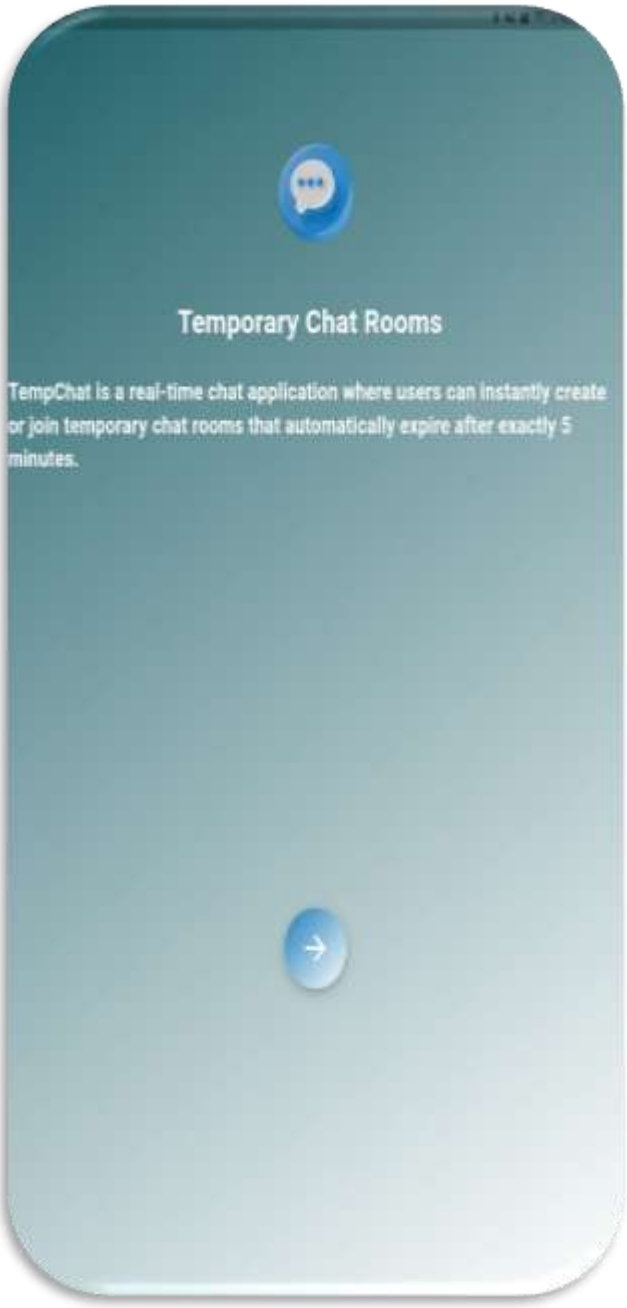
2. Channels

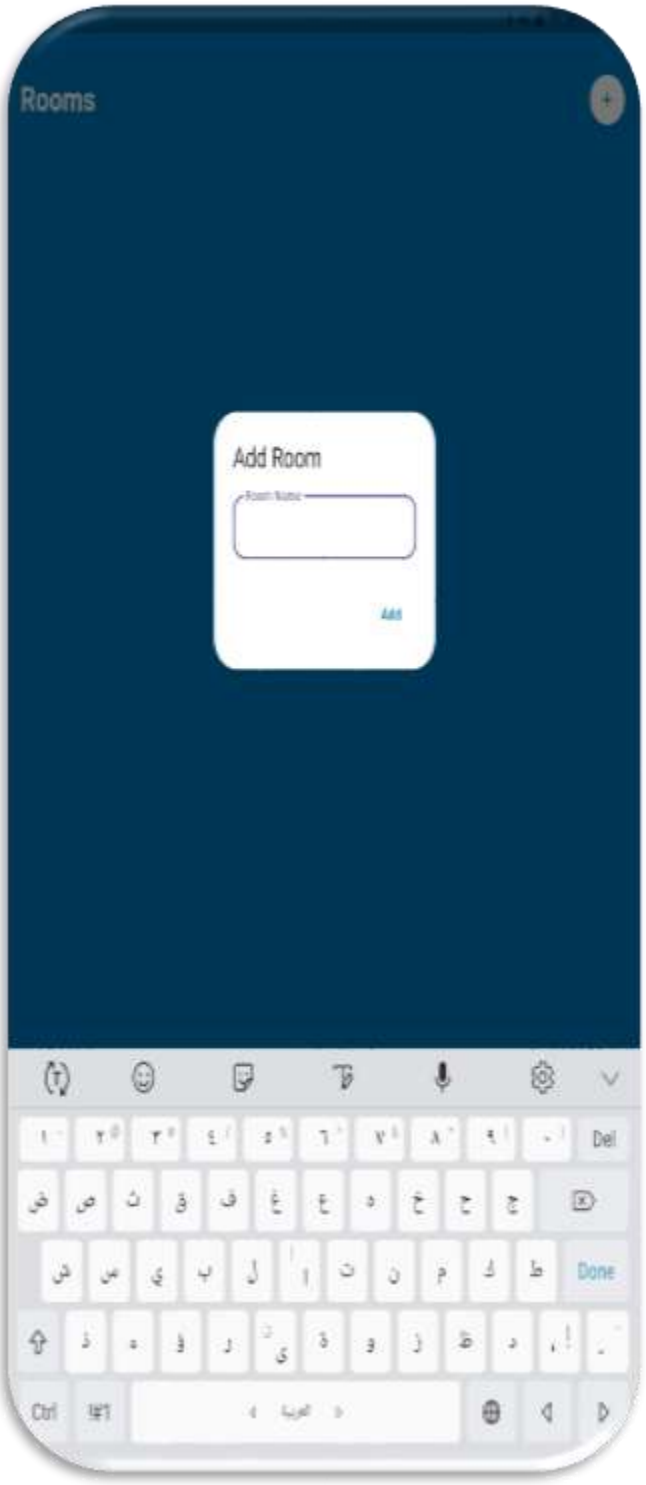
Channels هي الأداة اللي بتستخدمها الـ Goroutines عشان **تتكلم مع بعضها بأمان** من غير ما يحصل أي تضارب في البيانات، ودي أساس مبدأ **"التواصل عبر تشارك البيانات"**

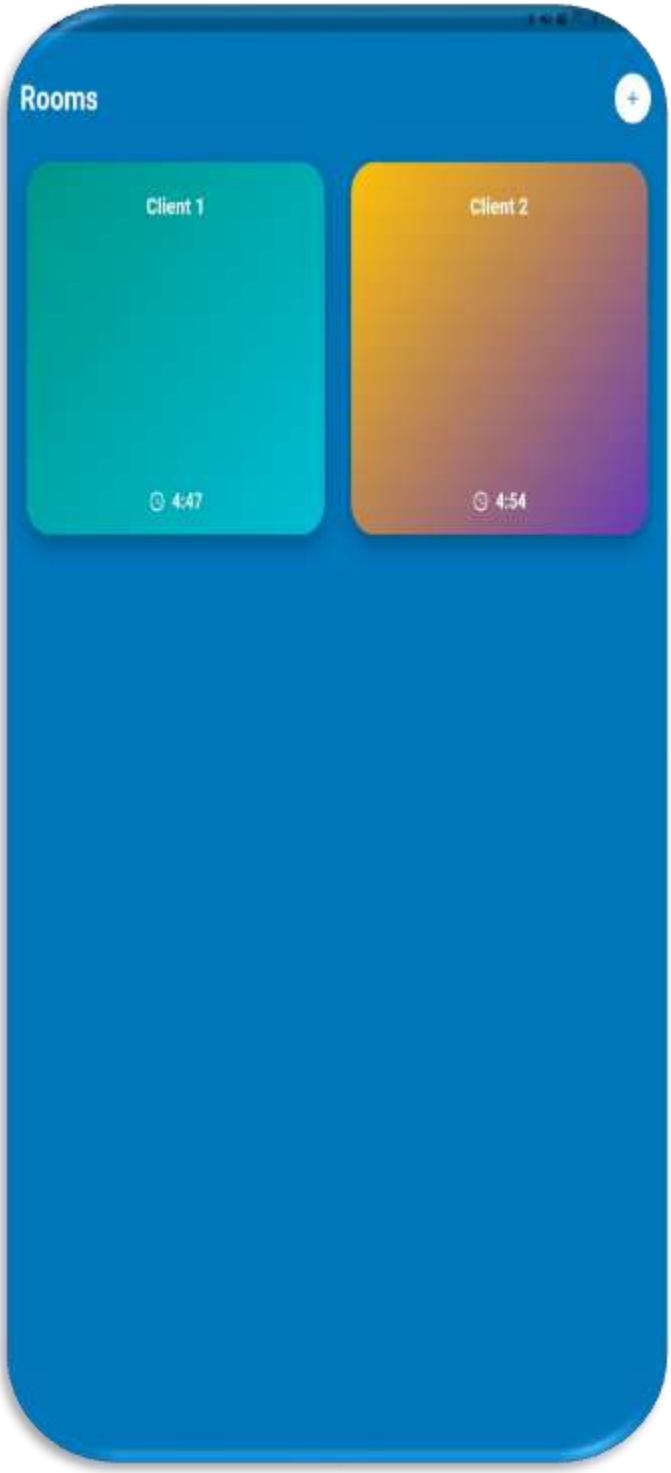
المفهوم	الوصف	استخدامها في المشروع
الـChannels	**"كوبري البيانات"**. هي الطريقة الآمنة لنقل البيانات بين الـGoroutines.	* **للإدارة:** `Register` و `Unregister` بحيث الـ`WSHandler` أو `ReadPump` يبعتوا عميل جديد أو عميل عايز يمشي للغرفة بأمان.
وظيفتها الأساسية	تزامن (Synchronization) وتبادل البيانات (Data Exchange). بتخلي البيانات تتنقل بطريقة منظمة ومحمية، بدل ما الـGoroutines تخش تعدل على نفس الذاكرة مباشرة.	* **للبث:** `Broadcast` قناة استقبال الرسائل للغرفة.
القنوات المُخزَّنة	قنوات ليها سعة تخزين (Buffer) زي `send` و `Broadcast` (سعتهم 256).	* **للإرسال:** قناة `send` الخاصة بكل عميل، عشان الغرفة تبعتلها رسائل الإخراج اللي العميل هيشوفها.

Give A look on APP






Rooms
Add Room
Room Name
Add
Del
Done
Ctrl


Rooms
Client 1
4:47
Client 2
4:54