StudyRoom Live

A real-time study room platform with presence, chat, shared timers/tasks, and an event-driven backend that can scale.

What you'll implement (minimum viable but impressive)

Service 1: API (REST) – ASP.NET Core

- Auth: register/login, JWT, refresh tokens (optional but nice)
- Rooms: create/join/leave, roles (owner/mod/member)
- Persistence: message history + room "artifacts"
    - Tasks (title, status, assignee)
    - Pomodoro timer sessions (start/stop, durations, who started)

Service 2: Realtime Gateway (WebSocket) – ASP.NET Core

- Clients connect with JWT
- Join room channels
- Live events:
    - presence: join/leave/online list
    - typing indicator
    - new messages
    - task updates
    - timer updates (tick events or "state updates" every second)

Service 3: Worker – .NET background service

- Consumes events from RabbitMQ
- Persists events to Postgres (messages, task changes, timer sessions)
- Idempotency: dedupe on `event_id` so reconnects/retries don't duplicate writes

- Optional: daily summaries per room (email/webhook), cleanup jobs

Data/infra (Docker Compose)

- Postgres
- RabbitMQ
- Redis (recommended for presence + rate limiting + event dedupe, but you can also do dedupe in Postgres)

The “portfolio wow” (things recruiters notice immediately)

1. Event-driven architecture (RabbitMQ)

- Gateway publishes `RoomEvent` -> Worker persists -> Gateway broadcasts
- Lets you scale gateway horizontally later

2. Idempotency + retries

- Every event has a GUID `event_id`
- Worker stores processed IDs (Redis SET with TTL or DB table with unique index)
- On retry: safe no-ops instead of duplicates

3. Rate limiting + abuse protection

- Per-user message rate: e.g. 5 msgs/second burst, then cooldown
- Do it in gateway using Redis token bucket (simple implementation)

4. Observability

- Structured logging (Serilog)
- Health checks: `/health` for each service
- Basic metrics endpoint (even just counters + latency logs is enough)

Concrete API sketch (so you can build fast)

REST endpoints (API service)

- POST `/auth/register`
- POST `/auth/login`
- POST `/rooms`
- POST `/rooms/{roomId}/join`
- POST `/rooms/{roomId}/leave`
- GET `/rooms/{roomId}`
- GET `/rooms/{roomId}/messages?cursor=...`
- POST `/rooms/{roomId}/tasks`
- PATCH `/rooms/{roomId}/tasks/{taskId}`
- POST `/rooms/{roomId}/pomodoro/start`
- POST `/rooms/{roomId}/pomodoro/stop`

WebSocket messages (Gateway)
Client -> server:

- `join_room { roomId }`
- `typing { roomId, isTyping }`
- `send_message { roomId, text }`
- `task_update { roomId, taskId, patch }`
- `timer_action { roomId, action }`

Server -> client:

- `presence_update { roomId, usersOnline[] }`
- `message_new { message }`
- `task_updated { task }`
- `timer_state { roomId, state }`
- `error { code, message }`

Event schema (RabbitMQ)
All events include:

- `event_id` (GUID)
- `event_type` (string)
- `room_id`
- `user_id`
- `timestamp`
- `payload` (object)

Pick a simple exchange like `room.events` with routing key `room.{roomId}`.

Recommended tech choices in .NET

- .NET 8, minimal APIs or controllers (either is fine)
- EF Core + Npgsql
- Serilog
- HealthChecks

- RabbitMQ.Client
- Redis: StackExchange.Redis (optional but strong)

What your README/demo should highlight

- “Run locally” in one command: `docker compose up`
- Architecture diagram (even a simple one)
- A 1–2 minute demo:
    1. Two browser tabs join same room
    2. Presence + typing updates
    3. Send messages, refresh to show persistence
    4. Stop worker container, send messages, restart worker, show no duplicates (idempotency)

1.