Backend Handoff: Basic Live Chat (Cross-Device)

Goal:
- Need Help user and Offer Help user can chat in real time across devices.
- Keep implementation simple and beginner-friendly.

1) Copy/paste SQL (Supabase)

```sql
create extension if not exists "uuid-ossp";

create table if not exists public.chat_conversations (
  id uuid primary key default uuid_generate_v4(),
  need_help_post_id uuid not null,
  need_help_user_id uuid not null references public.app_users(id),
  offer_help_user_id uuid not null references public.app_users(id),
  created_at timestamptz not null default now()
);

create table if not exists public.chat_messages (
  id uuid primary key default uuid_generate_v4(),
  conversation_id uuid not null references public.chat_conversations(id) on dele
te cascade,
  sender_user_id uuid not null references public.app_users(id),
  body text not null,
  created_at timestamptz not null default now()
);

create index if not exists idx_chat_messages_conversation_created
  on public.chat_messages(conversation_id, created_at desc);
```

Also append these tables to:
- `backend/src/main/resources/db/create_tables.sql`

2) Endpoints to create

A) Create/Get conversation
- `POST /api/chat/conversations`
- Body:

```json
{
  "needHelpPostId": "post-uuid",
  "needHelpUserId": "need-help-user-uuid",
  "offerHelpUserId": "offer-help-user-uuid"
}
```

- Behavior:
  - If conversation already exists for same post + same users, return it.
  - Else create new one and return it.

B) Get messages
- `GET /api/chat/conversations/{conversationId}/messages`
- Return oldest -> newest.

C) Send message
- `POST /api/chat/conversations/{conversationId}/messages`
- Body:

```json
```

```
{ "body": "Hola, puedo ayudarte mañana" }
```

- `sender_user_id` must come from authenticated user (auth principal), not body.

3) Response examples

Conversation response

```json
{
  "id": "conversation-uuid",
  "needHelpPostId": "post-uuid",
  "needHelpUserId": "need-help-user-uuid",
  "offerHelpUserId": "offer-help-user-uuid",
  "createdAt": "2026-02-14T12:00:00.000Z"
}
```

Message response

```json
{
  "id": "message-uuid",
  "conversationId": "conversation-uuid",
  "senderUserId": "user-uuid",
  "body": "Hola, puedo ayudarte mañana",
  "createdAt": "2026-02-14T12:05:00.000Z"
}
```

Error format

```json
{ "error": "Unauthorized" }
```

```json
{ "error": "Forbidden" }
```

```json
{ "error": "Validation failed" }
```

4) Security rules (minimum)

For both `GET messages` and `POST message`:
- allow only if authenticated user is either:
  - `need_help_user_id` OR
  - `offer_help_user_id`

For `POST message`:
- always set `sender_user_id` from auth principal.

5) Real-time (simple)

Use Supabase Realtime on `chat_messages` table.
Frontend subscribes to INSERT events filtered by `conversation_id`.

When new message is inserted:
- both users get event instantly.

- UI appends message without refresh.

6) Quick test checklist

- [ ] Create conversation endpoint works.
- [ ] Same request returns existing conversation (not duplicate).
- [ ] Get messages returns empty list initially.
- [ ] Send message stores row in `chat_messages`.
- [ ] Other user receives realtime event.
- [ ] Non-participant user gets 403/unauthorized.

7) Frontend expectations

Frontend will need these operations in this order:
1. Create/get conversation.
2. Load messages.
3. Subscribe to realtime new messages.
4. Send message.

That is enough for a first working cross-device live chat.