Commonality — PRD

> **Last updated:** 2026-02-25

1. Vision

Commonality removes the language barrier from real-time conversation. Users chat and call in their native language while every other participant reads and hears the conversation in theirs. The name reflects the product's core belief: understanding is what we all have in common.

2. Problem Statement

Billions of people speak different languages, yet most messaging and voice apps require participants to share a common language. Existing translation tools are copy-paste workflows that break conversational flow. There is no mainstream app where two people can have a natural, real-time conversation — text or voice — each in their own language, without friction.

3. Target Users

Persona	Description
Multilingual families	Grandparents, relatives, or in-laws who speak a different language from younger family members
Global remote teams	Colleagues collaborating across language barriers without waiting for a human translator
Travelers & expats	People connecting with locals, landlords, or service providers in a foreign country
Language learners	Users who want to communicate naturally while seeing translations of their own messages

4. Core Principles

- **Invisible translation** — Translation is not a feature the user invokes; it happens automatically in the background. Each user sees their own language natively.

- **Dual-write, not on-read** — Messages are translated and stored at write time, not rendered on-the-fly. Each user's message history is permanently in their language.
- **Voice is first-class** — Voice translation is not an afterthought. The STT → translate → TTS pipeline runs in real time so users can have a spoken conversation across languages.
- **Simple by default** — 1:1 conversations, no channels, no threads, no reactions. Ship the essential experience first.

---

5. Features — Current State (MVP)

5.1 Authentication

Capability	Details
Registration	Username, password (Argon2id hash), first name, last name, native language
Login	Username + password → JWT (24-hour TTL, HS256)
Session	JWT stored in `localStorage`; validated on each protected page load via `GET /api/auth/me`
Logout	Client-side token removal; no server-side revocation
Security guard	Server refuses to start if `JWT_SECRET` is the default value in non-development environments

Supported languages (signup dropdown): English, Spanish, French, German, Portuguese, Chinese, Japanese, Korean, Arabic, Hindi (10 total, BCP 47 codes).

5.2 Text Chat

Capability	Details
Create conversation	Enter another user's exact username; idempotent (returns existing chat if one exists); cannot chat with yourself
Chat inbox	Lists all conversations sorted by most recent activity; shows other user's name, last message preview (100 chars), and relative timestamp
Real-time messaging	Single persistent WebSocket per user session; messages delivered instantly via Redis pub/sub
Automatic translation	Each message stored twice: once in sender's language, once in recipient's language (via OpenAI). Skipped when both users share the same language
Message history	Cursor-based pagination, 50 messages per page, newest first. "Load older messages" button for earlier history

Capability	Details
Message display	Sent messages: right-aligned, indigo→pink gradient. Received: left-aligned, glassmorphism bubble. Timestamps per message

5.3 Voice Calls

Capability	Details
Initiate call	"Call" button in chat header navigates to voice room
Room infrastructure	LiveKit room named `chat-{chatId}` ; backend issues 1-hour access tokens
Translation pipeline	Server-side agent: ElevenLabs Scribe v2 STT (realtime WebSocket) → OpenAI translation → ElevenLabs TTS (streaming WebSocket) → audio published to LiveKit room
Trigger	Only committed/final STT transcript segments trigger translation (not partials)
Same-language optimization	No translation when source and target language match
Agent lifecycle	Joins room on first token request; auto-exits 5 seconds after room empties
Audio mode	Audio-only; mic toggle and leave button shown; camera/screenshare disabled
Disconnect	Navigates back to the text chat conversation

5.4 UI/UX

- **Dark mode only** with glassmorphism design (frosted glass cards, translucent inputs)
- **Color palette:** indigo (#6366f1) → violet (#8b5cf6) → pink (#ec4899) gradients
- **Logo:** Overlapping speech bubble SVG (indigo left bubble with sound waves, pink right bubble with connection dots) + "commonality" wordmark
- **Font:** Inter (Google Fonts)
- **Responsive layout:** centered cards on auth pages, full-width chat on conversation pages
- **Sticky navbar** on authenticated pages with logo, user info, and sign-out

---

6. User Flows

6.1 New User

Landing page (/) → Click "Sign up" → Fill form (username, password, name, language)
→ Submit → Auto-login → Redirected to /chat (empty inbox)
→ Click "New Chat" → Enter friend's username → Submit
→ Redirected to /chat/{chatId} → Type message → Send
→ Message appears in sender's language; recipient sees it in theirs

6.2 Returning User

Landing page (/) → Enter credentials → Log in → /chat (inbox)
→ Click a conversation → Last 50 messages load → Optionally load older
→ Send messages in real time

6.3 Voice Call

From /chat/{chatId} → Click "Call" → Navigate to /voice/{chatId}
→ Backend starts translation agent + issues LiveKit token
→ Browser requests mic permission → LiveKit room connects
→ Speak in native language → Other participant hears translated audio
→ Click "Leave" → Navigate back to /chat/{chatId}

---

7. Technical Architecture

7.1 Stack

Layer	Technology
Frontend	Next.js 14 (App Router), TypeScript, Tailwind CSS
Backend	Python 3.12, FastAPI, Uvicorn
Database	DynamoDB (Local for dev, AWS for production)
Cache / Pub-sub	Redis 7
Chat transport	WebSockets (FastAPI native)
Text translation	OpenAI API (gpt-4o-mini default)
Voice rooms	LiveKit (self-hosted dev, Cloud for production)
Voice STT	ElevenLabs Scribe v2 (realtime WebSocket)
Voice TTS	ElevenLabs TTS (streaming WebSocket)
Containerization	Docker + Docker Compose (6 services)

7.2 Backend Architecture

Single FastAPI monolith with four modules:

```
app/
├── auth/         # Registration, login, JWT, password hashing
├── chat/         # REST endpoints + WebSocket handler + translation
├── voice/        # LiveKit tokens + STT→translate→TTS pipeline
└── db/           # DynamoDB table creation + Redis client wrapper
```

Key patterns:

- Config via pydantic-settings (`app/config.py`) — all values from environment variables
- Singleton DynamoDB resource and Redis client (`app/dependencies.py`)
- JWT auth via `Authorization: Bearer` header (REST) or `?token=` query param (WebSocket)
- Tables auto-created on startup with `ResourceInUseException` handling for idempotency

7.3 Frontend Architecture

Next.js App Router with 5 pages:

```
app/
├── page.tsx                # Login
├── signup/page.tsx         # Registration
├── chat/
│   ├── page.tsx            # Chat inbox
│   └── [chatId]/page.tsx # Conversation
└── voice/
    └── [roomId]/page.tsx # Voice call
```

Key patterns:

- `useAuth()` hook protects pages — validates JWT on mount, redirects to `/` on failure
- `useWebSocket()` hook manages a single persistent WebSocket connection with message queuing during initial connect
- `api.ts` wraps `fetch` with automatic `Authorization` header injection and error extraction
- UI primitives (`Card` , `Button` , `Input` , `Select` , `Alert`) follow shadcn/ui patterns with glassmorphism styling

7.4 DynamoDB Schema

Table: `users`

PK	SK	GSI1PK	GSI1SK	Attributes
`USER#{userId}`	`PROFILE`	`USERNAME#{username}`	`PROFILE`	userId, username, firstName, lastName, nativeLanguage, passwordHash, createdAt

GSI1 enables lookup by username (login, uniqueness check).

Table: `chats`

PK	SK	Attributes
`CHAT#{chatId}`	`META`	chatId, memberUserIds (List), createdAt

Table: `user_chats`

PK	SK	Attributes
`USER#{userId}`	`CHAT#{chatId}`	chatId, otherUsername, otherUserId, lastMessagePreview, updatedAt

Two entries per chat (one per participant). Queried by userId for inbox listing.

Table: `messages`

PK	SK	Attributes
`USER#{userId}#CHAT#{chatId}`	`MSG#{timestamp}#{msgId}`	messageId, text, fromUserId, language, timestamp

Two entries per message (sender's language + recipient's language). Sort key gives chronological ordering. Cursor-based pagination via `ExclusiveStartKey` .

7.5 API Endpoints

Auth (`/api/auth`)

Method	Path	Auth	Description
POST	`/signup`	None	Register → returns JWT
POST	`/login`	None	Authenticate → returns JWT
GET	`/me`	Bearer	Current user profile

Chat (`/api/chats`)

Method	Path	Auth	Description
POST	`/`	Bearer	Create 1:1 chat by username
GET	`/`	Bearer	List user's conversations
GET	`/{chat_id}/messages`	Bearer	Paginated message history

WebSocket (`/api/ws`)

Protocol	Path	Auth	Description
WS	`/chat?token=JWT`	Query param	Real-time messaging

Voice (`/api/voice`)

Method	Path	Auth	Description
POST	`/token`	Bearer	Get LiveKit room token + start translation agent

Health

Method	Path	Description
GET	`/api/health`	Returns `{"status": "ok"}`

7.6 WebSocket Protocol

Client → Server:

```
{ "chat_id": "<uuid>", "text": "Hello!" }
```

Server → Client (message delivered):

```
{
  "type": "message",
  "chat_id": "<uuid>",
  "message": {
    "message_id": "<uuid>",
    "text": "...",
    "from_user_id": "<uuid>",
    "language": "en",
    "timestamp": "2026-02-25T01:00:00+00:00"
  }
}
```

Server → Client (error):

```
{ "error": "chat_id and text are required" }
```

7.7 Infrastructure

Local development: Docker Compose with 6 services (backend, frontend, DynamoDB Local, DynamoDB Admin, Redis, LiveKit dev server). Hot-reload via volume mounts.

Production deployment (planned):

Service	Platform
Frontend	Vercel
Backend	Railway
Database	AWS DynamoDB (PAY_PER_REQUEST)
Cache/Pub-sub	Upstash Redis (serverless, TLS)
Voice	LiveKit Cloud

See deploymentPlan.md for full deployment instructions.

---

8. Configuration

Backend Environment Variables

Variable	Default	Description
`OPENAI_API_KEY`	—	OpenAI API key
`OPENAI_TRANSLATION_MODEL`	`gpt-4o-mini`	Translation model
`LIVEKIT_API_KEY`	`devkey`	LiveKit API key
`LIVEKIT_API_SECRET`	`devsecret`	LiveKit API secret
`LIVEKIT_URL`	`ws://livekit:7880`	LiveKit server URL
`ELEVENLABS_API_KEY`	—	ElevenLabs API key
`ELEVENLABS_TTS_VOICE_ID`	`Xb7hH8MSUJpSbSDYk0k2`	TTS voice

Variable	Default	Description
`ELEVENLABS_TTS_MODEL`	`eleven_flash_v2_5`	TTS model
`DYNAMODB_ENDPOINT`	`None`	DynamoDB endpoint (set for local dev, omit for AWS)
`AWS_REGION`	`us-east-1`	AWS region
`AWS_ACCESS_KEY_ID`	`local`	AWS credentials
`AWS_SECRET_ACCESS_KEY`	`local`	AWS credentials
`REDIS_URL`	`redis://redis:6379/0`	Redis connection URL
`JWT_SECRET`	`change-me-in-production`	JWT signing key
`JWT_EXPIRATION_MINUTES`	`1440`	Token TTL (24h)
`PASSWORD_MIN_LENGTH`	`8`	Minimum password length
`CORS_ORIGINS`	`http://localhost:3000`	Allowed origins (comma-separated)

Frontend Environment Variables

Variable	Default	Description
`NEXT_PUBLIC_API_URL`	`http://localhost:8080`	Backend HTTP URL
`NEXT_PUBLIC_WS_URL`	`ws://localhost:8080`	Backend WebSocket URL
`NEXT_PUBLIC_LIVEKIT_URL`	`ws://localhost:7880`	LiveKit server URL
`NEXT_PUBLIC_PASSWORD_MIN_LENGTH`	`8`	Signup validation

---

9. Known Limitations

Area	Limitation
Chat model	1:1 only — no group conversations
Test coverage	Zero automated tests (test files are empty stubs)

Area	Limitation
WebSocket resilience	No reconnection logic; messages lost if connection drops
Token management	No refresh tokens; JWT in localStorage (XSS risk); no server-side revocation
User discovery	Must know exact username — no search or suggestions
Profile management	No way to update name, language, or password after registration
Message features	No edit, delete, read receipts, typing indicators, or unread counts
Voice — single voice	All TTS output uses one hardcoded voice regardless of speaker
Voice — scaling	Translation agent tracked in-process memory; not horizontally scalable
Rate limiting	None on any endpoint — no brute-force or abuse protection
Message length	No max length enforced — uncapped translation API costs
Notifications	No push or in-app notifications for new messages
Offline support	None — requires active connection
Language list	Hardcoded to 10 languages in the frontend dropdown

---

10. Future Roadmap

P0 — Production Readiness

Feature	Description
Production deployment	Deploy to Vercel + Railway + AWS DynamoDB + Upstash Redis + LiveKit Cloud (see deploymentPlan.md)
WebSocket reconnection	Automatic reconnect with exponential backoff; queue and flush messages on reconnect
Rate limiting	Per-IP and per-user throttling on auth endpoints, message sending, and API calls
Backend test suite	Unit and integration tests for auth, chat, voice, and DynamoDB operations
Message length limits	Enforce max character count to cap translation costs

P1 — Core Experience

Feature	Description

Feature	Description
Unread message counts	Badge on chat list items; clear on conversation open
Typing indicators	Show when the other user is typing via WebSocket
Read receipts	"Seen" status on messages
User search	Search users by name or username with autocomplete
Profile management	Edit name, language, password; add avatar
Push notifications	Browser notifications for new messages when not on the chat page
Per-speaker TTS voice	Select or auto-assign distinct voices per participant in voice calls

P2 — Growth Features

Feature	Description
Group chat	Conversations with 3+ participants; fan-out translation to each member's language
Group voice calls	Multi-party voice rooms with per-participant translation pipelines
Media messages	Send images, files, and voice notes with optional caption translation
Message reactions	Emoji reactions on messages
Message search	Full-text search across conversation history
Language auto-detect	Detect language from message content instead of relying solely on profile setting

P3 — Platform & Scale

Feature	Description
Mobile apps	React Native or native iOS/Android clients
Token refresh	Refresh tokens with HttpOnly cookies; server-side session management
Horizontal voice scaling	Move translation agent state to Redis or a dedicated service
Message encryption	End-to-end encryption with per-device keys
Analytics dashboard	Usage metrics, translation volume, active users
Admin panel	User management, content moderation, system health
Expanded language support	Dynamic language list from backend; support for 50+ languages

Feature	Description
Custom TTS voices	Let users choose or clone their own TTS voice

---

11. Success Metrics

Metric	Target
Translation latency (text)	< 1 second from send to delivery in recipient's language
Translation latency (voice)	< 3 seconds from speech to translated audio playback
WebSocket uptime	99.9% connection availability during active sessions
Translation accuracy	User-reported satisfaction > 90% for supported language pairs
Onboarding completion	> 80% of signups send their first message within 5 minutes