AI Companion Backend Starter

This canvas contains: 1) **OpenAPI 3.1 spec skeleton** (editable YAML) 2) **Go monorepo folder structure** + minimal boot code & interfaces

> Scope: Text + Voice convo, memory, voice profiles, realtime. Trim/add as you need.

---

1) OpenAPI 3.1 Spec — Skeleton (`/api/openapi.yaml`)

```yaml
openapi: 3.1.0
info:
  title: AI Companion API
  version: 0.1.0
  description: Backend for realtime AI companion with STT/TTS/LLM + memory
servers:
  - url: https://api.example.com/v1
security:
  - bearerAuth: []

paths:
  /auth/signup:
    post:
      summary: Create account
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SignupRequest'
      responses:
        '201': { $ref: '#/components/responses/CreatedUser' }
        '400': { $ref: '#/components/responses/BadRequest' }

  /auth/login:
    post:
      summary: Login and obtain tokens
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/LoginRequest'
      responses:
        '200': { $ref: '#/components/responses/Tokens' }
        '401': { $ref: '#/components/responses/Unauthorized' }

  /auth/refresh:
    post:
      summary: Refresh access token
      responses:
        '200': { $ref: '#/components/responses/Tokens' }

  /me:
    get:
      summary: Current user
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema: { $ref: '#/components/schemas/User' }

  /voice/profiles:
    post:
      summary: Create/Upload a voice profile
      requestBody:
        required: true
        content:
          multipart/form-data:
            schema:
              type: object
              properties:
                name: { type: string }
                provider: { type: string, enum: [elevenlabs, azure, playht,
local] }
                sample: { type: string, format: binary }
      responses:
        '201':
          description: Created
          content:
            application/json:
              schema: { $ref: '#/components/schemas/VoiceProfile' }

  /voice/profiles/{id}:
    get:
      summary: Get a voice profile
      parameters:
        - in: path
          name: id
          required: true
          schema: { type: string }
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema: { $ref: '#/components/schemas/VoiceProfile' }
```

```yaml
/conversations:
  post:
    summary: Start a conversation session
    requestBody:
      required: true
      content:
        application/json:
          schema:
            type: object
            properties:
              mode: { type: string, enum: [text, voice] }
              persona: { type: string }
    responses:
      '201': { $ref: '#/components/responses/CreatedConversation' }

/conversations/{id}/messages:
  get:
    summary: List messages in a conversation
    parameters:
      - in: path
        name: id
        required: true
        schema: { type: string }
    responses:
      '200':
        description: OK
        content:
          application/json:
            schema:
              type: array
              items: { $ref: '#/components/schemas/Message' }

/chat:
  post:
    summary: Send a text message and get AI reply
    requestBody:
      required: true
      content:
        application/json:
          schema: { $ref: '#/components/schemas/ChatRequest' }
    responses:
      '200': { $ref: '#/components/responses/ChatResponse' }

/chat:stream:
  post:
    summary: Stream tokens (SSE)
    description: Emits event-stream of assistant tokens
    responses:
      '200':
        description: text/event-stream
  /memories:
    post:
      summary: Create a pinned memory
      requestBody:
        required: true
        content:
          application/json:
            schema: { $ref: '#/components/schemas/Memory' }
      responses:
        '201': { description: Created }

  /memories/search:
    get:
      summary: Semantic memory search
      parameters:
        - in: query
          name: q
          schema: { type: string }
        - in: query
          name: k
          schema: { type: integer, default: 5 }
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: array
                items: { $ref: '#/components/schemas/Memory' }

components:
  securitySchemes:
    bearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

  responses:
    CreatedUser:
      description: Created
      content:
        application/json:
          schema: { $ref: '#/components/schemas/User' }
    Tokens:
      description: Access/Refresh tokens
      content:
        application/json:
          schema:
            type: object
            properties:
              access_token: { type: string }
            refresh_token: { type: string }
    BadRequest:
      description: Bad Request
    Unauthorized:
      description: Unauthorized

  schemas:
    User:
      type: object
      properties:
        id: { type: string }
        email: { type: string }
        display_name: { type: string }
        created_at: { type: string, format: date-time }

    VoiceProfile:
      type: object
      properties:
        id: { type: string }
        user_id: { type: string }
        provider: { type: string }
        voice_id: { type: string }
        status: { type: string, enum: [pending, ready, failed] }
        created_at: { type: string, format: date-time }

    Conversation:
      type: object
      properties:
        id: { type: string }
        user_id: { type: string }
        mode: { type: string, enum: [text, voice] }
        persona: { type: string }
        created_at: { type: string, format: date-time }

    Message:
      type: object
      properties:
        id: { type: string }
        conversation_id: { type: string }
        role: { type: string, enum: [user, assistant, system] }
        text: { type: string }
        audio_url: { type: string, format: uri }
        tokens_in: { type: integer }
        tokens_out: { type: integer }
        latency_ms: { type: integer }
        created_at: { type: string, format: date-time }

    ChatRequest:
      type: object
      required: [conversation_id, text]
      properties:
        conversation_id: { type: string }
        text: { type: string }
        tts_voice_id: { type: string }
        stream: { type: boolean, default: false }

    ChatReply:
      type: object
      properties:
        message: { $ref: '#/components/schemas/Message' }

    Memory:
      type: object
      properties:
        id: { type: string }
        user_id: { type: string }
        conversation_id: { type: string, nullable: true }
        kind: { type: string, enum: [profile, fact, summary] }
        text: { type: string }
        embedding: { type: array, items: { type: number }, description:
'Vector omitted in responses' }
        relevance: { type: number }
        created_at: { type: string, format: date-time }
```

Note: **Realtime WebSocket/WebRTC** endpoints typically documented outside OpenAPI or via AsyncAPI. Keep `/realtime` described in README.

---

2) Go Monorepo — Folder Structure

```
ai-companion/
├── api/
│   └── openapi.yaml
├── cmd/
│   └── gateway/
│       └── main.go
├── internal/
│   ├── http/
│   │   ├── router.go
│   │   ├── handlers/
│   │   │   ├── auth.go
│   │   │   ├── chat.go
│   │   │   ├── conversations.go
│   │   │   ├── voice.go
│   │   │   └── memories.go
│   │   └── middleware/
│   │       ├── auth.go
│   │       └── logging.go
│   ├── config/
│   │   └── config.go
│   ├── logger/
│   │   └── logger.go
│   ├── persistence/
│   │   ├── sql/
│   │   │   ├── db.go
│   │   │   └── migrations/
│   │   ├── redis/
│   │   │   └── client.go
│   │   └── vector/
│   │       └── qdrant.go
│   └── realtime/
│       ├── ws.go
│       └── webrtc.go (optional later)
├── pkg/
│   ├── llm/
│   │   ├── llm.go (interface)
│   │   └── openai.go
│   ├── stt/
│   │   ├── stt.go (interface)
│   │   └── whisper.go
│   ├── tts/
│   │   ├── tts.go (interface)
│   │   └── elevenlabs.go
│   ├── memory/
│   │   └── engine.go
│   └── util/
│       └── errs.go
├── go.mod
├── go.work (if multi-module)
├── Makefile
└── deploy/
    ├── docker-compose.yml
    └── k8s/
        ├── deployment.yaml
        └── service.yaml
```

Minimal Boot Code — `cmd/gateway/main.go`

```go
package main

import (
    "log"
    "net/http"

    "github.com/labstack/echo/v4"
    "github.com/labstack/echo/v4/middleware"
    "ai-companion/internal/http/handlers"
    mw "ai-companion/internal/http/middleware"
)

func main() {
    e := echo.New()
    e.HideBanner = true
    e.Use(middleware.Recover())
    e.Use(middleware.Logger())

    // JWT middleware (plug your secret)
    e.Use(mw.JWTOptional())

    // Routes
    api := e.Group("/v1")
    handlers.RegisterAuth(api)
    handlers.RegisterConversations(api)
    handlers.RegisterChat(api)
    handlers.RegisterVoice(api)
    handlers.RegisterMemories(api)

    log.Fatal(e.Start(":8080"))
}
```

Router Helpers — `internal/http/router.go`

```go
package http

// keep empty for now (helpers for mounting groups, versioning, etc.)
```

Middleware — `internal/http/middleware/auth.go`

```go
package middleware

import (
    "net/http"
    "github.com/labstack/echo/v4"
)

func JWTOptional() echo.MiddlewareFunc {
    return func(next echo.HandlerFunc) echo.HandlerFunc {
        return func(c echo.Context) error {
            // TODO: parse Authorization, attach user to context if valid
            return next(c)
        }
    }
}
```

Handlers — Auth (skeleton) `internal/http/handlers/auth.go`

```
package handlers

import (
    "net/http"
    "github.com/labstack/echo/v4"
)

type signupReq struct { Email, Password, DisplayName string }

type tokensRes struct { AccessToken string `json:"access_token"`;
RefreshToken string `json:"refresh_token"` }

func RegisterAuth(g *echo.Group) {
    g.POST("/auth/signup", signup)
    g.POST("/auth/login", login)
    g.POST("/auth/refresh", refresh)
}

func signup(c echo.Context) error { return c.NoContent(http.StatusCreated) }
func login(c echo.Context) error  { return c.JSON(http.StatusOK,
tokensRes{}) }
func refresh(c echo.Context) error{ return c.JSON(http.StatusOK,
tokensRes{}) }
```

Handlers — Conversations `internal/http/handlers/conversations.go`

```
package handlers

import (
    "net/http"
    "github.com/labstack/echo/v4"
)

type startConvReq struct { Mode string `json:"mode"`; Persona string
`json:"persona"` }

type conversation struct { ID, Mode, Persona string }

func RegisterConversations(g *echo.Group) {
    g.POST("/conversations", startConv)
    g.GET("/conversations/:id/messages", listMessages)
}

func startConv(c echo.Context) error   { return c.JSON(http.StatusCreated,
conversation{ID: "conv_123"}) }
func listMessages(c echo.Context) error{ return c.JSON(http.StatusOK,
[]any{}) }
```

Handlers — Chat `internal/http/handlers/chat.go`

```
package handlers

import (
    "net/http"
    "github.com/labstack/echo/v4"
)

type chatReq struct { ConversationID, Text, TTSVoiceID string; Stream bool }

type message struct { ID, Role, Text string }

type chatRes struct { Message message }

func RegisterChat(g *echo.Group) {
    g.POST("/chat", chat)
    g.POST("/chat:stream", chatStream) // Implement SSE later
}

func chat(c echo.Context) error {
    // TODO: call memory → llm → tts (optional) and persist message
    return c.JSON(http.StatusOK, chatRes{Message: message{ID: "msg_1", Role:
"assistant", Text: "Hello Irfan!"}})
}

func chatStream(c echo.Context) error { return
c.NoContent(http.StatusNotImplemented) }
```

Handlers — Voice `internal/http/handlers/voice.go`

```
package handlers

import (
    "net/http"
    "github.com/labstack/echo/v4"
)

type voiceProfile struct { ID, Provider, VoiceID, Status string }

func RegisterVoice(g *echo.Group) {
    g.POST("/voice/profiles", createVoice)
    g.GET("/voice/profiles/:id", getVoice)
}

func createVoice(c echo.Context) error { return c.JSON(http.StatusCreated,
voiceProfile{ID: "vp_1", Provider: "elevenlabs", VoiceID: "v_123", Status:
"pending"}) }
func getVoice(c echo.Context) error    { return c.JSON(http.StatusOK,
voiceProfile{ID: "vp_1", Provider: "elevenlabs", VoiceID: "v_123", Status:
"ready"}) }
```

Handlers — Memories `internal/http/handlers/memories.go`

```
package handlers

import (
    "net/http"
    "github.com/labstack/echo/v4"
)

type memory struct { ID, Kind, Text string; Relevance float64 }

func RegisterMemories(g *echo.Group) {
    g.POST("/memories", createMemory)
    g.GET("/memories/search", searchMemories)
}

func createMemory(c echo.Context) error { return
c.NoContent(http.StatusCreated) }
func searchMemories(c echo.Context) error { return c.JSON(http.StatusOK,
[]memory{}) }
```

Interfaces — Adapters (`/pkg`)

LLM Interface `pkg/llm/llm.go`

```
type StreamCallback func(delta string) error

type LLM interface {
    Stream(ctx context.Context, prompt string, tools any, cb StreamCallback)
error
}
```

STT Interface `pkg/stt/stt.go`

```
type PartialCallback func(partial string, isFinal bool) error

type STT interface {
    Stream(ctx context.Context, audio io.Reader, cb PartialCallback) error
}
```

TTS Interface `pkg/tts/tts.go`

```go
type AudioCallback func(chunk []byte) error

type TTS interface {
    Stream(ctx context.Context, text string, voice VoiceProfile, cb
AudioCallback) error
}

type VoiceProfile struct { Provider, VoiceID, Style string }
```

Makefile (snip)

```
run:
    go run ./cmd/gateway

lint:
    golangci-lint run

test:
    go test ./...
```

docker-compose (snip)

```yaml
version: '3.9'
services:
  api:
    build: .
    ports: ["8080:8080"]
    env_file: .env
  db:
    image: postgres:16
    environment:
      POSTGRES_PASSWORD: password
    ports: ["5432:5432"]
  redis:
    image: redis:7
    ports: ["6379:6379"]
  qdrant:
    image: qdrant/qdrant:latest
    ports: ["6333:6333", "6334:6334"]
```

Next Steps (edit me)

- Wire JWT in middleware; add `users` repo in `internal/persistence/sql`.
- Implement memory engine (`/pkg/memory`) with vector search.
- Add SSE to `/chat:stream` and basic WebSocket under `/realtime`.
- Create provider configs for OpenAI/Whisper/ElevenLabs in `/internal/config`.
- Hook logs/metrics (Zap + Prometheus + OTEL). ```