Prompt-Powered Kickstart: Building a Beginner’s Toolkit for GoLang Audio API

1. Title & Objective

Title: Getting Started with GoLang for Building Real-Time Audio Processing APIs
Objective: To create a simple GoLang API that simulates real-time audio processing by accepting an audio signal name and returning a processed response.

Why GoLang? GoLang is lightweight, fast, and great for concurrent tasks like audio streaming, making it ideal for AV and AI-integrated systems. This toolkit is designed to help beginners set up their first GoLang backend for AV-related tasks.

---

2. Quick Summary of the Technology

What is GoLang? GoLang (or Go) is an open-source programming language created by Google. It is known for its simplicity, efficiency, and powerful concurrency features.

Where is it used? GoLang is commonly used for: - Backend web APIs - Cloud services (Docker, Kubernetes) - Audio/Video data pipelines - AI systems integration

Real-world example: Biamp Tesira and other AV DSP systems can integrate backend APIs (like those built in GoLang) for automation or AI-based control systems.

---

3. System Requirements

- **OS:** Linux, macOS, or Windows
- **Tools:** VS Code or any text editor
- **Packages:** Go (version 1.21 or above)

---

4. Installation & Setup Instructions

Step 1: Install Go

Visit https://go.dev/dl/ and download the installer for your operating system.

After installation, verify with:

```
go version
```

Step 2: Create a Project Folder

```
mkdir go-audio-api
cd go-audio-api
```

Step 3: Initialize Go Module

```
go mod init go-audio-api
```

Step 4: Create the Main File

Create a file named `main.go`:

```
touch main.go
```

---

5. Minimal Working Example

Paste the following code into `main.go`:

```go
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "net/http"
)

type AudioRequest struct {
    Signal string `json:"signal"`
}

type AudioResponse struct {
    ProcessedSignal string `json:"processed_signal"`
}

func processAudio(signal string) string {
    return fmt.Sprintf("Processed_%s.wav", signal)
}

func handleAudio(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Use POST method", http.StatusMethodNotAllowed)
        return
    }
    var req AudioRequest
    json.NewDecoder(r.Body).Decode(&req)
    processed := processAudio(req.Signal)
    res := AudioResponse{ProcessedSignal: processed}
    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(res)
}

func main() {
    http.HandleFunc("/process", handleAudio)
    fmt.Println("Server running on port 8080...")
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

Run the API

```
go run main.go
```

Test with curl or Postman:

```
curl -X POST -H "Content-Type: application/json" -d '{"signal":"voice_input"}'
http://localhost:8080/process
```

Expected Output:

```
{"processed_signal":"Processed_voice_input.wav"}
```

---

6. AI Prompt Journal

Prompt	AI Response Summary	Helpfulness (1-5)
"Give me a step-by-step guide to build a GoLang API server"	Provided full setup including go.mod and main.go	5

Prompt	AI Response Summary	Helpfulness (1-5)
"Show a Go function that simulates processing audio input"	Helped build processAudio() logic	4
"Explain how to send and receive JSON in Go"	Guided JSON decoding/encoding logic	5
"Help me test my GoLang API using curl"	Suggested correct curl syntax	5

---

7. Common Issues & Fixes

Issue	Solution
`go: command not found`	Ensure Go is added to PATH. Reinstall from go.dev.
`port already in use`	Stop previous process or use a new port (e.g., 8081).
Invalid JSON input	Verify your POST data structure matches `AudioRequest`.

---

8. References

- Go.dev Documentation
- Moringa AI Learning Platform
- Postman API Testing
- Go JSON Docs

---

9. Reflection & Next Steps

Through generative AI prompting, I rapidly learned to scaffold, debug, and test a GoLang API. The process showed how AI tools accelerate technical onboarding for new languages.

Next Steps: Integrate this with an AV system like Biamp Tesira to send translated or processed audio responses dynamically.

End of Toolkit Document