MCP Server Boilerplate

This repository provides a minimal setup for building an **MCP (Model-Client-Protocol) server**, offering a standardized interface for AI models to interact with backend services through defined tools.

---

1. Overview

The MCP server exposes backend capabilities as tools that can be called by AI models to retrieve data or perform actions. This boilerplate helps you quickly scaffold and deploy an MCP server with customizable tools.

---

2. Setting Up the MCP Server

2.1 Project Structure

```
Unset
/cmd/your-mcp-server/
├── main.go       # Server entry point and tool definitions
├── Makefile      # Build commands
```

2.2 Dependencies

Install the required dependencies:

```
Unset
go get github.com/mark3labs/mcp-go/mcp
go get github.com/mark3labs/mcp-go/server
go get github.com/ilyakaznacheev/cleanenv
```

2.3 Create the MCP Server

In `cmd/your-mcp-server/main.go`:

Unset
```
package main

import (
    "context"
    "flag"
    "fmt"
    "log/slog"
    "os"

    "github.com/ilyakaznacheev/cleanenv"
    "github.com/mark3labs/mcp-go/mcp"
    "github.com/mark3labs/mcp-go/server"

    // Import your service package
    yourservice "github.com/your-org/your-repo/pkg/yourservice"
)

// Config holds server configuration.
type Config struct {
    Host string `env:"HOST" env-default:"localhost"`
    Port uint16 `env:"PORT" env-default:"4000"`
}

func main() {
    var mode = flag.String("mode", "stdio", "Server mode: 'stdio'
or 'sse'")
    flag.Parse()

    var cfg Config
    cleanenv.ReadEnv(&cfg)

    // Initialize your service
    // service := yourservice.NewService()

    s := server.NewMCPServer(
        "Your MCP Server", // Server name
        "1.0.0",           // Version
        server.WithResourceCapabilities(true, true),
    )

    // Define and add tools here

    if *mode == "sse" {
        slog.Info("Starting SSE server", "host", cfg.Host,
"port", cfg.Port)
        s.StartSSE(fmt.Sprintf("%s:%d", cfg.Host, cfg.Port))
    } else {
        slog.Info("Starting stdio server")
        s.StartStdio()
    }
}
```

---

3. Defining and Adding Tools

3.1 Basic Tool Definition

Unset

```
helloTool := mcp.NewTool("hello_world",
    mcp.WithDescription("Say hello to someone"),
    mcp.WithString("name",
        mcp.Required(),
        mcp.Description("Name of the person to greet"),
    ),
)
```

```
s.AddTool(helloTool, func(ctx context.Context, request
mcp.CallToolRequest) (*mcp.CallToolResult, error) {
    name := request.GetStringParam("name")
    result := fmt.Sprintf("Hello, %s!", name)
    return mcp.NewCallToolResult(result), nil
})
```

3.2 Tool with Multiple Parameters

```
Unset
findItemsTool := mcp.NewTool("find_items",
    mcp.WithDescription("Find items by category and date range"),
    mcp.WithString("category", mcp.Required(),
mcp.Description("Category to filter by")),
    mcp.WithString("start_date", mcp.Required(),
mcp.Description("Start date (YYYY-MM-DD)")),
    mcp.WithString("end_date", mcp.Required(),
mcp.Description("End date (YYYY-MM-DD)")),
)

s.AddTool(findItemsTool, findItemsHandler)
```

3.3 Tool with Array Parameter

```
Unset
filterTool := mcp.NewTool("filter_by_ids",
    mcp.WithDescription("Filter items by a list of IDs"),
    mcp.WithArray("ids", mcp.Description("List of item IDs")),
)

s.AddTool(filterTool, filterByIdsHandler)
```

---

4. Implementing Tool Handlers

4.1 Basic Handler

Unset

```
func helloHandler(ctx context.Context, request
mcp.CallToolRequest) (*mcp.CallToolResult, error) {
    name := request.GetStringParam("name")
    result := fmt.Sprintf("Hello, %s!", name)
    return mcp.NewCallToolResult(result), nil
}
```

4.2 Handler with Service Integration

Unset

```
func findItemsHandler(ctx context.Context, request
mcp.CallToolRequest, service *yourservice.Service)
(*mcp.CallToolResult, error) {
    category := request.GetStringParam("category")
    startDate := request.GetStringParam("start_date")
    endDate := request.GetStringParam("end_date")

    start, err := time.Parse("2006-01-02", startDate)
    if err != nil {
        return nil, fmt.Errorf("invalid start date: %w", err)
    }

    end, err := time.Parse("2006-01-02", endDate)
    if err != nil {
        return nil, fmt.Errorf("invalid end date: %w", err)
    }

    items, err := service.FindItems(ctx, category, start, end)
    if err != nil {
        return nil, err
    }

    return mcp.NewCallToolResult(items), nil
}
```

4.3 Handler with Array Parameter

```
Unset
func filterByIdsHandler(ctx context.Context, request
mcp.CallToolRequest, service *yourservice.Service)
(*mcp.CallToolResult, error) {
    idsInterface := request.GetArrayParam("ids")

    var ids []string
    for _, id := range idsInterface {
        if idStr, ok := id.(string); ok {
            ids = append(ids, idStr)
        }
    }

    items, err := service.GetItemsByIds(ctx, ids)
    if err != nil {
        return nil, err
    }

    return mcp.NewCallToolResult(items), nil
}
```

---

5. Running the MCP Server

5.1 Environment Setup

Create a `.env` file:

```
Unset
HOST=localhost
PORT=4000
```

5.2 Run in SSE Mode

```
Unset
go run cmd/your-mcp-server/main.go -mode sse
```

---

6. License

MIT © [Your Company]

Let me know if you'd like to add contribution guidelines, CI/CD setup instructions, or any branding/customization before you publish!