Echo Framework (Go Language): 1-Hour Learning Module

Module Overview

This learning module introduces the **Echo web framework** for Go (Golang), focusing on its essential concepts, features, and hands-on usage. By the end of this session, you will be able to set up Echo, understand its main components, and build a simple web application.

Learning Objectives

- Understand what Echo is and its role in Go web development
- Set up a basic Echo project
- Implement routing and middleware
- Handle JSON requests and responses
- Build a simple RESTful API

Agenda

Time	Topic
0–10 min	Introduction to Echo
10–20 min	Setting Up and Hello World
20–35 min	Routing and Parameters
35–45 min	Middleware and Request Handling
45–55 min	JSON Handling and RESTful API
55–60 min	Q&A / Recap / Further Resources

0–10 min: Introduction to Echo

- **What is Echo?**
  - Echo is a high-performance, minimalist web framework for Go, designed for building scalable and robust web applications.
- **Key Features:**
  - Fast and lightweight HTTP router
  - Middleware support
  - Flexible routing with parameters

- Template rendering
- Data binding and validation
- Extensibility and strong community support

10–20 min: Setting Up and Hello World

Prerequisites

- Go installed on your machine

Installation

```
go get github.com/labstack/echo/v4
```

Hello World Example

```
package main

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

func main() {
    e := echo.New()
    e.GET("/", func(c echo.Context) error {
        return c.String(http.StatusOK, "Hello, World!")
    })
    e.Start(":8080")
}
```

- Run the application and visit `http://localhost:8080` to see "Hello, World!".

20–35 min: Routing and Parameters

Defining Routes

- **GET Route with Parameter:**

```
e.GET("/users/:name", func(c echo.Context) error {
    name := c.Param("name")
    return c.String(http.StatusOK, "Hello, " + name)
})
```

- **POST Route Example:**

```
e.POST("/users", func(c echo.Context) error {
    name := c.FormValue("name")
```

```
    return c.String(http.StatusOK, "User " + name + " created")
})
```

- Test these routes using `curl` or Postman.

35–45 min: Middleware and Request Handling

Using Middleware

- **Logger and Recovery Middleware:**

```
import "github.com/labstack/echo/v4/middleware"

e.Use(middleware.Logger())
e.Use(middleware.Recover())
```

- Middleware can be used for logging, authentication, error handling, and more.

45–55 min: JSON Handling and RESTful API

Handling JSON Data

```
type User struct {
    Name  string `json:"name"`
    Email string `json:"email"`
}

e.POST("/users", func(c echo.Context) error {
    u := new(User)
    if err := c.Bind(u); err != nil {
        return err
    }
    return c.JSON(http.StatusCreated, u)
})
```

- Send a POST request with JSON data to `/users` and Echo will parse and respond with the same data.

55–60 min: Q&A / Recap / Further Resources

Recap

- Echo’s core concepts: setup, routing, middleware, JSON handling
- How to build a simple RESTful API

Further Resources

- Official Echo Documentation
- Community forums and GitHub repository
- Tutorials and real-world examples

Suggested Activities

- Experiment with adding more routes and middleware
- Try integrating template rendering for HTML responses
- Explore Echo’s validation and error handling features

References

- HayaGeek: Golang Echo Tutorial with Practical Examples
- Madalin.me: Introduction to the Echo Web Framework in Go
- Echo, LabStack: Introduction - Echo, LabStack
- Echo LabStack: High performance, extensible, minimalist Go web framework