1-Hour Learning Module: Building a CRUD API with Echo (Go) & Postman

Overview

This module guides you through creating a simple CRUD (Create, Read, Update, Delete) REST API using the Echo framework in Go and testing it with Postman. Designed for a 1-hour hands-on session.

Table of Contents

1. Introduction

- **Echo**: A high-performance, minimalist Go web framework for RESTful APIs.
- **CRUD**: Core data operations—Create, Read, Update, Delete.
- **Postman**: A popular tool for testing and interacting with APIs.

2. Environment Setup

Prerequisites:

- Go (version 1.16+)
- Basic Go knowledge
- Postman installed

Setup Steps:

1. Create a new Go project:

```
mkdir echo-crud-demo && cd echo-crud-demo
go mod init echo-crud-demo
```

2. Install Echo:

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

3. Project Structure

Organize your files as follows:

```
echo-crud-demo/
├── main.go
├── handler/
│   └── user.go
├── model/
│   └── user.go
```

4. Implementing CRUD Operations

a. Model Definition (`model/user.go`)

```
package model

type User struct {
    ID   int    `json:"id"`
    Name string `json:"name"`
}
```

b. Handler Implementation (`handler/user.go`)

```
package handler

import (
    "net/http"
    "strconv"
    "github.com/labstack/echo/v4"
    "echo-crud-demo/model"
)

var users = []model.User{}
var idCounter = 1

func CreateUser(c echo.Context) error {
    u := new(model.User)
    if err := c.Bind(u); err != nil {
        return err
    }
    u.ID = idCounter
    idCounter++
    users = append(users, *u)
    return c.JSON(http.StatusCreated, u)
}
```

```
func GetUsers(c echo.Context) error {
    return c.JSON(http.StatusOK, users)
}

func GetUser(c echo.Context) error {
    id, _ := strconv.Atoi(c.Param("id"))
    for _, u := range users {
        if u.ID == id {
            return c.JSON(http.StatusOK, u)
        }
    }
    return c.NoContent(http.StatusNotFound)
}

func UpdateUser(c echo.Context) error {
    id, _ := strconv.Atoi(c.Param("id"))
    updated := new(model.User)
    if err := c.Bind(updated); err != nil {
        return err
    }
    for i, u := range users {
        if u.ID == id {
            users[i].Name = updated.Name
            return c.JSON(http.StatusOK, users[i])
        }
    }
    return c.NoContent(http.StatusNotFound)
}

func DeleteUser(c echo.Context) error {
    id, _ := strconv.Atoi(c.Param("id"))
    for i, u := range users {
        if u.ID == id {
            users = append(users[:i], users[i+1:]...)
            return c.NoContent(http.StatusNoContent)
        }
    }
    return c.NoContent(http.StatusNotFound)
}
```

c. Main Application (`main.go`)

```
package main

import (
    "echo-crud-demo/handler"
    "github.com/labstack/echo/v4"
)

func main() {
    e := echo.New()
    e.POST("/users", handler.CreateUser)
    e.GET("/users", handler.GetUsers)
    e.GET("/users/:id", handler.GetUser)
    e.PUT("/users/:id", handler.UpdateUser)
```

```
    e.DELETE("/users/:id", handler.DeleteUser)
    e.Logger.Fatal(e.Start(":8080"))
}
```

5. Testing with Postman

1. **Run the server:**

   ```
   go run main.go
   ```

2. **Open Postman** and create the following requests:

Operation	Method	Endpoint	Body (JSON)
Create User	POST	http://localhost:8080/users	`{ "name": "Alice" }`
Get All	GET	http://localhost:8080/users	
Get by ID	GET	http://localhost:8080/users/1	
Update User	PUT	http://localhost:8080/users/1	`{ "name": "Bob" }`
Delete User	DELETE	http://localhost:8080/users/1	

- Use Postman to send requests and observe responses.

6. Summary & Next Steps

- You have built a simple CRUD API using Echo in Go and tested it with Postman.
- Next steps:
  - Integrate a real database (e.g., PostgreSQL, MySQL).
  - Explore authentication and advanced API design.
  - Refactor and modularize for larger applications.

References

- Echo Framework documentation
- Example CRUD API with Echo
- Postman API testing

This documentation is designed for quick onboarding and hands-on practice with Go, Echo, and Postman.