Go: creare un'API REST per una To-Do List

Le API REST (Application Programming Interface forniscono un modo standardizzato per le applicazioni di comunicare tra loro attraverso il protocollo HTTP. In questo articolo, esploreremo come creare una semplice API REST in Go per gestire una lista di attività da fare (ToDo List). Utilizzeremo il linguaggio di programmazione Go per creare un server web che consentirà agli utenti di aggiungere, modificare, eliminare e ottenere attività dalla loro lista.

Prerequisiti

Per seguire questo tutorial, è necessario avere familiarità con i concetti di base di programmazione in Go e avere Go installato sul proprio sistema. Se non hai familiarità con il linguaggio Go, puoi trovare risorse utili sulla documentazione ufficiale di Go.

Assicurati di avere un ambiente di sviluppo Go funzionante prima di procedere.

Creare una struttura per le attività

Per iniziare, definiamo una struttura che rappresenta un'attività nella nostra ToDo List. Apri il tuo editor di codice e crea un file chiamato `main.go`. Inserisci il seguente codice:

```
package main

import (
```

```
        "encoding/json"
        "fmt"
        "net/http"
)

type Task struct {
        ID   int    `json:"id"`
        Text string `json:"text"`
}

var tasks []Task

func main() {
        http.HandleFunc("/tasks", tasksHandler)
        http.ListenAndServe(":8080", nil)
}
```

In questo passaggio, abbiamo creato una struct `Task` che rappresenta un'attività con un ID e un testo descrittivo. Abbiamo anche creato una variabile `tasks` per memorizzare le attività.

Implementare le operazioni CRUD

Ora, implementiamo le operazioni CRUD (Create, Read, Update, Delete) per gestire la lista delle attività. Aggiungi il seguente codice al file `main.go`:

```
func tasksHandler(w http.ResponseWriter, r
*http.Request) {
        if r.Method == http.MethodGet {
                getTasksHandler(w, r)
        } else if r.Method == http.MethodPost {
                addTaskHandler(w, r)
```

```
        } else if r.Method == http.MethodPut {
                updateTaskHandler(w, r)
        } else if r.Method == http.MethodDelete {
                deleteTaskHandler(w, r)
        } else {

w.WriteHeader(http.StatusMethodNotAllowed)
        }
}

func getTasksHandler(w http.ResponseWriter, r
*http.Request) {
        w.Header().Set("Content-Type",
"application/json")
        json.NewEncoder(w).Encode(tasks)
}

func addTaskHandler(w http.ResponseWriter, r
*http.Request) {
        var newTask Task
        err :=
json.NewDecoder(r.Body).Decode(&newTask)
        if err != nil {
                w.WriteHeader(http.StatusBadRequest)
                return
        }

        newTask.ID = len(tasks) + 1
        tasks = append(tasks, newTask)
        w.WriteHeader(http.StatusCreated)
}

func updateTaskHandler(w http.ResponseWriter, r
*http.Request) {
```

```
        var updatedTask Task
        err :=
json.NewDecoder(r.Body).Decode(&updatedTask)
        if err != nil {
                w.WriteHeader(http.StatusBadRequest)
                return
        }

        taskID := updatedTask.ID
        if taskID <= 0 || taskID > len(tasks) {
                w.WriteHeader(http.StatusNotFound)
                return
        }

        tasks[taskID-1].Text = updatedTask.Text
        w.WriteHeader(http.StatusOK)
}

func deleteTaskHandler(w http.ResponseWriter, r
*http.Request) {
        taskIDStr := r.URL.Query().Get("id")
        taskID, err := strconv.Atoi(taskIDStr)
        if err != nil || taskID <= 0 || taskID >
len(tasks) {
                w.WriteHeader(http.StatusNotFound)
                return
        }

        tasks = append(tasks[:taskID-1],
tasks[taskID:]...)
        w.WriteHeader(http.StatusOK)
}
```

In questo passaggio, abbiamo implementato le funzioni per ottenere la lista delle attività, aggiungere una nuova attività e gestire le richieste HTTP.

Nell'implementazione dell'operazione di aggiornamento (`updateTaskHandler`), stiamo decodificando i dati inviati nella richiesta per ottenere l'attività aggiornata. Controlliamo quindi che l'ID dell'attività sia valido e quindi aggiorniamo il testo dell'attività corrispondente nell'elenco delle attività.

Nell'implementazione dell'operazione di eliminazione (`deleteTaskHandler`), stiamo ottenendo l'ID dell'attività dalla query della richiesta e convertendolo in un numero intero. Successivamente, rimuoviamo l'attività corrispondente dall'elenco delle attività.

Assicurati di aver importato il pacchetto `"strconv"` all'inizio del tuo file `main.go` per convertire stringhe in numeri interi.

Esecuzione dell'applicazione

Ora che abbiamo implementato le operazioni CRUD base, possiamo eseguire la nostra applicazione. Apri il terminale nella directory del progetto e digita il seguente comando:

```
go run main.go
```

Il server dovrebbe avviarsi sulla porta 8080. Ora puoi interagire con l'API utilizzando strumenti come `curl` o un'interfaccia utente come Postman.

Conclusione

In questo articolo, abbiamo imparato come creare una semplice API REST in Go per gestire una ToDo List. Abbiamo definito una struttura per rappresentare le attività, implementato le operazioni CRUD base e creato un server web per gestire le richieste HTTP. Questo tutorial fornisce solo

una base di partenza, ma da qui puoi espandere e migliorare l'API aggiungendo funzionalità come l'autenticazione degli utenti, la persistenza dei dati e ulteriori operazioni di gestione delle attività.