TP : Création d'une API REST basique avec Gin

Vous devez développer une API REST pour gérer une liste de tâches. Chaque tâche est représentée par un identifiant unique, un titre et un statut (complet ou incomplet). L'API doit prendre en charge les opérations CRUD (création, lecture, mise à jour et suppression) pour les tâches.

1. Définir une structure `Task` qui représente une tâche avec les champs suivants :
   - `ID` : un identifiant unique de la tâche (un nombre entier)
   - `Title` : le titre de la tâche (une chaîne de caractères)
   - `Status` : le statut de la tâche (une chaîne de caractères, soit "complet" ou "incomplet")
2. Créer une variable `tasks` de type slice `[]Task` pour stocker les tâches.
3. Définir les routes de l'API REST avec Gin :
   - GET `/tasks` : récupère la liste des tâches
   - POST `/tasks` : crée une nouvelle tâche
   - GET `/tasks/:id` : récupère les détails d'une tâche spécifique par son identifiant
   - PUT `/tasks/:id` : met à jour une tâche spécifique par son identifiant
   - DELETE `/tasks/:id` : supprime une tâche spécifique par son identifiant
4. Implémenter les handlers (fonctions) pour chaque route :
   - GET `/tasks` : renvoie la liste des tâches en tant que réponse JSON
   - POST `/tasks` : crée une nouvelle tâche à partir des données JSON envoyées et l'ajoute à la liste des tâches
   - GET `/tasks/:id` : récupère les détails d'une tâche spécifique par son identifiant et renvoie les données JSON correspondantes
   - PUT `/tasks/:id` : met à jour les données d'une tâche spécifique par son identifiant à partir des données JSON envoyées
   - DELETE `/tasks/:id` : supprime une tâche spécifique par son identifiant de la liste des tâches
5. Installez le framework Gin en utilisant la commande `go get -u github.com/gin-gonic/gin`.
6. Utilisez la méthode `gin.Run()` pour exécuter l'API sur un port spécifique.

Solution

```
package main


import (
"github.com/gin-gonic/gin"
```

```
	"net/http"
	"strconv"
)

// Structure pour représenter une tâche
type Task struct {
	ID     int    `json:"id"`
	Title  string `json:"title"`
	Status string `json:"status"`
}

// Variable pour stocker les tâches
var tasks []Task

func main() {
	router := gin.Default()

	// Route pour récupérer la liste des tâches
	router.GET("/tasks", getTasks)

	// Route pour créer une nouvelle tâche
	router.POST("/tasks", createTask)

	// Route pour récupérer les détails d'une tâche spécifique
	router.GET("/tasks/:id", getTask)

	// Route pour mettre à jour une tâche spécifique
	router.PUT("/tasks/:id", updateTask)

	// Route pour supprimer une tâche spécifique
	router.DELETE("/tasks/:id", deleteTask)

	// Exécute l'API sur le port 8080
	router.Run(":8080")
}

// Handler pour récupérer la liste des tâches
func getTasks(c *gin.Context) {
	c.JSON(http.StatusOK, tasks)
}
```

```
// Handler pour créer une nouvelle tâche
func createTask(c *gin.Context) {
	var task Task

	// Bind les données JSON envoyées à la structure Task
	if err := c.ShouldBindJSON(&task); err != nil {
		c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
		return
	}

	// Génère un nouvel identifiant unique pour la tâche
	task.ID = len(tasks) + 1

	// Ajoute la tâche à la liste des tâches
	tasks = append(tasks, task)

	c.JSON(http.StatusCreated, task)
}

// Handler pour récupérer les détails d'une tâche spécifique
func getTask(c *gin.Context) {
	id, err := strconv.Atoi(c.Param("id"))
	if err != nil {
		c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})
		return
	}

	for _, task := range tasks {
		if task.ID == id {
			c.JSON(http.StatusOK, task)
			return
		}
	}

	c.JSON(http.StatusNotFound, gin.H{"error": "Task not found"})
}

// Handler pour mettre à jour une tâche spécifique
func updateTask(c *gin.Context) {
```

```
    id, err := strconv.Atoi(c.Param("id"))
    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})
        return
    }

    var updatedTask Task
    if err := c.ShouldBindJSON(&updatedTask); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": err.Error()})
        return
    }

    for index, task := range tasks {
        if task.ID == id {
            // Met à jour les champs de la tâche avec les nouvelles valeurs
            tasks[index].Title = updatedTask.Title
            tasks[index].Status = updatedTask.Status

            c.JSON(http.StatusOK, tasks[index])
            return
        }
    }

    c.JSON(http.StatusNotFound, gin.H{"error": "Task not found"})
}

// Handler pour supprimer une tâche spécifique
func deleteTask(c *gin.Context) {
    id, err := strconv.Atoi(c.Param("id"))
    if err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid task ID"})
        return
    }

    for index, task := range tasks {
        if task.ID == id {
            // Supprime la tâche de la liste des tâches
            tasks = append(tasks[:index], tasks[index+1:]...)

            c.JSON(http.StatusOK, gin.H{"message": "Task deleted"})
```

```
□□□return
□□}
□}

□c.JSON(http.StatusNotFound, gin.H{"error": "Task not found"})
}
```

Ce code utilise le framework Gin pour créer un routeur qui définit les routes de l'API REST. Chaque route est associée à un handler (fonction) qui est exécuté lorsque la route est appelée.

La structure `Task` est définie avec les champs ID, Title et Status correspondant aux données d'une tâche.

La variable `tasks` est uneslice (tranche) qui est utilisée pour stocker les tâches.

Les handlers implémentent les opérations CRUD pour les tâches :

- `getTasks` renvoie la liste des tâches en tant que réponse JSON.
- `createTask` crée une nouvelle tâche à partir des données JSON envoyées et l'ajoute à la liste des tâches.
- `getTask` récupère les détails d'une tâche spécifique par son identifiant et renvoie les données JSON correspondantes.
- `updateTask` met à jour les données d'une tâche spécifique par son identifiant à partir des données JSON envoyées.
- `deleteTask` supprime une tâche spécifique par son identifiant de la liste des tâches.

L'API est exécutée sur le port 8080 en utilisant `router.Run(":8080")`.

Assurez-vous d'installer le framework Gin en utilisant la commande `go get -u github.com/gin-gonic/gin` avant d'exécuter le code.

Revision #3
Created 2023-05-19 10:36:33 UTC by Noé Larrieu-Lacoste
Updated 2023-05-19 13:37:47 UTC by Noé Larrieu-Lacoste