Restful API Server

Simple Server

GitHub repo

Pour utiliser **Gin**, il suffit d'importer `github.com/gin-gonic/gin` au niveau de son fichier main et de créer une variable qui va contenir notre fameux routeur.

```
package main

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

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

Il faut ensuite lui définir des routes sur lesquels il va écouter. Dans notre exemple, nous ferons 2 GET qui renvoient un JSON facilement grâce à la librairie **Gin.**

```
package main

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

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

	router.GET("/", func(c *gin.Context) {
		c.JSON(200, gin.H{
			"message": "Hello World!",
		})
	})
	router.GET("/ping", func(c *gin.Context) {
		c.IndentedJSON(200, gin.H{
			"message": "pong",
```

```
□□})
□})
}
```

Enfin, il faut lui dire de se lancer et d'écouter sur un port spécifique grâce à une dernière ligne.

```
package main

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

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

□router.GET("/", func(c *gin.Context) {
□□c.JSON(200, gin.H{
□□□"message": "Hello World!",
□□})
□})
□router.GET("/ping", func(c *gin.Context) {
□□c.IndentedJSON(200, gin.H{
□□□"message": "pong",
□□})
□})

□router.Run(":9090")
}
```

Dans ce code, nous :

- Initialisons un routeur Gin en utilisant Default.
- Utilisons le GET, pour associer la méthode HTTP `GET` et le chemin `/, /ping` à une fonction contenant le contexte de la requête
- Utilisons `c.JSON` ou bien `c.IndentedJSON` permettent de simplement convertir une structure (en l'occurrence `gin.H` qui permet d'en créer une à la volée)

Il suffit alors de lancer notre magnifique app avec la commande `go run main.go` et aller tester nos routes.

Untitled

Untitled

Aussi simple que ça.

Simple music server

GitHub repo

Pour complexifier un peu plus les choses, on va refaire la même chose, mais avec quelques structures et un peu de découpage. L'objectif étant de servir une API de gestion d'une liste d'album de musique (Très originale oui).

On va essayer de faire du pseudo MVC et l'architecture de notre application sera la suivante :

```
controllers/
|  controller.go
|  command.go
|  query.go
data/
|  albums.go
models/
|  album.go
main.go
go.mod
go.sum
```

Models

Nous allons dans un premier temps définir la structure d'un album. Il faut déclarer dans le fichier `models/album.go` la structure suivante :

```
package models

type Album struct {
ID     string  `json:"id"`
Title  string  `json:"title"`
Artist string  `json:"artist"`
Price  float64 `json:"price"`
}
```

Les balises telles que `json:"artist"` spécifient le nom d'un champ lorsque le contenu de la structure est sérialisé en JSON.

Sans eux, le JSON utiliserait les noms de champs des propriétés, avec la majuscule, ce qui n'est pas très courant (pour rappel, en go, la majuscule, en début de variable ou fonction, permet de définir sa visibilité en dehors de son package) .

Data

On va ensuite déclarer une liste d’albums qui nous serviront de “base de données” pour notre application.

Remplissons dans le fichier `data/albums.go` de cette manière :

```
package data


import "gin-form/simple_music_api/models"


var Albums = []models.Album{
□{
□□ID:     "1",
□□Title:  "Taste of you",
□□Artist: "Rezz",
□□Price:  1.99,
□},
□{
□□ID:     "2",
□□Title:  "Go",
□□Artist: "Google",
□□Price:  9999,
□},
□{
□□ID:     "3",
□□Title:  "C#",
□□Artist: "Microsoft",
□□Price:  -1,
□},
}
```

Controllers

Occupons-nous de la partie controllers désormais !

Dans un premier temps, nous allons écrire nos fonctions servant à récupérer les données uniquement (en mode CQS tu connais).

Le fichier `controllers/query.go` va contenir deux fonctions :

- Une permettant de récupérer la liste des albums
- Une autre permettant de récupérer un seul album par son **ID**

La première partie du fichier ressemblera simplement à ça

```
package controllers

import (
	"gin-form/simple_music_api/data"
	"github.com/gin-gonic/gin"
	"net/http"
)

func getAlbums(c *gin.Context) {
	c.IndentedJSON(http.StatusOK, data.Albums)
}
```

Dans ce code, nous :

- Écrivons une fonction `getAlbums` qui prend un `gin.Context` en paramètre.
  - `gin.Context` est la partie la plus importante de Gin. Il prend en charge la requête, les détailles, la validation et sérialisation JSON, et plus encore.
- Appelons la fonction `c.IdentedJSON` afin de sérialiser notre tableau `data.Albums` en JSON indenté proprement.
- Utilisons une librairie interne à go `net/http` pour récupérer le code HTTP voulu (200). On pourrait écrire directement 200 à la main, mais maintenant vous savez que cette librairie existe

---

La deuxième méthode est un peu plus complexe et permet de récupérer un album parmi ceux existants avec son ID, qui sera passé dans le chemin de la requête (`/album/:id`).

```
func getAlbumByID(c *gin.Context) {
	id := c.Param("id")

	for _, album := range data.Albums {
		if album.ID == id {
```

```
		c.IndentedJSON(http.StatusOK, album)
		return
	}
}
c.IndentedJSON(http.StatusNotFound, gin.H{"error": "Album not found"})
}
```

Nous allons maintenant nous occuper du fichier `controllers/command.go` qui contiendra notre fonction permettant d'ajouter un album à notre liste.

```
package controllers

import (
	"gin-form/simple_music_api/data"
	"gin-form/simple_music_api/models"
	"github.com/gin-gonic/gin"
	"net/http"
)

func addAlbum(c *gin.Context) {
	var newAlbum models.Album

	if err := c.BindJSON(&newAlbum); err != nil {
		c.IndentedJSON(400, gin.H{
			"message": "Invalid JSON",
			"error":   err.Error(),
		})
		return
	}

	data.Albums = append(data.Albums, newAlbum)
	c.IndentedJSON(http.StatusCreated, newAlbum)
}
```

Dans cette fonction nous :

- Déclarons une variable `newAlbum` de type `Album`
- Utilisons la méthode fournit par **Gin** `c.BindJSON` qui va tenter de parser le body de notre requête dans notre structure en se basant sur le format décrit plus haut (les fameux `json:"artist"`).

    - Si, on n’y parvient pas, on renvoie un code erreur avec un message et stoppons l’exécution de la méthode.
- Ajoutons notre nouvel album à notre tableau
- Renvoyons un code de Création et l’album qui vient d’être enregistré

Les fonctions écrites plus hautes sont privées, il va falloir donc faire quelque chose pour qu’elles puissent être utilisées par notre routeur se trouvant dans le fichier `main.go`.

Ça sera le but de notre fichier `controllers/controller.go` qui va s’occuper de faire notre routage :

```
package controllers

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

func SourceControllers(router*gin.Engine) {
router.GET("/albums", getAlbums)
router.GET("/albums/:id", getAlbumByID)
router.POST("/albums", addAlbum)
}
```

Nous pouvons enfin relier tout ça à notre routeur principal dans le fichier `main.go`

```
package main

import (

"gin-form/simple_music_api/controllers"
"github.com/gin-gonic/gin"
)

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

controllers.SourceControllers(router)

router.Run(":8080")
}
```

On peut maintenant aller tester tout ça

Untitled

Untitled

Untitled

Untitled

Untitled

GGWP □□

Revision #1
Created 2022-05-09 19:27:54 UTC by Noé Larrieu-Lacoste
Updated 2022-05-09 19:30:46 UTC by Noé Larrieu-Lacoste