

Gestion d'hôtel

Cheikh Ahmadou Bamba KAMARA

16/03/2023

Mise en place d'une API REST de gestion d'hôtel permettant de réaliser les fonctionnalités conçues et de les représenter de façon sécurisée grâce au langage go.

1. Mise en place du fichier de connexion à la base de données:

A screenshot of a code editor window titled 'config.go'. The editor shows Go code for a database connection. The code is as follows:

```
1 package config
2
3 import (
4     "database/sql"
5 )
6
7 func Connect() *sql.DB {
8     db, err := sql.Open("mysql", "apache:@tcp(127.0.0.1:3308)/hotel")
9     if err != nil {
10         panic(err.Error())
11     }
12     return db
13 }
14
```

Dans ce fichier, nous avons la fonction `Connect()` qui permet de se connecter à la base de données et nous allons faire appel à cette fonction à chaque fois que nous aurons besoin de nous connecter sur la base de données.

2. Mise en place du fichier principal qui va gérer les routes et les requêtes:

```

1  package main
2
3  import (
4      "fmt"
5      "log"
6      "net/http"
7
8      _ "github.com/go-sql-driver/mysql"
9      _ "github.com/gorilla/mux"
10
11     controller "first.go/Controller"
12 )
13
14 func main() {
15     router := mux.NewRouter()
16     router.HandleFunc("/reservation/", controller.AllReservation).Methods("GET")
17     router.HandleFunc("/reservation/{id}", controller.GetReservationById).Methods("GET")
18     router.HandleFunc("/reservation/insert/", controller.InsertReservation).Methods("POST")
19     router.HandleFunc("/reservation/delete/{id}", controller.DeleteReservation).Methods("DELETE")
20     router.HandleFunc("/reservation/update/{id}", controller.UpdateReservation).Methods("PUT")
21
22     router.HandleFunc("/chambre/", controller.AllChambre).Methods("GET")
23     router.HandleFunc("/chambre/{id}", controller.GetChambreById).Methods("GET")
24     router.HandleFunc("/chambre/insert/", controller.InsertChambre).Methods("POST")
25     router.HandleFunc("/chambre/delete/{id}", controller.DeleteChambre).Methods("DELETE")
26     router.HandleFunc("/chambre/update/{id}", controller.UpdateChambre).Methods("PUT")
27
28     router.HandleFunc("/niveau/", controller.AllNiveau).Methods("GET")
29     router.HandleFunc("/niveau/{id}", controller.GetNiveauById).Methods("GET")
30     router.HandleFunc("/niveau/insert/", controller.InsertNiveau).Methods("POST")
31     router.HandleFunc("/niveau/delete/{id}", controller.DeleteNiveau).Methods("DELETE")
32     router.HandleFunc("/niveau/update/{id}", controller.UpdateNiveau).Methods("PUT")
33
34     router.HandleFunc("/facture/", controller.AllFacture).Methods("GET")
35     router.HandleFunc("/facture/{id}", controller.GetFactureById).Methods("GET")
36     router.HandleFunc("/facture/insert/", controller.InsertFacture).Methods("POST")
37     router.HandleFunc("/facture/delete/{id}", controller.DeleteFacture).Methods("DELETE")
38     router.HandleFunc("/facture/update/{id}", controller.UpdateFacture).Methods("PUT")
39
40     router.HandleFunc("/annexe/", controller.AllAnnexe).Methods("GET")
41     router.HandleFunc("/annexe/{id}", controller.GetAnnexeById).Methods("GET")
42     router.HandleFunc("/annexe/insert/", controller.InsertAnnexe).Methods("POST")
43     router.HandleFunc("/annexe/delete/{id}", controller.DeleteAnnexe).Methods("DELETE")
44     router.HandleFunc("/annexe/update/{id}", controller.UpdateAnnexe).Methods("PUT")
45 }

```

3. Mise en place du modèle dans lequel nous aurons les structures qui correspondent aux tables mysql:

```

1 package model
2
3 type Reservation struct {
4     Id_reservation int    `json:"id_reservation"`
5     Prenom         string `json:"prenom"`
6     Nom            string `json:"nom"`
7     Telephone      string `json:"telephone"`
8     Nuitee         int     `json:"nuitee"`
9     DateReservation string `json:"dateReservation"`
10    DateEntree      string `json:"dateEntree"`
11    DateSortie      string `json:"dateSortie"`
12    Chambre_numero  string `json:"chambre_numero"`
13    Facture_numero  string `json:"facture_numero"`
14 }
15
16 type Response_reservation struct {
17     Status int    `json:"status"`
18     Message string `json:"message"`
19     Data    []Reservation
20 }
21
22 type Chambre struct {
23     Numero    int    `json:"numero"`
24     Classe    string `json:"classe"`
25     Etat      string `json:"etat"`
26     TarifChambre float64 `json:"tarifChambre"`
27     Numero_niveau int    `json:"niveau_numeroNiveau"`
28 }
29
30 type Response_chambre struct {
31     Status int    `json:"status"`
32     Message string `json:"message"`
33     Data    []Chambre
34 }
35
36 type Niveau struct {
37     Numero    int    `json:"numeroNiveau"`
38     NombreChambre int    `json:"nombreChambre"`
39 }
40
41 type Response_niveau struct {
42     Status int    `json:"status"`
43     Message string `json:"message"`
44     Data    []Niveau

```

Pour chaque structure, nous avons les attributs et la réponse qui sera renvoyée en cas de requête effectuée sur la structure en question

4. Mise en place des contrôleurs:

Nous avons les fonctions “create, read, read b yid, insert, update et delete” dans chaque controller. Évidemment, nous avons séparé chaque classe avec ses propres fonctions. Les captures ci-dessous montrent, l’implémentation des fonctions sur la structure chambre

```
config.go  controller_chambre.go X
Controller > controller_chambre.go > ...
1 package controller
2
3 import (
4     "database/sql"
5     "encoding/json"
6     "fmt"
7     "log"
8     "net/http"
9     "strconv"
10
11     config "first.go/Config"
12     model "first.go/Model"
13     "github.com/gorilla/mux"
14 )
15
16 //-----GET_ALL-----
17
18 func AllChambre(w http.ResponseWriter, r *http.Request) {
19     var chambre model.Chambre
20     var response model.Response_chambre
21     var arrChambre []model.Chambre
22
23     db := config.Connect()
24     defer db.Close()
25
26     rows, err := db.Query("SELECT * FROM chambre")
27
28     if err != nil {
29         log.Println(err)
30     }
31
32     for rows.Next() {
33         err = rows.Scan(&chambre.Numero, &chambre.Classe, &chambre.Etat, &chambre.TarifChambre, &chambre.Numero_niveau)
34         if err != nil {
35             log.Fatal(err.Error())
36         } else {
37             arrChambre = append(arrChambre, chambre)
38         }
39     }
40
41     response.Status = 200
42     response.Message = "Success"
43     response.Data = arrChambre
44 }
```

```
// ----- INSERT -----
func InsertChambre(w http.ResponseWriter, r *http.Request) {
    var response model.Response_chambre

    db := config.Connect()
    defer db.Close()

    err := r.ParseMultipartForm(4096)
    if err != nil {
        panic(err)
    }
    classe := r.FormValue("classe")
    etat := r.FormValue("etat")
    tarifChambre := r.FormValue("tarifChambre")
    niveau_numeroNiveau := r.FormValue("niveau_numeroNiveau")

    _, err = db.Exec("INSERT INTO chambre(classe, etat, tarifChambre, niveau_numeroNiveau) VALUES(?, ?, ?, ?)", classe, etat, tarifChambre, niveau_numeroNiveau)

    if err != nil {
        log.Print(err)
        return
    }
    response.Status = 200
    response.Message = "Insert data successfully"
    fmt.Print("Insert data to database")

    w.Header().Set("Content-Type", "application/json")
    w.Header().Set("Access-Control-Allow-Origin", "")
    json.NewEncoder(w).Encode(response)
}
```

```

func GetChambreById(w http.ResponseWriter, r *http.Request) {
    // Récupération de l'identifiant de la chambre dans les paramètres de l'URL
    params := mux.Vars(r)
    id, err := strconv.Atoi(params["id"])
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    // Connexion à la base de données
    db, err := sql.Open("mysql", "apache:tcp(127.0.0.1:3308)/hotel")
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    defer db.Close()

    // Exécution de la requête de sélection
    row := db.QueryRow("SELECT * FROM chambre WHERE numero = ?", id)

    // Récupération des données de la chambre
    var chambre model.Chambre
    err = row.Scan(&chambre.Numero, &chambre.Classe, &chambre.Etat, &chambre.TarifChambre, &chambre.Numero_niveau)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    // Création de la réponse
    response, err := json.Marshal(chambre)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    // Envoi de la réponse
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    w.Write(response)
}

```

```

func DeleteChambre(w http.ResponseWriter, r *http.Request) {
    // Récupération de l'identifiant de la chambre depuis les paramètres de l'URL
    idStr := mux.Vars(r)["id"]
    id, err := strconv.Atoi(idStr)
    if err != nil {
        http.Error(w, err.Error(), http.StatusBadRequest)
        return
    }

    // Connexion à la base de données
    db, err := sql.Open("mysql", "apache@tcp(127.0.0.1:3308)/hotel")
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    defer db.Close()

    // Exécution de la requête de suppression
    result, err := db.Exec("DELETE FROM chambre WHERE numero = ?", id)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }

    // Vérification que la chambre a bien été supprimée
    rowsAffected, err := result.RowsAffected()
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
        return
    }
    if rowsAffected == 0 {
        http.Error(w, "Chambre not found", http.StatusNotFound)
        return
    }

    // Envoi de la réponse
    response := model.Response_reservation{
        Status: http.StatusOK,
        Message: "Chambre deleted successfully",
        Data:    nil,
    }
    jsonResponse, err := json.Marshal(response)
    if err != nil {
        http.Error(w, err.Error(), http.StatusInternalServerError)
    }
}

```



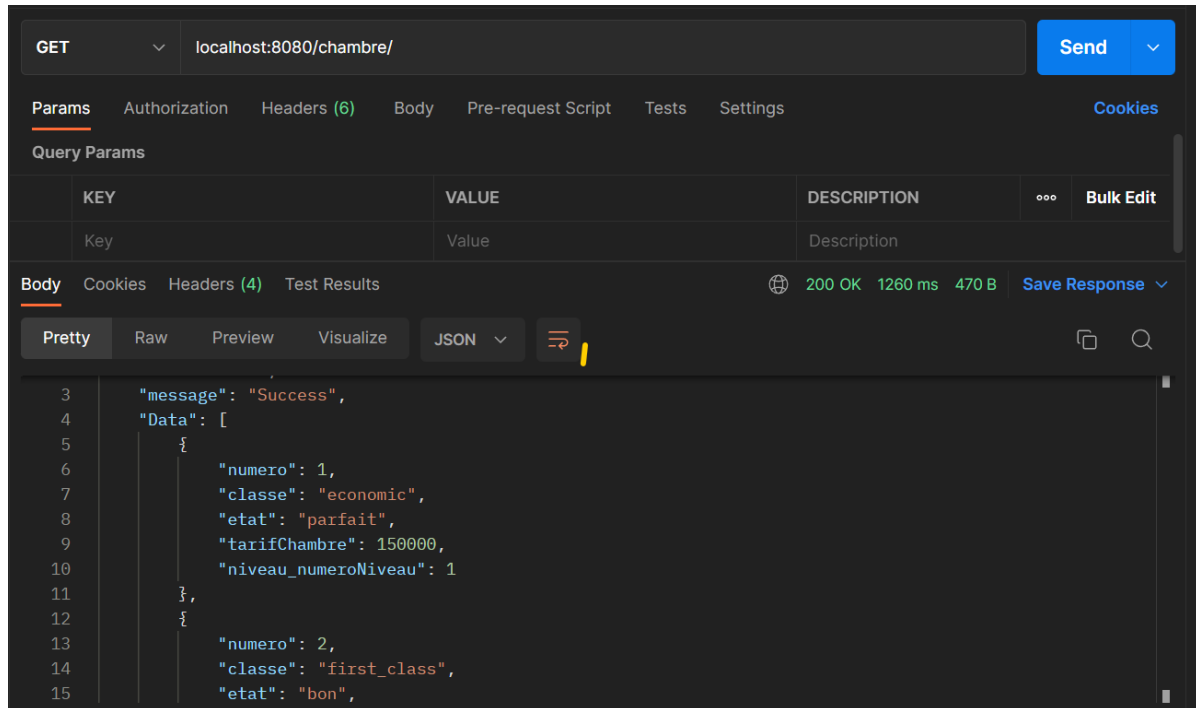
```

config.go  controller_chambre.go  UpdateChambre
Controller > controller_chambre.go > UpdateChambre
177 func UpdateChambre(w http.ResponseWriter, r *http.Request) {
178     // Récupération de l'ID de la chambre à modifier
179     vars := mux.Vars(r)
180     idChambre, err := strconv.Atoi(vars["id"])
181     if err != nil {
182         http.Error(w, err.Error(), http.StatusBadRequest)
183         return
184     }
185
186     // Lecture des données de la chambre à partir du corps de la requête
187     var chambre model.Chambre
188     err = json.NewDecoder(r.Body).Decode(&chambre)
189     if err != nil {
190         http.Error(w, err.Error(), http.StatusBadRequest)
191         return
192     }
193
194     // Connexion à la base de données
195     db, err := sql.Open("mysql", "apache:@tcp(127.0.0.1:3308)/hotel")
196     if err != nil {
197         http.Error(w, err.Error(), http.StatusInternalServerError)
198         return
199     }
200     defer db.Close()
201
202     // Exécution de la requête de mise à jour
203     result, err := db.Exec("UPDATE chambre SET classe = ?, etat = ?, tarifChambre = ?, niveau_numeroNiveau = ? WHERE numero = ?", &chambre.Classe, &chambre.Etat,
&chambre.TarifChambre, &chambre.Niveau_numeroNiveau, idChambre)
204     if err != nil {
205         http.Error(w, err.Error(), http.StatusInternalServerError)
206         return
207     }
208
209     // Vérification que la chambre a bien été modifiée
210     rowsAffected, err := result.RowsAffected()
211     if err != nil {
212         http.Error(w, err.Error(), http.StatusInternalServerError)
213         return
214     }
215     if rowsAffected == 0 {
216         http.Error(w, "La chambre n'a pas été trouvée", http.StatusNotFound)
217         return
218     }
219

```

5. Tests sur Postman:

La fonction qui affiche toutes les chambres:



The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:8080/chambre/
- Status:** 200 OK, 1260 ms, 470 B
- Response Body (JSON):**

```
{  "message": "Success",  "Data": [    {      "numero": 1,      "classe": "economic",      "etat": "parfait",      "tarifChambre": 150000,      "niveau_numeroNiveau": 1    },    {      "numero": 2,      "classe": "first_class",      "etat": "bon",
```

KEY	VALUE	DESCRIPTION
Key	Value	Description

La fonction qui permet de rechercher une chambre selon son id

localhost:8080/chambre/3

GET localhost:8080/chambre/3

Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (3) Test Results 200 OK 14 ms 203 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "numero": 3,
3   "classe": "economic",
4   "etat": "parfait",
5   "tarifChambre": 150000,
6   "niveau_numeroNiveau": 1
7 }
```

La fonction d'insertion:

POST localhost:8080/chambre/insert/

Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

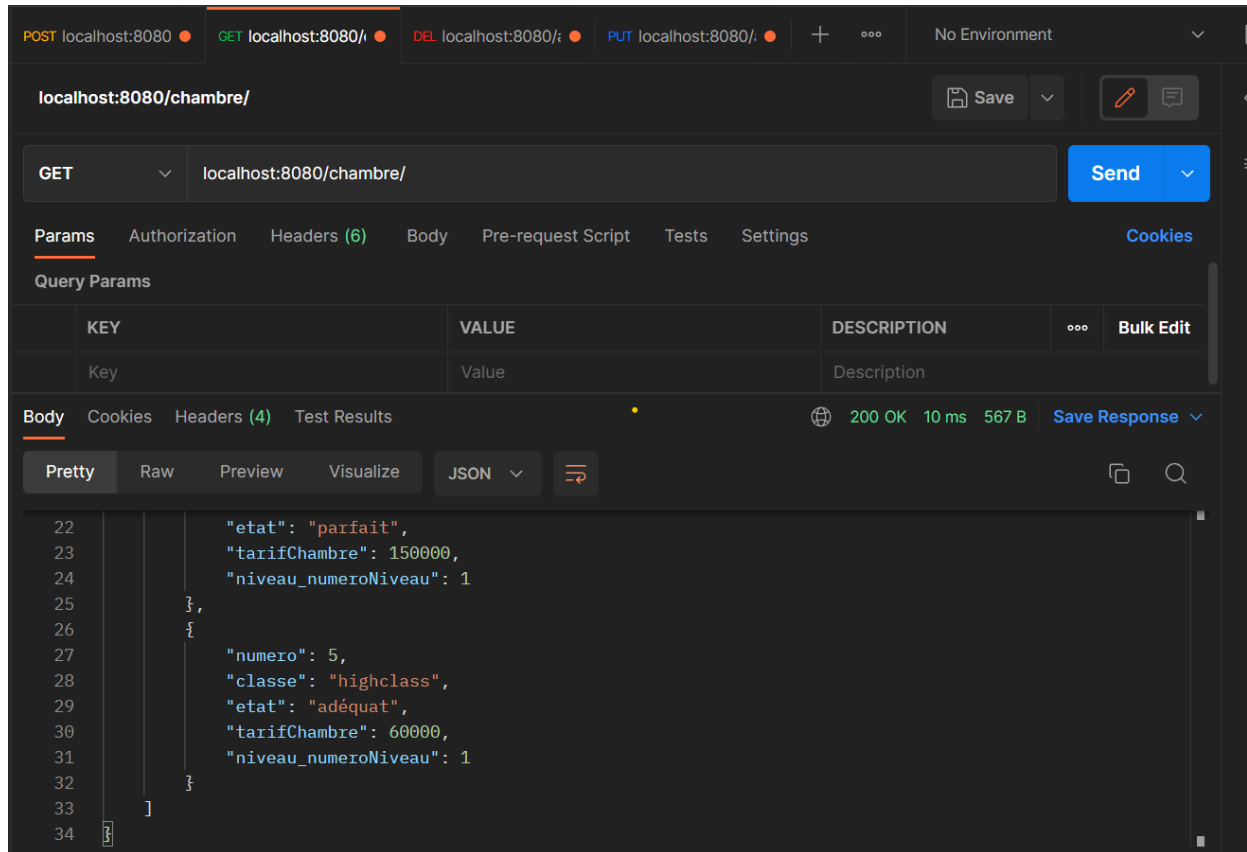
KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> classe	highclass	
<input checked="" type="checkbox"/> etat	adéquat	
<input checked="" type="checkbox"/> tarifChambre	60000	
<input checked="" type="checkbox"/> niveau_numeroNiveau	1	
Key	Value	Description

Body Cookies Headers (4) Test Results 200 OK 31 ms 204 B Save Response

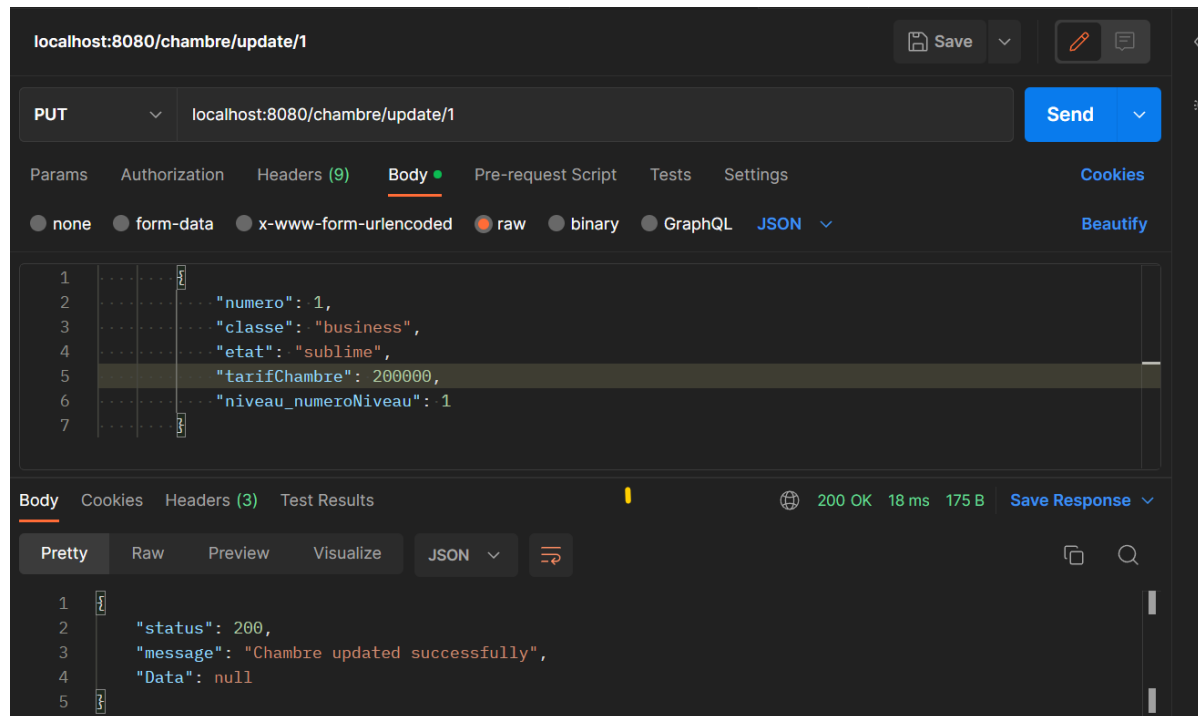
Pretty Raw Preview Visualize JSON

```
1 {
2   "status": 200,
3   "message": "Insert data successfully",
4   "Data": null
5 }
```

On voit que l'insertion a été effectuée avec succès et on le voit lorsqu'on fait appel à la fonction qui affiche toutes les chambres



La fonction qui permet de faire une mise à jour:



Les mises à jour sont automatiquement pris en compte

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** localhost:8080/chambre/
- Status:** 200 OK
- Time:** 11 ms
- Size:** 470 B

The **Body** tab is selected, displaying the response in JSON format:

```
{
  "status": 200,
  "message": "Success",
  "Data": [
    {
      "numero": 1,
      "classe": "business",
      "etat": "sublime",
      "tarifChambre": 200000,
      "niveau_numeroNiveau": 1
    },
    {
      "numero": 2,
```

Et enfin, la fonction qui supprime un enregistrement:

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** localhost:8080/chambre/delete/5
- Buttons:** Send, Cookies
- Tabs:** Params, Authorization, Headers (6), Body, Pre-request Script, Tests, Settings
- Query Params:** A table with columns KEY, VALUE, and DESCRIPTION. It contains one row with 'Key' and 'Value'.
- Body:** A table with columns KEY, VALUE, and DESCRIPTION. It contains one row with 'Key' and 'Value'.
- Response:** 200 OK, 17 ms, 175 B. Status: 200 OK, 17 ms, 175 B. Save Response
- Body Tab:** Pretty, Raw, Preview, Visualize. JSON format selected.
- Response Body (JSON):**

```
{  "status": 200,  "message": "Chambre deleted successfully",  "Data": null}
```

Voici, un récapitulatif de ce qui a été fait, naturellement le travail a été élargie sur toutes les tables de la base de données mais pour ne pas rendre un rapport trop encombrant nous avons choisi de vous montrer une seule classe et de vous montrer les tests