Validate Day05

Created @October 27, 2021 3:45 PM

Introduction

평가 진행 시 다음과 같은 사항들을 지켜주십시오.

- 상호 간 존중하는 마음으로 예의를 지키며 평가를 진행합니다.
- 평가 진행 중 발생한 기능 장애들은 평가자와 함께 문제점을 밝혀내도록 하십시오. 이 과정에서 토론과 논의가 오갈 수 있습니다.
- 동료가 이해한 것과 다르게 이해할 수도 있다는 점을 배려하십시오. 열린 마음으로 평가에 임하는 것을 권장합니다.

Guidelines

반드시 제출 시 이용되었던 Git 레포지토리를 기준으로 평가를 진행해야 합니다.

평가 진행 시 악의가 있는 별칭 등으로 결과를 조작하지는 않았는지 꼼꼼히 확인하십시오. 그리고 평가 진행 시 이용될 수 있는 스크립트에 대해서도 크로스 체크를 통해 별 문제가 없다는 것을 확인하여 당황스러운 상황을 피할 수 있도록 하십시오.

빈 레포지토리, (서브젝트에서 의도되지 않은) 기능 상에 문제가 있는 프로그램, 설명할 수 없는 코드가 포함된 치팅 등의 상황에서는 **FAIL**을 부여합니다. 특히 치팅에 대해선 경위와 사유를 듣고, 다음에도 같은 상황이 벌어지지 않도록 장려해주십시오.

General

- 제출된 레포지토리가 **gofmt**와 **golangci-lint**를 만족하는지 하십시오.

```
gofmt -l [filename]
golanci-lint run [filename]
```

- 제출된 레포지토리에 go.sum이 없거나 변조되어 모듈이 정상 작동하지 않을 수 있습니다. **go mod tidy**를 이용하여 모듈 정리를 진행 했음에도 문제가 있다면, **FAIL**입니다.
- 모든 기능들은 서브젝트에 명시된 *_test.go 파일을 이용하여 제출 이전에 검증되어 있어야 합니다. 테스트 파일이 잘 작성되어 있는지 확인하십시오. 테스트 케이스가 빈약하다면 평가자가 테스트 케이스를 요구할 수 있으며, 제시된 테스트 케이스가 정상적으로 작동하지 않는다면 **FAIL**입니다.

✅ **YES**

✅ **No**

Validation

1) ex00

- 전역 변수는 단 하나만 사용되어야 합니다.
- 전역 변수의 초기화가 main 함수 내에 있어야 합니다.
- 아래와 같은 원형으로 함수들이 작성되어 있는지 확인하십시오.

```
func Create(dictionary map[string]string, word, defnition string)
```

```
func Read(dictionary map[string]string, word string) string
```

```
func Update(dictionary map[string]string, word, definition string)
```

```
func Delete(dictionary map[string]string, word string)
```

- main 함수와 테스트 파일을 확인하십시오.

✅ **YES** ✅ **No**

2) ex01

- 사전에 해당되는 타입이 Dictionary로 정의되어 있어야 합니다.
- CRUD에 해당되는 Error들이 열거형 상수로 정의되어 있어야 합니다. 이 때 타입은 DictError이어야 하며, 각 Error는 Error 함수 호출이 가능해야 합니다.
- 각 Error의 문구는 아래와 같아야 합니다.

```
Create -> "Create Error: Word already exists in dictionary"
Read ->   "Read Error: Word is not in dictionary"
Update -> "Update Error: Word does not exist in dictionary"
Delete -> "Delete Error: Word cannot be removed from dictionary"
```

- CUD는 Read 함수를 기반으로 동작해야 합니다.
- 기존에 구현된 4개의 함수가 Dictionary의 메서드로 바뀌어야 하며, Error를 고려한 반환 타입을 올바르게 갖도록 되어 있어야 합니다.
- main 함수와 테스트 파일을 확인하십시오.

✅ **YES** ✅ **No**

3) ex02

- HTTP 통신에 github.com/gorilla/mux를 이용했는지 확인하십시오. 이를 이용하여 http.Handler를 만드는 함수가 Handler라는 이름으로 정의되어 있는지도 확인하십시오.

```
func Handler() http.Handler
```

- Handler 함수 내에는 "/dictionary"라는 경로에 대해서 "GET", "POST", "PUT", "DELETE"가 잘 작성되어 있으며, 각 Method에 해당하는 함수의 이름이 GetDict,

PostDict, PutDict, DeleteDict인지 확인하십시오. (이 때 기존에 이용하고 있던 사전의 전역 변수 위치가 바뀔 수 있습니다.)

- HTTP의 Request와 Response를 주고 받는 데이터는 Content-Type이 application/json이어야 하며, encoding/json을 이용하고 있어야 합니다. (json을 다루는 다른 패키지도 가능합니다.)
- "GET"에 대한 요청은 WORD라는 QueryString으로 넘어가서 처리되는지 확인하십시오.
- "POST", "PUT"에 대한 요청은 Request의 Body를 이용하여 단어와 정의가 넘어가서 처리되는지 확인하십시오.
- "DELETE" 역시 Request의 Body를 이용하여 단어를 넘겨서 처리되어야 합니다.
- 단어와 정의를 한 쌍으로 묶는 구조체를 추가로 선언할 수 있습니다.
- 각 Method의 Response로 넘어온 StatusCode가 아래와 같은지 확인하십시오.

```
- "GET"의 성공 → StatusOK / "GET"의 실패 → StatusNotFound
- "POST"의 성공 → StatusCreated / "POST"의 실패 → StatusBadRequest
- "PUT"의 성공 → StatusOK / "PUT"의 실패 → StautsBadRequest
- "DELETE"의 성공 → StatusOK / "DELETE"의 실패 → StatusBadRequest
```

- 각 Method의 StatusCode 검증 외에도, 원하는 데이터가 정상적으로 얻어졌는지 확인하십시오.

```
- "GET" -> 단어와 정의를 얻을 수 있어야 합니다.
- 그 외 -> 각 요청에 해당하는 작업의 성공 여부를 Success 구조체로 얻어냅니다.
```

- HTTP를 이용하는 서버의 구동이 4242 포트로 구동되는지 확인하십시오. 또한 구동된 서버는 문제가 있을 때 panic을 내야하고, 이에 대한 복구 루틴이 recover로 작성되어 있어야 합니다.
- main 함수와 테스트 파일을 확인하십시오.

✅ **YES** ✅ **No**

4) ex03

- Logger로 github.com/sirupsen/logrus가 이용되는지 확인하십시오.

- Logger를 어떤 식으로 운용하고 있는지, 어떤 설정을 했는지 확인하십시오.
- Logger에 해당되는 객체를 전역으로 운용할 수 있도록 선언되어 있는지 확인하십시오.
- Handler에 등록된 Router 별로 기록을 잘 작성하는지 확인하십시오. 이 때 기록 횟수는 Request를 받은 직후 1회, Request를 처리한 후 1회로 총 2회 이상이 되어 있어야 합니다.
- Logging에 대해선 별도의 테스트 과정이 없어도 무방합니다.

☑️ **YES** ☑️ **No**

Ratings

✔️ **OK** ⭐ **Outstanding project**

🗂️ **Empty work** 💾 **Invalid program** 👀 **Cheat** 💣 **Crash**