Задание по программированию: Кодогенератор http-фреймворка

Вы не отправили работу. Для успешной сдачи вам необходимо набрать 1/1 баллов.

ⓘ Похоже, это ваше первое задание по программированию. Подробнее

Срок сдачи Сдайте это задание до March 4, 11:59 PM PST

Инструкции

Моя работа

Обсуждения

Кодогенерация очень широко используется в го и надо уметь пользоваться этим инструментом.

hw5_codegen.zip

В этом задании вам необходимо будет написать кодогенератор, который ищет методы структуры, помеченный спец меткой и генерирует для них следующий код:

- http-обёртки для этих методов
- проверку авторизации
- проверки метода (GET/POST)
- валидацию параметров
- заполнение структуры с параметрами метода
- обработку неизвестных ошибок

Т.е. вы пишите программу (в файле `handlers_gen/codegen.go`) потом запускаете её, передавая в качестве параметров путь до файла для которого надо сгенерировать код, и путь до файла, в который записать результат. Запуск

будет выглядеть примерно так: `go build handlers_gen/* && ./codegen api.go api_handlers.go`. Т.е. запускаться он будет как `бинарник_кодогенератора что_парсим.го куда_парсим.го`

Хардкодить не надо. Все данные - имена полей, доступные значения, граничные значения - всё брать из самой струкруты, struct tags `apivalidator` и кода который мы парсим.

Если вы руками вписываете имя структуры, которое должно попасть в результирующий код после генерации - значит вы делаете не правильно, даже если у вас проходят тесты. Ваш кодогенератор должен работать универсально для любых полей и значений из тех что ему известны. Писать код надо так, чтобы он отработал на неизвестном вам коде, аналогичном api.go.

Единственное чем можно пользоваться - `type ApiError struct` при проверке ошибки. Считаем что это какая-то общеизвестная структура.

Кодогенератор уммет обрабатывать следующие типы полей структуры:

- int
- string

Нам доступны следующие метки валидатора-заполнятора `apivalidator`:

- required - поле не должно быть пустым (не должно иметь значение по-умолчанию)
- paramname - если указано - то брать из параметра с этим именем, иначе lowercase от имени
- enum - "одно из"
- default - если указано и приходит пустое значение (значение по-умолчанию) - устанавливать то что написано указано в default
- min - >= X для типа int, для строк len(str) >=
- max - <= X для типа int

Формат ошибок смотрите в тестах. Порядок следования ошибок:

- наличие метода (в ServeHTTP)
- метод (POST)
- авторизация
- параметры в порядке следования в структуре

Авторизация проверяется просто на то что в хедере пришло значение `100500`

Сгенерённый код будет иметь примерно такую цепочку

- `ServeHTTP` - принимает все методы из мультиплексора, если нашлось - вызывает `handler$methodName`, если нет - говорит 404

- `handler$methodName` - обёртка над методом структуры `$methodName` - осуществляет все проверки, выводит ошибки или результат в формате JSON
- `$methodName` - непосредственно мето структуры для которого мы генерируем код и который парсим. имеет префикс `apigen:api` за который следует json с иметем метода, типом и требованием авторизации. Его генерировать не нужно, он уже есть.

```
type SomeStructName struct{}

func (h *SomeStructName ) ServeHTTP(w http.ResponseWriter, r *http.Request)
{
  switch r.URL.Path {
  case "...":
    h.wrapperDoSomeJob(w, r)
  default:
    // 404
  }
}

func (h *SomeStructName ) wrapperDoSomeJob() {
  // заполнение структуры params
  // валидирование параметров
  res, err := h.DoSomeJob(ctx, params)
  // прочие обработки
}
```

По структуре кодогенератора - надо найти все методы, для каждого метода сгенерировать валидацию входящих параметров и прочие проверки в handler$methodName, для пачки методов структуры сгенерировать обвязку в ServeHTTP

Над ошибками кодогенератора можно сильно не заморачиваться - параметры который в него передаются будем считать гарантированно корректными.

Пример для кодогенерации смотрите в папке с заданием.

Что надо парсит в ast:

- node.Decls -> ast.FuncDecl - это метод. у него надо проверить что есть метка и начать генерировать для него обёртку
- node.Decls -> ast.GenDecl -> spec.(*ast.TypeSpec) + currType.Type. (*ast.StructType) - это структура. она нужна чтобы по ней генерить валидацию для метода, который мы нашли в проедыдущем пункте
- https://golang.org/pkg/go/ast/#FuncDecl - тут смотрите к какой структуре относится метод

Вы можете использовать как шаблоны чтобы сгенерировать сразу весь метод, так и собирать код из маленьких кусков.

Структура директории с заданием:

- example/ - пример с кодогенерацией из 3-й лекции 1-й части курса. Можно этот код взять за основу.

- handlers_gen/codegen.go - сюда вам писать код
- api.go - этот файл вам надо скармливать в кодогенератор. редактировать его не надо
- main.go - тут всё ясно. редактировать не надо
- main_test.go - этот файл надо запускать для тестирования после кодогенерации. редактировать не надо

Запуск тестов будет происходить так:

```
# находясь в этой папке

# расширение .exe только для счастливых обладателей windows

# собирает кодогенератор и сразу же запускает генерацию http-хендлеров для файла api.go, записывая результат в api_handlers.go

go build handlers_gen/* && ./codegen.exe api.go api_handlers.go

# запуск тестов

go test -v
```

Примечание

- В тесты закралась ошибка "login must **me** not empty". Но поскольку оно было обнаружено после старта - считает что это фича, чтобы не сломать код других людей и обрабатываем "empty"-поля с "me
- Если при загрузке задания выдаёт ошибку `Test failed - preparation step - codegen compilation failed` - убедитесь что у вас не используются внешние библиотеки, уже было несколько случаев что подставлялся какой-то другой пакет с шаблонизатором, не из стандартной библиотеки

How to submit

When you're ready to submit, you can upload files for each part of the assignment on the "My submission" tab.