Всероссийский конкурс исследовательских и проектных работ школьников «Высший пилотаж» Всероссийский конкурс-конференцияшкольников «Авангард»

Мульти-язычный веб-фреймворк на языке Go

Проект

Автор: Мигин Андрей,Учащийся 10 класса,
ЧОУ «Первая гимназия Максимум»
Г. Пенза

2024 г.

Оглавление

Введение.

Веб-разработка… Казалось бы, всё в этой сфере исследовано. Хотите работать со статическими страницами? HTML, CSS и JavaScript сделают всё, что вам надо, и даже больше. Не хватает? Есть React, Angular и прочее. Нужен сервер? Не вопрос, что хотите реализовать? Есть самые разные варианты: от ручного разбора HTTP-заголовков и простейшего CGI на языке C, заканчивая такими «монстрами», как Django или Java Servlet. Хотите и то, и другое сразу? Вперед! Next JS для вас! И всё-таки…

Ни один из вариантов не универсален. Проекты укрупняются, и каркас фреймворка начинает становится помехой. Приходится подключать сторонние языки, что чаще всего является головной болью для программиста, тестировщика и архитектора. Мало того, фреймворки дают в то же время и слишком мало. Например, очень часто не хватает нормальных WebSocket и OAuth из коробки. Хочется комбинировать разные подходы, не вдаваясь в детали реализации, например, «библиотека + фреймворк», «фреймворк + CRM», «CRM + FastCGI» … Мало того, многие CRM обладают костыльным API. Решением станет мой… наверное, всё-таки фреймворк.

Актуальность обусловлена укрупнением проектов до такой степени, что ни один каркас их не держит, а решения на рынке пока нет.

Глава 1 Теоретическая часть.

1.1 История Веб-разработки

За последние 30 лет веб-разработка изменилась кардинально. В 1989 Тим Бернерс-Ли предложил концепцию Всемирной паутины (WWW, World Wide Web), разработал протокол HTTP (Hyper Text Transfer Protocol, протокол передачи гипертекста) и формат языка HTML (Hyper Text Markup Language, язык гипертекстовой разметки). Поначалу HTML представлял собой текст с ссылками на другие страницы (*гиперссылки*, от этого *гипертекстовая* разметка). Вслед за этим выходит первый веб-браузер – Mosaic. Потом HTML научился понимать 16-цветные изображения, ещё чуть позже добавили поддержку таблиц, что привело к упорядочиванию контента. Но это Front-end (внешний вид). Гораздо печальнее всё было на серверах. Начинают приобретать популярность CGI-решения – небольшие сценарии, которые ошеломительно быстро запускаются и не менее быстро завершаются. Проблема состояла в отсутствии языка. Технически вариантов было три: C, Java и Perl. Но C был и остаётся слишком низкоуровневым решением, на которым сделать что-либо в Web очень сложно. Java прочно закрепился за коммерческим сегментом, но был непомерно «объемным» для энтузиастов. Поэтому и оставался только Perl, но тот был не предназначен для Web. Но в 1995 году произошла революция: вышел в свет язык программирования PHP. Этот язык, созданный специально для web, стал основным языком. 1995 год выдался ярким и для фанатов HTML: Netscape создаёт прорывной проект: язык JavaScript, максимально простой даже для непрограммиста. Он сделал страницы интерактивными, и стал главным языком программирования в Web по сей день.

В 1996 году появляется CSS и Macromedia Flash (будущий Adobe Flash). Это привело к неистовому развитию front-end разработки.CSS – это язык стилей, сделавший сайты более привлекательными. Flash же на долгое время стал основой для веб-анимации. Ещё через год появляется тот самый HTML 4, который по сей день изучают в школах. Компьютеры того времени не могли «переваривать» всё это.

В этом состоянии Web оставался ещё некоторое время – вплоть до выхода Django в 2005 году и появления Node в 2009. Это был прорывной момент в Web-разработке. С учётом популярности .NET большая часть разработчиков начала уходить с PHP.

Front-end разработка тоже не стояла на месте – появление jQuery и первого ECMAScript привели к стандартизации языка JavaScript. Теперь разработчику не надо было угадывать, в каком браузере работает скрипт и какие команды использовать. Вскоре выйдет HTML5 (2014 год), AngularJS (2010 год), React (2013 год) и так далее. Фреймворки будут расти и множиться как на дрожжах. Более того, объединяя серверный Node JS с клиентским JS, создавая проекты вроде Next JS.

1.2 Инструменты

На текущий момент мы целый «зоопарк» самых разных решений. Расскажу о тех, которые задействую впоследствии:

- PHP. Это, наверное, самый web-ориентированный язык программирования. Он до сих пор занимает свою нишу и ещё надолго останется одним из самых популярных языков в Web
- C++. Это новая версия языка C. Из интересного было добавлено ООП. Он не используется в Web, но используется очень часто в других отраслях.

- Node JS. Это серверная версия языка JavaScript (JS), упомянутого ранее. Очень популярен в микросервисах и стартапах.
- Python. Это универсальный язык программирования. Сложно сказать, где он *не* используется. Web, игры, ИИ, desktop-приложения, android… В Web используются в основном его фреймворки Django (для сложных сайтов) и Flask (для более легковесного)
- Go. О нём чуть подробнее

1.3. Язык программирования Go

Go – это компилируемый язык программирования, придуманы Робом Пайком (Rob Pike). Он очень быстрый и используется в CLI (консольных программах), Web и создании библиотек.

Особенности языка:

1. Очень высокая скорость работы
2. Параллельность на уровне ядра
3. Хорошо укомплектованная стандартная библиотека
4. Поддержка работы с C на уровне синтаксиса
5. Очень выразительное ООП (которое тут ООП и не является)

Исходя из этих пяти пунктов я решил делать ядро моего фреймворка на Go. Назвал я его EasyServer.

1.4 Фреймворк, библиотека, CRM…

Фактически, мой проект работает не так, как фреймворк. Он совмещает в себе элементы библиотеки (код работает с EasyServer API, а не наоборот), CRM (половина сервера настраивается через конфигурационные файлы, т. е. без знаний программирования можно собрать себе базовый сервер), «полуфабрикат» (код EasyServer можно использовать как каркас для своего проекта). Мало того, он поддерживает работу с FastCGI (более быстрая версия CGI, например, PHP-FPM для PHP). Это сделано намеренно. Очень часто в процессе работы над проектом приходится менять структуру или даже целый стек. Именно поэтому я выбрал такую систему: чтобы во время роста проекта не приходилось делать болезненные переходы.

1.5 Git

Git — это система контроля версий, использованная в проекте. Она позволяет отслеживать все изменения в проекте. Более того, с помощью Git можно легко разделять между собой задачи и потенциально совместно работать над проектом. Весь код с помощью Git можно хранить онлайн — например, с помощью таких сервисов, как GitHub (на нём лежит мой проект).

Глава 2. Практическая часть

2.1. Создание архитектуры

Если театр начинается с вешалки, то хороший инструмент на Go — с архитектуры. В моем случае она играет ключевую роль, т. к. проект должен быть прост для понимания не-программистами. Поэтому, первое, что я сделал — организовал архитектуру (Приложение 1). Рассмотрим ее подробнее:

- Папки .venv, .github и .vscode служебные — они нужны для Python, GitHub и редактора кода соответственно
- Папка build содержит исполняемые файлы и скомпилированные C-библиотеки (файлы .so в Linux и .dll в Windows), запускающие тестовый веб-сервер и не только. Обратите внимание, что папка серая — она не хранится в Git.
- Папка cmd хранит весь код программных компонентов, которые собираются в исполняемые файлы.
- Папка config содержит всю конфигурацию проекта. Она выполняется *из исходного кода.* При этом она не выглядит так страшно, как это происходит обычно в таких проектах. (Приложение 2). Как видите, она очень похожа на JSON-формат.
- В папке internal находится исходный код «ядра» EasyServer. Имя internal в Go означает, что использоваться код в этой папке может только в этом проекте
- Папка modules содержит модули EasyServer (о них чуть позже)
- Предполагаемая папка plugins должна содержать плагины EasyServer (части кода, встраиваемые в проект — на стадии реализации)
- Предполагаемая папка cgi должна содержать вызываемые FastCGI-скрипты
- Отсутствующая на фотографии папка ffi содержит информацию о связках с другими языками
- Папка templates хранит шаблоны для разных страниц и не только
- Файл .air.toml нужен для тестирования сервера в реальном времени
- Файл .env нужен для настройки переменных среды
- Файл .gitignore описывает, что нужно хранить в Git
- go.mod и go.sum содержат информацию о сторонних библиотеках — например, об использованной версии godotenv (библиотека для разбора файла .env)
- lib.go и есть та библиотека, которую может использовать разработчик (API, Application Programming Interface)
- LICENSE — это лицензия (GPL v3)
- Makefile — это скрипт для сборки проекта
- README.md — это ознакомительный файл, рассказывающий о проекте

2.2 Ядро

Ядро EasyServer — это его основная часть. Она описывает работу JSON-версии конфигурации, структуру маршрутизации, перехватчиков и базы данных (в том числе моделей — о них далее). Кроме того, организован ряд утилит, необходимых в проекте. Файлов очень много, их характер очень общий, поэтому здесь их приводить не буду.

2.3 Модули

Модули — это те самые элементы маршрутизации, подключаемые *только* в исходном коде (либо через API), и настраиваемые в конфигурации отдельного модуля. Наверное, основным стандартным модулем является auth. Приведу несколько примеров из него.

2.3.1 Перехватчик

Задача перехватчика — перед обработкой страницы выполнить некоторые действия. Например, здесь — распознать пользователя по его ключу.

```
package app

import (
"context"
"fmt"
"net/http"
"strings"

"github.com/golang-jwt/jwt/v5"

// Modules
"github.com/BIQ-Cat/easyserver/modules/auth/datakeys"
"github.com/BIQ-Cat/easyserver/modules/auth/models"

// Internals
"github.com/BIQ-Cat/easyserver/internal/routes"
"github.com/BIQ-Cat/easyserver/internal/utils"

// Configuration
config "github.com/BIQ-Cat/easyserver/config/base"
moduleconfig "github.com/BIQ-Cat/easyserver/config/modules/auth"
)

type UserKey struct{}

func JWTAuthentication(next http.Handler) http.Handler {
return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
controller := routes.GetController(r.URL.Path)
if _, ok := controller.Data[datakeys.RequireAuth]; controller == nil || !ok {
next.ServeHTTP(w, r)
return
}

var response map[string]interface{}
tokenHeader := r.Header.Get("Authorization")
if tokenHeader == "" {
response = utils.Message(false, "Missing auth token")
w.WriteHeader(http.StatusForbidden)
utils.Respond(w, response)
return
}

bearerToken := strings.Split(tokenHeader, " ") // []string{"Bearer", "<token>"} (?)
if len(bearerToken) != 2 && bearerToken[0] != "Bearer" {
```

```
response = utils.Message(false, "Invalid/Malformed auth token")
w.WriteHeader(http.StatusForbidden)
utils.Respond(w, response)
return
}

tokenPart := bearerToken[1]
tk := &models.Token{}
token, err := jwt.ParseWithClaims(tokenPart, tk, func(_ *jwt.Token) (interface{}, error) {
return []byte(config.EnvConfig.TokenPassword), nil
})

if err != nil {
response = utils.Message(false, "Malformed authentication token")
w.WriteHeader(http.StatusForbidden)
utils.Respond(w, response)
return
}

if !token.Valid {
response = utils.Message(false, "Token is not valid.")
w.WriteHeader(http.StatusForbidden)
w.Header().Add("Content-Type", "application/json")
utils.Respond(w, response)
return
}

if config.Config.Debug {
fmt.Printf("User %v\n", tk.UserID)
}

if verificationRequired(*tk, r.URL.Path, controller) {
response = utils.Message(false, "Verification required.")
w.WriteHeader(http.StatusForbidden)
w.Header().Add("Content-Type", "application/json")
utils.Respond(w, response)
return
}

ctx := context.WithValue(r.Context(), UserKey{}, tk.UserID)
r = r.WithContext(ctx)
next.ServeHTTP(w, r)
})
}

func verificationRequired(tk models.Token, path string, c *routes.Controller) bool {
```

```
	_, ok := c.Data[datakeys.RequireVerification]
	return !tk.Verified && (moduleconfig.Config.Verify.Require && path != "/auth/verify-send"
&& path != "/auth/verify-recieve" || ok)
	}
```

2.3.2 Маршрутизатор (контроллер)

Задача контроллера — определить, что происходит, если был произведен HTTP-запрос к странице. Стоит учесть, что модуль сразу определяет абсолютный путь до него. Контроллер отвечает за логику, но не более. Например, смена пароля по адресу /auth/change-password:

```
	package controllers

	import (
	"net/http"

	"github.com/jinzhu/gorm"

	// Modules

	moduleconfig "github.com/BIQ-Cat/easyserver/config/modules/auth"
	"github.com/BIQ-Cat/easyserver/modules/auth/app"
	"github.com/BIQ-Cat/easyserver/modules/auth/datakeys"
	"github.com/BIQ-Cat/easyserver/modules/auth/models"

	// Internals
	"github.com/BIQ-Cat/easyserver/internal/db"
	"github.com/BIQ-Cat/easyserver/internal/routes"
	"github.com/BIQ-Cat/easyserver/internal/utils"
	// Configuration
	)

	func init() {
	changePassword := func(w http.ResponseWriter, r *http.Request) {
	id := r.Context().Value(app.UserKey{}).(uint)
	var acc models.Account

	err := db.GetDB().Table("accounts").Where("id = ?", id).First(&acc).Error
	if err == gorm.ErrRecordNotFound {
	panic(err) // must exist
	} else if err != nil {
	utils.HandleError(w, err)
	return
	}

	if !moduleconfig.Config.Verify.SetPasswordAfter || acc.Password != "" {
	utils.Respond(w, utils.Message(false, "Forbidden"))
	return
```

```
}

err = r.ParseForm()
if err != nil {
utils.HandleError(w, err)
return
}

if r.Form.Has("password") {
utils.Respond(w, utils.Message(false, "Invalid request"))
return
}

resp, err := acc.ChangePassword([]byte(r.Form.Get("password")))
if err != nil {
utils.HandleError(w, err)
}
utils.Respond(w, resp)
}

Route["change-password"] = routes.Controller{
Handler: http.HandlerFunc(changePassword),
Methods: []string{http.MethodPost},
Data: map[string]any{
datakeys.RequireAuth: true,
},
}
}
```

2.3.3 Модель

Модель — это представление объекта в базе данных. Она представляет структурный тип плюс несколько методов к ней. С ней работает ядро. Пример структуры:

```
type Account struct {
db.ModelSkeleton
Email string `json:"email,omitempty"`
Password string `json:"password"`
Token string `json:"token" gorm:"-:all"`
Phone string `json:"phone,omitempty"`
Username string `json:"username"`
Verified bool `json:"verified"`
VerificationOTP string `json:"-"`
RestorePasswordOTP string `json:"-"`
TimeVerificationOTPSet time.Time `json:"-"`
TimeForgotPasswordOTPSet time.Time `json:"-"`
}
```

Пример метода (отправка почты):

```
func (a *Account) SendEmailOTP(email string, isVerification bool, host string)
(map[string]interface{}, error) {
type emailData struct {
URL string
FirstName string
Subject string
Token string
}

if a.Verified && isVerification {
return utils.Message(false, "Account is already verified"), nil
}

if !moduleconfig.Config.Create.Email.Require {
return utils.Message(false, "Email verification is disabled on this server"), nil
}

if a.Email != email {
return utils.Message(false, "Incorrect email"), nil
}

otp, msg, err := a.setOTP(isVerification)
if msg != nil || err != nil {
return msg, err
}

var controller, subject, template string

if isVerification {
controller = "verify-recieve"
subject = moduleconfig.Config.Verify.EmailSubject
template = "verify.html"
} else {
controller = "reset-password"
subject = moduleconfig.Config.RestorePassword.EmailSubject
template = "reset.html"
}

data := emailData{
URL: host + "/auth/" + controller + "?token=" +
base64.StdEncoding.EncodeToString([]byte(otp)) + "&visual=1",
FirstName: a.Username,
Subject: subject,
Token: otp,
```

```
        }

        err = utils.SendEmail(a.Email, subject, &data, template)
        if err != nil {
        return utils.Message(false, "Error while sending E-mail: "+err.Error()), nil
        }

        return utils.Message(true, "OTP is sent on Email"), nil
        }
```

(как вы видите, я прилагаю не весь файл, а фрагменты из разных файлов здесь)

2.4 Мультиязычность (FFI)

В папке ffi хранятся самые разные обёртки для разных языков (не Go). Вследствие объема кода, множества сопутствующих пояснений и низкоуровневых тонкостей код приводить не буду, расскажу лишь концепцию работы каждого языка. Вы можете посмотреть сами в проекте на GitHub (Приложение 3)

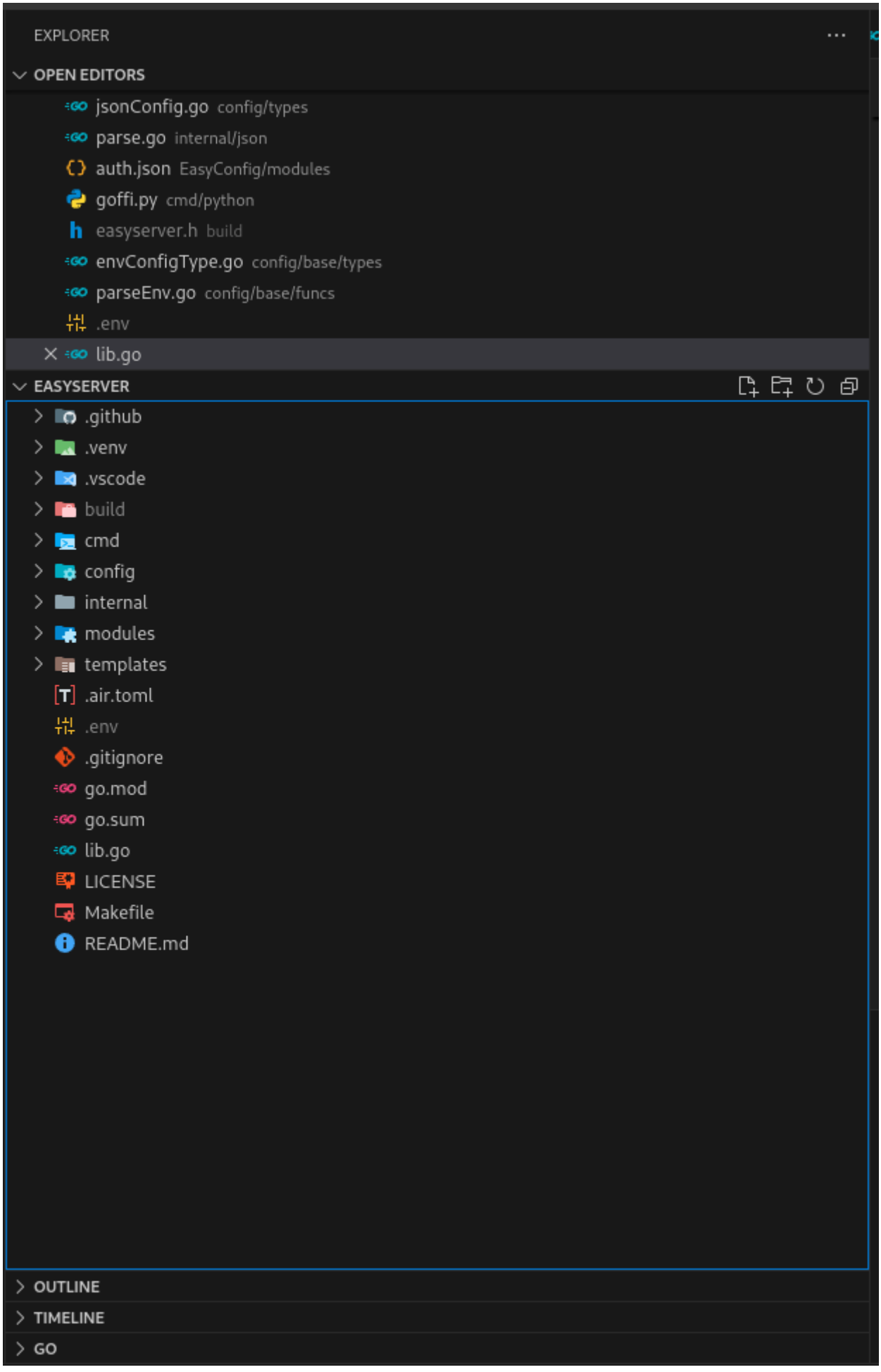
- C/C++: тут всё просто. С использованием инструмента cgo производится трансляция кода на Go в код на C с заменой типов данных Go на указатели (дескрипторы). В дальнейшем работа ведётся через них.
- PHP: предполагается наличие PHP-FPM. Запускается в формате FastCGI-скриптов. Все данные передаются через HTTP-заголовки X-EasyServer-*
- Python: используется Python C API для встраивания кода на Python и Python-скрипт для перевода библиотеки C в удобоваримый для Python формат.

Заключение

На данный момент мне предстоит ещё много работы. В частности, необходимо сделать плагины, завершить переход с GORM (сторонняя библиотека для SQL) на database/sql и подключить NodeJS. Но я уже могу с уверенностью сказать, что первый прототип получился рабочим и удачным. Единственная беда — Makefile пока работает лишь в UNIX-совместимых системах (Linux, BSD и MacOS). Для Windows нужен отдельный bat-файл. Он уже в процессе написания и потенциально будет готов в ближайшем будущем. Любой желающий может принять участие в разработке проекта, так как это проект с открытым исходным кодом и он доступен на GitHub всем желающим.

Приложения

Приложение 1. Архитектур


EXPLORER
OPEN EDITORS
jsonConfig.go config/types
parse.go internal/json
auth.json EasyConfig/modules
goffi.py cmd/python
easyserver.h build
envConfigType.go config/base/types
parseEnv.go config/base/funcs
.env
lib.go
EASYSERVER
.github
.venv
.vscode
build
cmd
config
internal
modules
templates
.air.toml
.env
.gitignore
go.mod
go.sum
lib.go
LICENSE
Makefile
README.md
OUTLINE
TIMELINE
GO

```go
package moduleconfig      AnDev, last month via  PR #8 · Add library structu

import (
    "net/http"

    "github.com/BIQ-Cat/easyserver/config/modules/cors/types"
)

var Config = types.Config{
    AllowedOrigins: []string{"*"},
    AllowedMethods: []string{
        http.MethodGet,
        http.MethodPost,
        http.MethodPut,
        http.MethodDelete,
        http.MethodPatch,
        http.MethodHead,
    },
    AllowedHeaders:   []string{"*"},
    AllowCredentails: false,
}
```

Приложение 2. Конфигурация

Приложение 3. Ссылка на GitHub

https://www.github.com/BIQ-Cat/EasyServer

Рецензия на проектную работу ученика
10 класса ЧОУ «Первая гимназия Максимум»
Мигина Андрея
на тему: «Мульти-язычный веб-фреймворк на языке Go»

Главным вопросам проекта является создание удобного и полезного фреймворка. Поэтому актуальность данного проекта очевидна.

Материал, который ученик использует в проекте, выходит далеко за рамки школьного, что является следствием интереса и самостоятельного изучения всевозможной литературы по данной теме.

Оценивая работу в целом, стоит отметить хорошую структуру проекта, работа соответствует плану и поставленным задачам, логически продумана. Оформление работы и культура письма обеспечивают понимание изложенного материала.

Оформление работы: работа оформлена в полном соответствии с предъявляемыми требованиями.

Работа, представленная для рецензирования, является самостоятельным проектом, выполнена на высоком теоретическом и практическом уровне, заслуживает положительной оценки.

13.12.2024 г. Фахретдинова Л,А.