Харківський університет радіоелектроніки Факультет
комп'ютерних наук
Кафедра програмної інженерії ЗВІТ
до практичного заняття з дисципліни "Аналіз
та рефакторинг коду"
на тему: "Мова програмування Go. Code Conventions"

Виконав ст. гр ПЗПІ-22-2
Єременко Андрій Віталійович

Перевірив доцент кафедри ПІ
Лещинський Володимир Олександрович

Харків 2024

МЕТА РОБОТИ

Мета – ознайомитися з основними принципами та конвенціями написання коду в Go. Ми розглянемо ключові правила, які допомагають зробити код читабельним, зрозумілим і відповідним до найкращих практик спільноти розробників Go.

ВИСНОВКИ

Дотримання код-стилю та конвенцій у Go покращує якість програмного забезпечення, роблячи його легшим для спільного використання, підтримки та розвитку, що сприяє зростанню ефективності командної роботи та забезпеченню високих стандартів коду.

Додаток А

GoLang

Code Convention

Andrii Yeremenko

Agenda

1. Вступ
2. Значення стандартизації коду для команди розробників
3. Організація проєкту та файлів
4. Форматування коду
5. Іменування
6. Коментарі та документування коду
7. Конвенції стилю коду
8. Код на основі тестування
9. Приклади оформлення коду

70%

Часу розробники читають код, а не пишуть його.

Як правильно організувати проєкт у Go?

Практичні поради:

- **Використовуйте змістовні імена директорій: уникайте загальних назв, як-от “utils” або “misc”.**
- **Дотримуйтесь рекомендацій Go Community, щоб ваш проєкт залишався зрозумілим іншим розробникам.**

Go рекомендує стандартну організацію проєкту:

```
myproject/
  ├── cmd/       // Вхідні точки програми
  ├── pkg/       // Загальний код, який можна повторно використовувати
  ├── internal/  // Приватні пакети (доступ лише в межах проєкту)
  ├── configs/   // Файли конфігурацій
  ├── test/      // Тести
  └── main.go    // Головний файл програми
```

Як правильно організувати проєкт у Go?

Практичні поради:

- **Використовуйте змістовні імена директорій: уникайте загальних назв, як-от “utils” або “misc”.**
- **Дотримуйтесь рекомендацій Go Community, щоб ваш проєкт залишався зрозумілим іншим розробникам.**

Go рекомендує стандартну організацію проєкту:

```
myproject/
  ├── cmd/       // Вхідні точки програми
  ├── pkg/       // Загальний код, який можна повторно використовувати
  ├── internal/  // Приватні пакети (доступ лише в межах проєкту)
  ├── configs/   // Файли конфігурацій
  ├── test/      // Тести
  └── main.go    // Головний файл програми
```

Форматування коду — стандарт у Go

```go
// getPort returns the port number to use for the server.
func getPort() (string, error) {  2 usages  Andrii Yeremenko
    port := getEnv( key: "PORT", DefaultPort)
    if _, err := strconv.Atoi(port); err != nil {
        return "", err
    }
    return port, nil
}

// startServer initializes and starts the web server.
func startServer(port, certFilePath, keyFilePath string, m *manager.ResourceManager) {  2 usages
    server := web_server.NewServerBuilder().
        SetPort(port).
        AddHandler( path: "/news", handler.NewNewsHandler(m).Handle).
        AddHandler( path: "/sources", handler.NewFeedsManagerHandler(m).Handle).
        AddHandler( path: "/availableFeeds", handler.NewAvailableFeedsHandler(m).Handle).
        Build()

    log.Println( v...: "Starting server on port " + port + " ...")

    err := server.ListenAndServeTLS(certFilePath, keyFilePath)
    if err != nil {
        log.Fatalf( format: "server failed to start: %v", err)
    }
    log.Println( v...: "Server started successfully")
}
```

Go використовує gofmt для автоматичного форматування.

Основні принципи:

- **Табуляція** замість пробілів.
- Довжина рядка: рекомендовано ≤ 80 символів.

Іменування

Go дотримується **CamelCase** для ідентифікаторів.

Імена, що починаються з великої літери, доступні **ззовні пакета**.

Використовуйте короткі, але змістовні назви.

Уникайте **однобуквених імен** для змінних, крім ітераторів (i, j).

Іменуйте константи відповідно до їхньої суті, уникайте магічних чисел.

Коментарі та документування коду

• Коментарі повинні пояснювати **логіку**, а не очевидний код.

• Використовуйте **документуючі коментарі** для функцій, структур, пакетів.

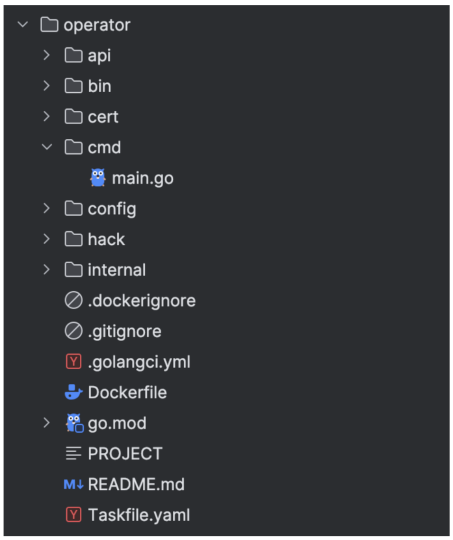
• Уникайте надлишкових або застарілих коментарів.

```
// Parser is a component that takes input resource.Resource and converts it into a structured and unified
// article.Article format.
type Parser interface {  9 usages  5 implementations  Andrii Yeremenko
    Parse(content resource.Resource) ([]article.Article, error)  5 implementations
}
```

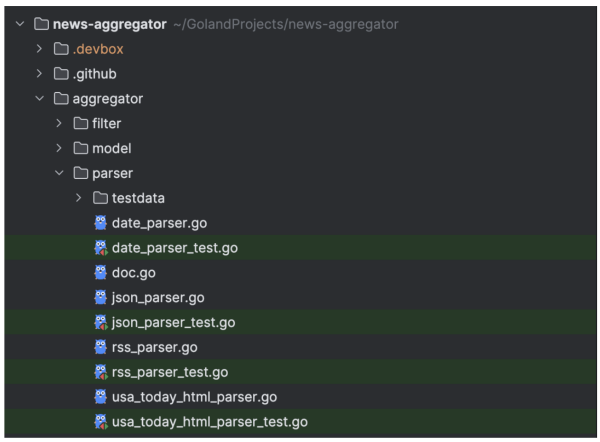
Конвенції стилю коду в Go

Go розроблено з акцентом на простоту та єдність стилю.

1. **Структура пакета:**
   - Ім'я пакета = ім'я каталогу (мінімум слів, без префіксів).
   - Головний файл: main.go для виконуваних програм.
2. **Експортовані та неекспортовані об'єкти:**
   - Імена з великої літери (PascalCase) — експортовані.
   - Імена з малої літери — доступні тільки в межах пакета.
3. **Лаконічність коду:**
   - Уникайте надмірної абстракції, пишіть простий, прямолінійний код.



Кодування на основі тестування (TDD) в Go



TDD: Спочатку пишемо тест, потім реалізуємо код.

- Тести в Go — це файли із суфіксом _test.go.
- Використовуйте пакет testing для створення тестів.

Приклади оформлення коду

```go
package parser

import (
    "encoding/json"
    "news-aggregator/aggregator/model/article"
    "news-aggregator/aggregator/model/resource"
    "strings"
)

// JSONParser is an aggregator.Parser that parses JSON data.
type JSONParser struct{}  10 usages  Andrii Yeremenko

type jsonArticle struct {  2 usages  Andrii Yeremenko
    Source struct {
        Name string `json:"name"`
    } `json:"source"`
    Author      string `json:"author"`
    Title       string `json:"title"`
    Description string `json:"description"`
    PublishedAt string `json:"publishedAt"`
    Link        string `json:"url"`
}

type jsonResponse struct {  3 usages  Andrii Yeremenko
    Articles []jsonArticle `json:"articles"`
}

// Parse parses the JSON content into a list of articles.
func (p *JSONParser) Parse(resource resource.Resource) ([]article.Article, error) {  Andrii Yeremenko
    byteContent := []byte(resource.Content())

    response, err := p.unmarshalJSON(byteContent)
    if err != nil { return nil, err }

    articles, err := p.extractArticles(response, resource)
    if err != nil { return nil, err }

    return articles, nil
}
```

Дякую за увагу!

Презентацію підготував Єременко Андрій. Ст. гр. ПЗПІ-22-2