TECHNOLOGIA

TWORZENIE WYDAJNYCH APLIKACJI WEBOWYCH Z WYKORZYSTANIEM JĘZYKA GO

mgr inż. Krzysztof Żabiński

magazyn.wsei.edu.pl

Wstęp

Go, potocznie nazywany często Golang, to stosunkowo młody język programowania, stworzony przez Google w 2007 roku i publicznie udostępniony w 2009 roku. Jego głównymi twórcami są Robert Griesemer, Rob Pike i Ken Thompson. Język ten powstał w odpowiedzi na rosnące potrzeby w zakresie prostoty, wydajności i łatwości utrzymania aplikacji w środowisku chmurowym oraz wielowątkowym.
Główną myślą przyświecającą twórcom było stworzenie języka, który będzie się szybko kompilować oraz będzie wydajny “at runtime”. Składnia języka jest wzorowana na innych językach C-pochodnych, próbując zebrać ich najlepsze cechy. Język jest silnie typowany, a aplikacje w nim tworzone w założeniu mają być stabilne oraz łatwe w utrzymaniu (kompilator sam wymusza wiele dobrych praktyk tworzenia oprogramowania). Go posiada wbudowaną obsługę wielowątkowości, dzięki czemu zrównoleglenie wykonywania operacji przy użyciu tego języka jest relatywnie proste. Znajduje zastosowanie przede wszystkim we współczesnych aplikacjach webowych, opartych o architekturę mikroserwisów, wykorzystujących REST API.

Historia

Go powstał w odpowiedzi na rozwój technologiczny, z którym mierzyło się wiele firm technologicznych, w tym Google, na początku XXI wieku. Ilość danych zaczęła zwiększać się wykładniczo, a co za tym idzie, tradycyjne metody obliczeniowe przestały być wystarczające do zapewnienia oczekiwanej wydajności. W związku z tym zaczęto budować rozwiązania oparte o rozproszone przetwarzanie danych oraz skalowalne. Do tego pojawiły się nowe wyzwania związane z właściwą obsługą sieci komputerowych, w oparciu o które one działają. Języki powstałe w zeszłym stuleciu nie były przystosowane do tego rodzaju wymagań. W związku z tym Google zdecydowało się na stworzenie własnego języka, który odpowie na te wyzwania (przy zachowaniu wydajności znanej z C/C++), łącząc ją z prostotą pisania kodu spotykaną w bardziej współczesnych językach interpretowanych, np. w Pythonie oraz z obsługą współbieżności i przystosowaniem do tworzenia aplikacji webowych.

Główne zalety języka Go

Wydajność: Go jest językiem kompilowanym do kodu maszynowego (w przeciwieństwie do języków kompilowanych do kodu pośredniego, np. Javy), który potrafi efektywnie zarządzać pamięcią. Dodatkowo posiada wbudowany garbage collector.
Prostota: Składnia jest zbliżona do języków C-pochodnych, a kompilator potrafi zweryfikować dobre praktyki tworzenia oprogramowania (przykładem jest wskazywanie nieużywanych zmiennych i wyrzucanie błędu kompilacji w przypadku ich wystąpienia).
Wbudowana współbieżność: goroutines oraz channels pozwalają na łatwe zrównoleglanie operacji wykonywanych przez aplikację, nadążając za infrastrukturalnym rozwojem technologicznym i pozwalając na radzenie sobie z przetwarzaniem dużych zbiorów danych.
Niezależność od platformy: kompilator Go występuje na wszystkie współcześnie występujące platformy (Windows, macOS, Linux). Dodatkowo, programy napisane w Go mogą być bez większych problemów uruchamiane na popularnych platformach chmurowych, np. AWS czy Google Cloud Platform.

Tworzenie aplikacji webowych

Najpopularniejszym zastosowaniem języka Go jest tworzenie aplikacji webowych (backendu tych aplikacji). Obsługa tego typu aplikacji jest wbudowana w język i nie ma konieczności wykorzystywania dodatkowych frameworków bądź bibliotek, aby tworzyć aplikacje webowe. Go słynie z wydajnego i równocześnie prostego w użyciu serwera HTTP (net/http). Poniżej prosty przykład wykorzystania serwera HTTP do stworzenia najprostszej aplikacji

```go
package main

import (
    "fmt"
    "net/http"
)

func helloHandler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, "Hello, World!")
}

func main() {
    http.HandleFunc("/", helloHandler)
    http.ListenAndServe(":8080", nil)
}
```

Powyższy kod uruchomi serwer HTTP na porcie 8080 i będzie wyświetlał wiadomość “Hello, World!”

Framework Gin

W tworzeniu aplikacji webowych zdecydowanie pomagają frameworki. Jest ich wiele, praktycznie dla każdego współcześnie używanego języka backendowego. Go nie jest tutaj wyjątkiem. Najbardziej znanym i najczęściej wykorzystywanym frameworkiem webowym dla Go jest Gin. Jest on tym, czym Spring jest dla Javy, a .NET dla C#. Oczywiście istnieją inne frameworki, jednak trudno uzasadnić ich wykorzystanie mając do dyspozycji właśnie Gina. Twórcy reklamują go jako “super łatwy” oraz “super szybki”. O ile są to oczywiście hasła marketingowe, to trudno się z nimi nie zgodzić.
Najważniejszą cechą frameworka jest jego wydajność. Tak naprawdę to wydajność była podstawowym kryterium, które przyświecało autorom przy jego tworzeniu. Framework jest zdecydowanie najszybszy oraz zużywa zdecydowanie najmniej pamięci ze wszystkich frameworków Go.
Kwestia łatwości użycia jest również warta podkreślenia. Framework oczywiście obsługuje wszystkie metody HTTP. Najważniejsza jest jednak łatwość definiowania routingu (przykład poniżej zobrazuje, jak można to osiągnąć).
Gin obsługuje middleware. Najprościej można to wyjaśnić na przykładzie obsługi zapytania HTTP. Obsługa middleware umożliwia przetwarzanie danego zapytania przed i po jego obsłużeniu. Przykłady zastosowania: logowanie, walidacja danych, zarządzanie sesjami itp.
Gin, w przeciwieństwie do wielu innych frameworków, posiada wbudowane natywne wsparcie dla enkodowania i dekodowania danych w formacie JSON I XML.
Celem zobrazowania jak łatwym i zrozumiałym frameworkiem jest Gin, poniżej przedstawiony jest przykład prostej aplikacji obsługującej routing:

```go
package main

import (
    "net/http"
    "github.com/gin-gonic/gin"
)

func main() {
    // Tworzenie domyślnego routera Gin
    router := gin.Default()

    // Obsługa podstawowej trasy GET na root
    router.GET("/", func(c *gin.Context) {
        c.String(http.StatusOK, "Witaj w aplikacji Gin!")
    })
```

```go
// Trasa z parametrem URL
router.GET("/hello/:name", func(c *gin.Context) {
    name := c.Param("name")
    c.String(http.StatusOK, "Witaj, %s!", name)
})

// Trasa GET z parametrami zapytania
router.GET("/query", func(c *gin.Context) {
    firstName := c.DefaultQuery("firstName", "Gość")
    lastName := c.Query("lastName")
    c.String(http.StatusOK, "Cześć, %s %s!", firstName, lastName)
})

// Obsługa trasy POST z danymi JSON
router.POST("/json", func(c *gin.Context) {
    var json struct {
        Message string `json:"message"`
    }
    if err := c.BindJSON(&json); err != nil {
        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid JSON"})
        return
    }
    c.JSON(http.StatusOK, gin.H{"received": json.Message})
})

// Uruchomienie serwera na porcie 8080
router.Run(":8080")
```

Krótkie wyjaśnienie podstawowych elementów przykładowego kodu

1. T**worzenie routera**

gin.Default() tworzy domyślny router, z domyślnymi ustawieniami middleware

2. **Obsługa metod HTTP GET**

router.GET("/", ...) definiuje trasę GET dla ścieżki, którą podajemy w pierwszym argumencie. Drugim argumentem jest funkcja, która w tym przypadku po prostu zwraca statyczny string

3. **Obsługa metod HTTP GET z parametrami:**

router.GET("/hello/:name", ...) w ten sposób możemy utworzyć router obsługujący dynamiczny paremetr:name, możemy go przekazać w adresie URL zapytania

4. **Routing z parametrami zapytania**

Jeśli chcemy, aby nasz serwer był w stanie pobrać parametry z adresu URL zapytania, możemy skorzystać z funkcji c.Query("param"), która pobiera wartość parametru param, a c.DefaultQuery("param", "default") umożliwia ustawienie wartości domyślnej

5. **Obsługa routingu POST z danymi JSON**

router.POST("/json", ...) obsługuje zapytania HTTP POST z danymi JSON. c.BindJSON(&json) mapuje parametry z ciała zapytania na strukturę json

6. **Uruchomienie serwera na wybranym porcie**:

router.Run(":8080") uruchamia serwer na porcie 8080

Porównanie wydajności

Póki co przedstawione zostały tylko i wyłącznie suche fakty, podkreślające zalety języka Go oraz wysoką wydajność frameworka Gin. Celem zobrazowania jak dużym skokiem wydajnościowym może być zmiana technologii właśnie na Go Gin, przedstawione jest porównanie z jednym z najpopularniejszych na świecie frameworków dla języka Java, tj. Spring Boot. Porównanie dotyczy prostej aplikacji webowej typu “Hello world!”, obsługującej jedno zapytanie HTTP GET, zwracające jeden statyczny string.

Kod aplikacji napisanej w Go z frameworkiem Gin:

```go
package main

import (
  "net/http"
  "github.com/gin-gonic/gin"
)

func main() {
  r := gin.New()

  r.GET("/", func(c *gin.Context) {
    c.String(http.StatusOK, "Hello world!")
  })

  r.Run(":3000")
}
```

Kod analogicznej aplikacji napisanej w Javie z wykorzystaniem Spring Boot:

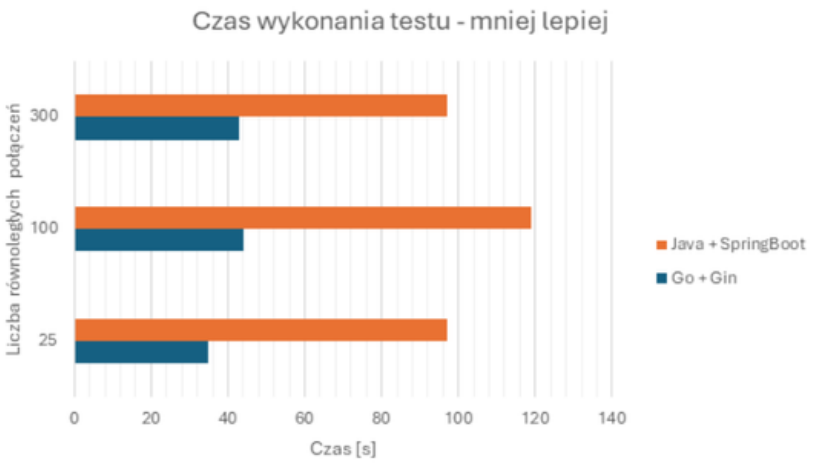
```java
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.http.ResponseEntity;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
```

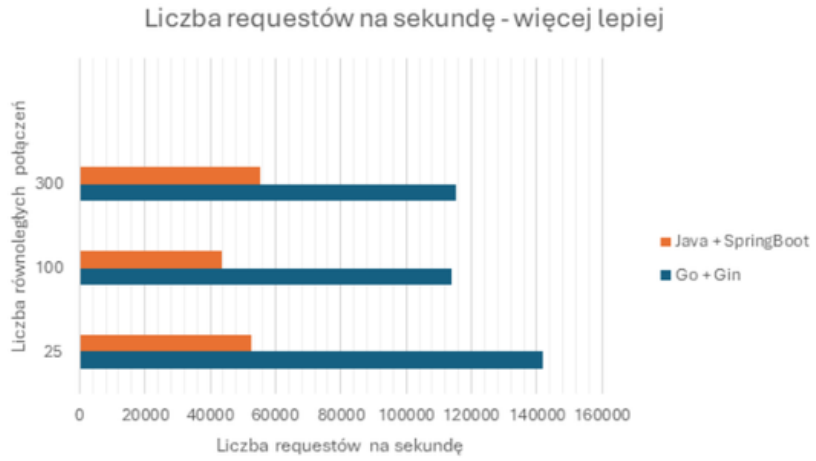
```java
@RestController
public class DemoApplication {

  public static void main(String[] args) {
    SpringApplication.run(DemoApplication.class, args);
  }

  @GetMapping("/")
  public String handleRequest() {
    return "Hello world!";
  }
}
```

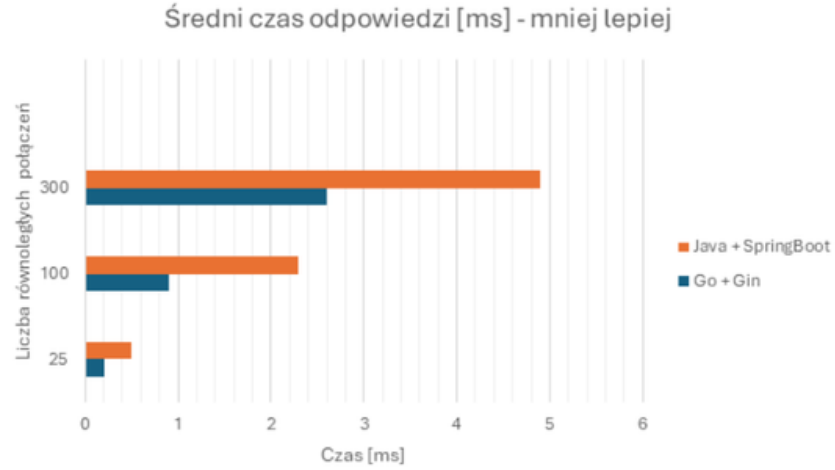
GO JEST WYDAJNYM, SZYBKIM, MULTIPLATFORMOWYM JĘZYKIEM PROGRAMOWANIA, KTÓRY ZNALAZŁ SZEROKIE ZASTOSOWANIE WE WSPÓŁCZESNYCH APLIKACJACH WEBOWYCH. MOŻNA POWIEDZIEĆ, ŻE GO KONSOLIDUJE ZALETY WSZYSTKICH JĘZYKÓW C-POCHODNYCH, STARAJĄC SIĘ ROZWIĄZAĆ TRAPIĄCE JE PROBLEMY

Do wykonania testów obciążeniowych wykorzystane zostało narzędzie Bombardier HTTP, w każdej sesji testowej wysłano 5 milionów requestów, w następujących konfiguracjach: 25, 100 oraz 300 równoległych połączeń. Testy wykonano na komputerze Macbook Pro z procesorem M1 i 16 GB pamięci operacyjnej.

Wyniki porównania przedstawione są poniżej:







Jak widać na powyższym porównaniu, wyniki wydajnościowe są wręcz spektakularne na korzyść Go. Wybrane miary jakości wskazują, że dla tego prostego przykładu wydajność Go z frameworkiem Gin jest średnio dwukrotnie lepsza niż w przypadku aplikacji napisanej w języku Java z frameworkiem Spring Boot. Eksperyment ten potwierdza wskazane wcześniej zalety Go. Biorąc pod uwagę, że współczesne aplikacje webowe są o wiele bardziej skomplikowane niż ta prosta aplikacja testowa, korzystając z języka Go możemy oszczędzić ogromne ilości czasu obliczeniowego, co może mieć realny wpływ na znaczącą poprawę wydajności produktów opartych o Go.

Podsumowanie

Go jest wydajnym, szybkim, multiplatformowym językiem programowania, który znalazł szerokie zastosowanie we współczesnych aplikacjach webowych. Do szerokiego rozpropagowania przyczyniły się z pewnością kwestie marketingowe związane z tym, że za stworzeniem języka stoi Google. Natomiast to nie marketing się tutaj broni, sam w sobie język oraz jego założenia. Można powiedzieć, że Go konsoliduje zalety wszystkich języków C-pochodnych, starając się rozwiązać trapiące je problemy. Tak naprawdę główne jego wady dotyczą w zasadzie faktu, że język jest relatywnie młody. Dotyczą one ograniczonego wsparcia bibliotek i historycznych problemów z zarządzaniem dependencjami. Niemniej jednak, zalety znacząco przeważają nad wadami i zdecydowanie warto spróbować zacząć swoją przygodę z Go.

Bibliografia

1. https://go.dev/
2. https://gin-gonic.com/
3. https://spring.io/
4. https://medium.com/deno-the-complete-reference/go-gin-vs-springboot-hello-world-perfomance-comparison-e535c9d6c36

NAJWAŻNIEJSZĄ CECHĄ FRAMEWORKA GIN JEST JEGO WYDAJNOŚĆ. TAK NAPRAWDĘ TO WYDAJNOŚĆ BYŁA PODSTAWOWYM KRYTERIUM, KTÓRE PRZYŚWIECAŁO AUTOROM PRZY JEGO TWORZENIU. FRAMEWORK JEST ZDECYDOWANIE NAJSZYBSZY ORAZ ZUŻYWA ZDECYDOWANIE NAJMNIEJ PAMIĘCI ZE WSZYSTKICH FRAMEWORKÓW GO