



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Компьютерные системы и сети

О Т Ч Е Т

по лабораторной работе № 5

Название: Основы асинхронного программирования на Golang

Дисциплина: Языки интернет-программирования

Студент	<u>ИУ6-32Б</u> (Группа)	<u>01.10.2024</u> (Подпись, дата)	<u>Т.А. Гаджиев</u> (И.О. Фамилия)
Преподаватель		<u>01.10.2024</u> (Подпись, дата)	<u>В.Д. Шульман</u> (И.О. Фамилия)

Москва, 2024

Цель работы: изучение основ асинхронного программирования с использованием языка Golang

Ход работы.

1. Ознакомились с курсом <https://stepik.org/course/54403/info>
2. Сделали форк данного репозитория в GitHub, клонировали получившуюся копию локально, создали от мастера ветку дев и переключились на нее:

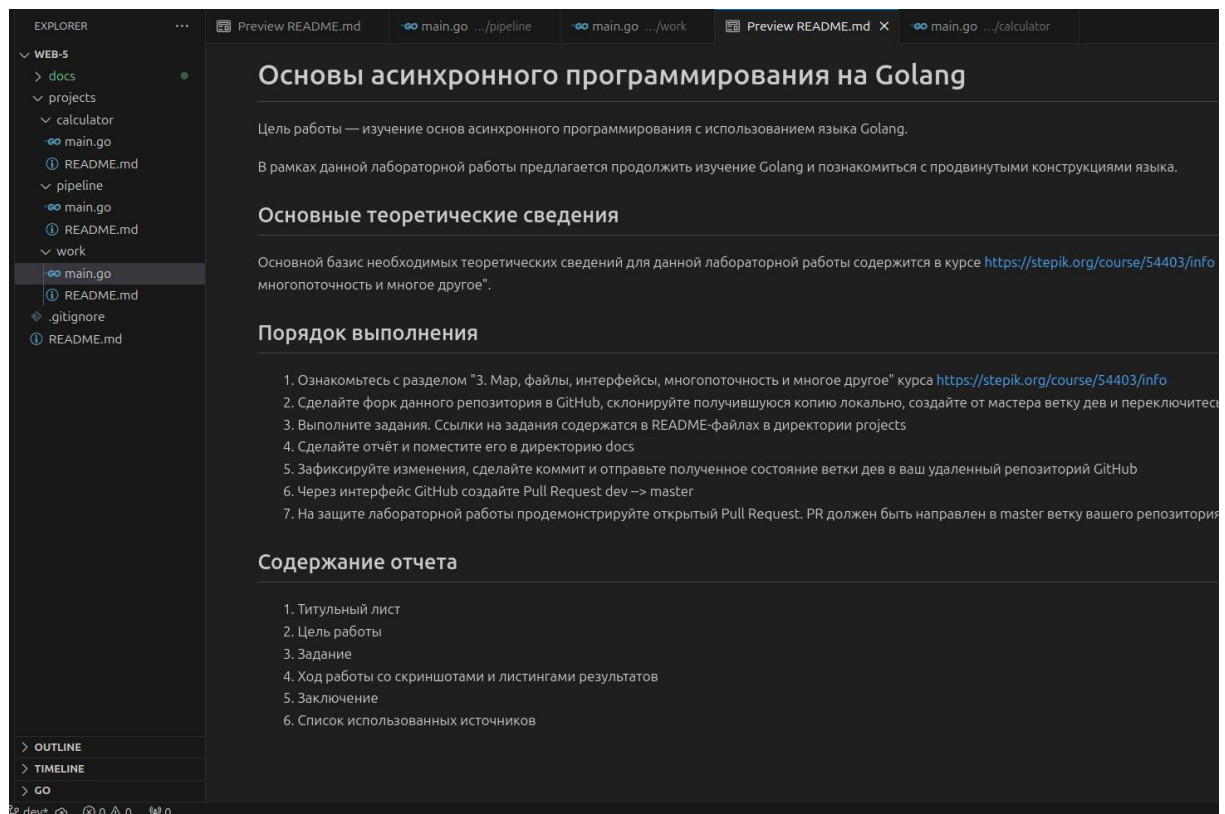


Рисунок 1 — Скопированный репозиторий

3. Решили 3 задачи на языке GoLang. Коды задач и результаты их работы прикрепили ниже:

Задача 1(Calculator):

```
package main
import (
    "fmt"
    "time"
)
func calculator(firstChan <-chan int, secondChan <-chan int, stopChan <-chan struct{}) <-chan int {
    output := make(chan int)
    go func() {
        defer close(output) // Отложенная функция
```

```

for {
select {
case x := <-firstChan:
output <- x * x
case x := <-secondChan:
output <- x * 3
case <-stopChan:
return // надо завершить только после получения сигнала остановки
}
}
}()
return output
}
func main() {
firstChan := make(chan int)
secondChan := make(chan int)
stopChan := make(chan struct{})
outputChan := calculator(firstChan, secondChan, stopChan)
go func() {
time.Sleep(5 * time.Second)
close(stopChan)
}()
// горутинка, которая будет отправлять данные в каналы
go func() {
firstChan <- 5
secondChan <- 10
firstChan <- 21
secondChan <- 13
}()
for result := range outputChan {
fmt.Println(result)
}
}

```

The screenshot shows a Go IDE with a code editor and an output window. The code in the editor is a Go function that takes two channels and a stop channel, and returns a channel of results. The output window shows the results of the function: 25, 30, 441, and 39.

```

42      go func() {
43          firstChan <- 5
44          secondChan <- 10
45          firstChan <- 21
46          secondChan <- 13
47      }()
48

```

PROBLEMS OUTPUT DEBUG CONSOLE TESTS

```

25
30
441
39

```

Рисунок 2 — Вывод задачи 1

Задача 2(Pipeline):

```

package main
import "fmt"
func removeDuplicates(inputStream, outputStream chan string) {
var prevValue string
for value := range inputStream {
if value != prevValue {
outputStream <- value
prevValue = value
}
}
close(outputStream)
}
func main() {
inputStream := make(chan string)
outputStream := make(chan string)
go func() {
inputStream <- "apple"
inputStream <- "banana"
inputStream <- "banana"
inputStream <- "cherry"
inputStream <- "apple"
inputStream <- "date"
close(inputStream)
}()
go removeDuplicates(inputStream, outputStream)
for value := range outputStream {
fmt.Println(value)
}
}

```

The screenshot shows a Go IDE with a dark theme. The editor displays a Go function `go func() {` with several lines of code assigning values to `inputStream` and then closing it. Below the editor, there are four tabs: **PROBLEMS**, **OUTPUT**, **DEBUG CONSOLE**, and **TERMINAL**. The **TERMINAL** tab is active, showing the output of the program: `apple`, `banana`, `cherry`, `apple`, and `date`, each on a new line.

```

20      go func() {
21          inputStream <- "apple"
22          inputStream <- "banana"
23          inputStream <- "banana"
24          inputStream <- "cherry"
25          inputStream <- "apple"
26          inputStream <- "date"
27          close(inputStream)
28      }()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```

apple
banana
cherry
apple
date

```

Рисунок 3 — Вывод задачи 2

Задача 3(Work):

```

package main

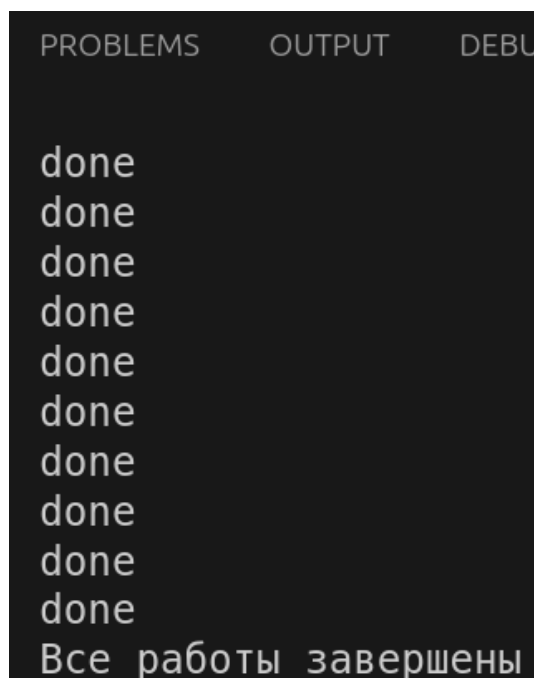
import (
    "fmt"
    "sync"
    "time"
)

func work() {
    time.Sleep(time.Millisecond * 50)
    fmt.Println("done")
}

func main() {
    var wg sync.WaitGroup
    wg.Add(10)

    for i := 0; i < 10; i++ {
        go func() {
            defer wg.Done()
            work()
        }()
    }
    wg.Wait()
    fmt.Println("Все работы завершены") //для проверки, на степике строчку убрать
}

```



The screenshot shows the output of a Go program. At the top, there are three tabs: 'PROBLEMS', 'OUTPUT', and 'DEBUG'. The 'OUTPUT' tab is selected, displaying the following text:

```

done
done
done
done
done
done
done
done
done
done
Все работы завершены

```

Рисунок 4 — Вывод задачи 3

4. Зафиксировали изменения, сделали коммит и отправили полученное состояние ветки dev в удаленный репозиторий GitHub. Через интерфейс GitHub создали Pull Request dev --> master

Заключение: в ходе лабораторной работы изучили основы асинхронного программирования с использованием языка Golang. Освоили управление потоками и решили 3 задачи на эту тему.