

Лабораторна робота №3

Sockets

Зміст

1. Лабораторна робота 3	2
1.1 Постановка задачі	2
1.2 Вимоги до роботи	2
1.3 Вимоги до оформлення звіту	2
1.4 Варіанти завдань	3
2. Рекомендації по виконанню роботи	5
2.1 Попередні налаштування.....	5
2.2 Предметна область та будова додатку.....	6
2.3 Hub.....	6
2.4 Client.....	7
2.5 Main	10
2.6 FrontEnd.....	11
2.7 Запуск додатку	13

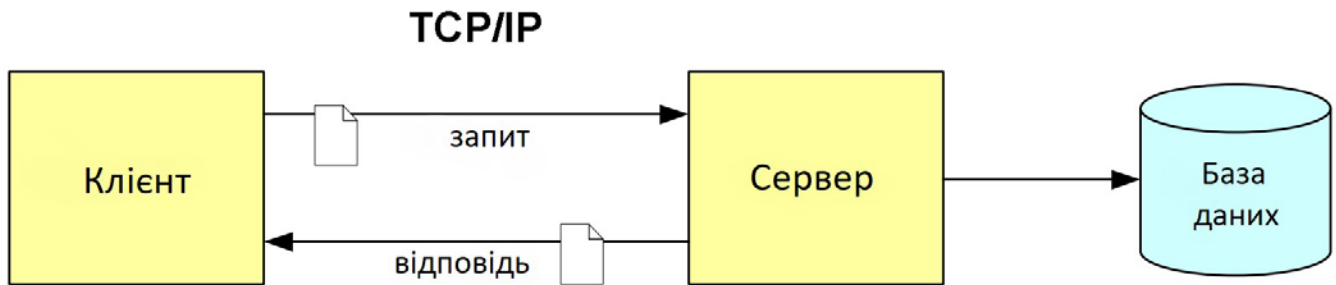
1. Лабораторна робота 3

1.1 Постановка задачі

На основі програми, розробленої в рамках лабораторної роботи No2, створити серверну програму, що забезпечує виконання віддалених запитів з управління об'єктами. Відомості про об'єкти повинні зберігатися в базі даних.

Розробити клієнтську програму, що відправляє серверу запити на введення, редагування і отримання інформації про об'єкти.

Взаємодія між клієнтом і сервером має здійснюватися по протоколу TCP / IP.



1.2 Вимоги до роботи

Сервер запускається у фоновому режимі і не має інтерфейсу користувача. Сервер повинен забезпечувати виконання операцій, представлених у вашому варіанті завдання.

На кожен запит клієнта сервер повинен відправляти відповідь. Відповіді сервера повинні відрізнятися в залежності від результату виконання запиту клієнта (позитивна / негативна відповідь).

Сервер орієнтований на взаємодію з єдиним клієнтом. Підтримку взаємодії сервера з декількома клієнтами реалізовувати не потрібно.

Сервер повинен забезпечувати унікальність ідентифікаторів об'єктів при виконанні операцій додавання і редагування.

Клієнтська програма відправляє запити серверу і демонструє відповіді користувачу. Клієнтська програма не вимагає створення інтерфейсу користувача. Тестування працездатності клієнта здійснюється на основі сценаріїв, які демонструють можливості програми.

Формат інформаційних повідомлень, якими обмінюються клієнт і сервер, визначається відповідно до номеру варіанта (див. Розділ 2.3).

Рекомендована мова програмування – Go.

1.3 Вимоги до оформлення звіту

Звіт повинен містити:

- титульну сторінку
- постановку задачі
- вихідний код програми (включаючи web.xml, таблиці css, і т.п.)

- опис програми (опис класів, методів, полів)
- опис протоколу взаємодії клієнта і сервера (тобто ДЕТАЛЬНИЙ опис формату та структури ВСІХ типів інформаційних повідомлень, якими обмінюються клієнт і сервер між собою, по аналогії з табл. 1 і 2).

1.4 Варіанти завдань

Варіант 1

Предметна область Карта Світу
Об'єкти Автори, Книги

Карта світу містить множину *країн*. Для кожної *країни* визначена множина *міст*.

Необхідні операції Отримання повного списку міст із зазначенням назви країни

Варіант 2

Предметна область Бібліотека
Об'єкти Автори, Книги

Книги в бібліотеці згруповані по *авторам*. У кожного *автора* є множина *книг*.

Необхідні операції Отримання повного списку книг із зазначенням ПІБ автора

Варіант 3

Предметна область Відділ кадрів
Об'єкти Підрозділи, Співробітники

Існує множина *підрозділів* підприємства. У кожному *підрозділі* працює множина *співробітників*.

Необхідні операції Отримання списку співробітників із зазначенням назви підрозділу

Варіант 4

Предметна область Навчальний відділ
Об'єкти Групи, Студенти

Існує множина навчальних *груп*. Кожна група включає в себе множину *студентів*.

Необхідні операції Отримання повного списку студентів із зазначенням назви групи

Варіант 5

Предметна область Автосалон
Об'єкти Виробники автомобілів, Марки

Марки автомобілів згруповані по виробникам. У кожного *виробника* є множина *марок*.

Необхідні операції Отримання повного списку марок з назвою виробника

Варіант 6

Предметна область Агентство новин
Об'єкти Категорії новин, Новини

Новини згруповані по *категоріям*. У кожній *категорії* є множина *новин*.

Необхідні операції Отримання повного списку новин із зазначенням категорії

Варіант 7

Предметна область Продуктовий магазин
Об'єкти Категорія продукту, Продукт

Продукти в магазині згруповані за *категоріями*. Для кожної *категорії* визначено множину *продуктів*.

Необхідні операції Отримання списку продуктів із зазначенням категорії

Варіант 8

Предметна область Футбол
Об'єкти Команди, Гравці

Існує множина футбольних команд. Для кожної команди визначено множину гравців.
Необхідні операції Отримання повного списку гравців із зазначенням назви команди

Варіант 9

Предметна область Музичний магазин
Об'єкти Виконавці, Альбоми

У музичному магазині альбоми згруповані за виконавцями. Для кожного виконавця задано множину альбомів.
Необхідні операції Отримання повного списку альбомів із зазначенням виконавця

Варіант 10

Предметна область Аеропорт
Об'єкти Авіакомпанії, Рейси

Існує множина авіакомпаній. Для кожної авіакомпанії визначені її рейси.
Отримання повного списку рейсів із зазначенням назви авіакомпанії

Варіант 11

Предметна область Файлова система
Об'єкти Папки, Файли

Існує множина папок (незалежних один від одного). Для кожної папки визначено множину файлів.
Необхідні операції Отримання списку файлів із зазначенням папки

Варіант 12

Предметна область Розклад занять
Об'єкти Дні тижня, Заняття
Примітка Існує множина днів. Для кожного дня визначено перелік занять.
Необхідні операції Отримання повного списку занять із зазначенням дня

Варіант 13

Предметна область Записна книжка
Об'єкти Календарні дні, Заходи
Примітка Існує множина днів. Для кожного дня визначено перелік заходів.
Необхідні операції Отримання повного списку заходів із зазначенням дня

Варіант 14

Предметна область Відео-магазин
Об'єкти Жанри, Фільми

Існує множина жанрів. Для кожного жанру визначений перелік фільмів.
Необхідні операції Отримання списку фільмів із зазначенням жанру

Варіант 15

Предметна область Залізнична дорога
Об'єкти Дороги, Станції

Існує множина залізниць. У відомстві кожної дороги знаходиться множина станцій.
Необхідні операції Отримання повного списку станцій із зазначенням назви дороги

Варіант 16

Предметна область	Склад
Об'єкти	Секції, Товари

Товари на складі згруповані по *секціях*. Для кожної *секції* задано множину *товарів*.

Необхідні операції	Отримання списку товарів із зазначенням секції
--------------------	--

Варіант 17

Предметна область	Кафедра університету
Об'єкти	Викладачі, Дисципліни

На кафедрі існує множина *викладачів*. Для кожного *викладача* задано множину *дисциплін*.

Необхідні операції	Отримання списку дисциплін із зазначенням ПІБ викладача
--------------------	---

Варіант 18

Предметна область	Програмне забезпечення
Об'єкти	Виробники, Програмні продукти

Програмні *продукти* згруповані по *виробникам*. Для кожного *виробника* задано множину *продуктів*.

Необхідні операції	Отримання списку продуктів із зазначенням виробника
--------------------	---

Варіант 19

Предметна область	Геометрія
Об'єкти	Багатокутники, Вершини

Існує множина *багатокутників*. Кожен *багатокутник* складається з довільної кількості вершин.

Отримання повного списку багатокутників з зазначенням всіх вершин

Варіант 20


Предметна область	Схема метро
Об'єкти	Лінії, Станції

Існує множина *ліній* метрополітену. Кожна *лінія* складається з послідовності *станцій*.

Необхідні операції	Отримання списку станцій із зазначенням лінії
--------------------	---

2. Рекомендації по виконанню роботи

2.1 Попередні налаштування

Для розробки додатку можна використовувати будь-який текстовий редактор (хоч ) , потрібно мати встановлений go, а також нам знадобиться бібліотека gorilla/websocket, яку можна встановити наступною командою:

```
go get -u github.com/gorilla/websocket
```

Також створюємо go module:

```
go mod init ws-chat
```

```
go mod tidy
```

2.2 Предметна область та будова додатку

Реалізуємо простий веб-чат. Для серверної сторони визначимо два типи: `Client` та `Hub`. Сервер створює екземпляр `Client` для кожного websocket з'єднання. Сам `Client` слугує посередником між websocket з'єднанням та єдиним екземпляром `Hub`, який у свою чергу реєструє клієнтів та транслює повідомлення.

Додаток запускає одну `goroutine` для `Hub` та дві для кожного `Client`. `Goroutines` комунікують через канали: у `Hub` це канали для реєстрації і видалення клієнтів та для транслювання повідомлень. Клієнт містить буферизований канал для вихідних повідомлень. Одна з рутин клієнта зчитує повідомлення з цього каналу і надсилає повідомлення до websocket, а інша зчитує з websocket та надсилає до `Hub`.

2.3 Hub

У цьому підпункті зміни будуть вноситися тільки до файлу `hub.go`.

```
package main

type Hub struct {
    clients map[*Client]bool

    broadcast chan []byte

    register chan *Client

    unregister chan *Client
}

func newHub() *Hub {
    return &Hub{
        broadcast: make(chan []byte),
        register:  make(chan *Client),
        unregister: make(chan *Client),
        clients:   make(map[*Client]bool),
    }
}

func (h *Hub) run() {
    for {
        ...
    }
}
```

Хаб реєструє користувачів, додаючи посилання на об'єкт як ключ до `map`. Його значення завжди `true`.

...

```

select {
case client := <-h.register:
    h.clients[client] = true
...

```

Для видалення користувача потрібно буде не тільки видалити посилання на об'єкт з `map`, але й закрити клієнтський `send` канал, аби повідомити клієнту, що повідомлень надходити більше не буде.

```

...

case client := <-h.unregister:
    if _, ok := h.clients[client]; ok {
        delete(h.clients, client)
        close(client.send)
    }
...

```

Хаб відправляє повідомлення, циклічно проходячи по клієнтах з `map` та відправляючи повідомлення на клієнтський `send` канал. Якщо буфер цього каналу переповнений, то хаб вважає, що клієнт або не відповідає, або не функціонує. У такому випадку, хаб видаляє цього клієнта та закриває `websocket`.

```

...

case message := <-h.broadcast:
    for client := range h.clients {
        select {
        case client.send <- message:
        default:
            close(client.send)
            delete(h.clients, client)
        }
    }
}

```

2.4 Client

У цьому підпункті зміни будуть вноситися тільки до файлу `client.go`.

```

package main

import (
    "bytes"
    "log"
    "net/http"

```

```

    "time"

    "github.com/gorilla/websocket"
)

const (
    // Time allowed to write a message to the peer.
    writeWait = 10 * time.Second

    // Time allowed to read the next pong message from the peer.
    pongWait = 60 * time.Second

    // Send pings to peer with this period. Must be less than pongWait.
    pingPeriod = (pongWait * 9) / 10

    // Maximum message size allowed from peer.
    maxMessageSize = 512
)

var (
    newline = []byte{'\n'}
    space   = []byte{' '}
)

var upgrader = websocket.Upgrader{
    ReadBufferSize: 1024,
    WriteBufferSize: 1024,
}

type Client struct {
    hub *Hub

    conn *websocket.Conn

    send chan []byte
}

...

```

У функції `main` додатку функція `servesWs` визначається як HTTP handler, який перевизначає HTTP з'єднання як WebSocket протокол, створює клієнта, реєструє його в хабі та відкладає його видалення, використовуючи `defer`.

Після цього, HTTP handler запускає клієнтський метод `writePump` як goroutine. Цей метод передає повідомлення з `send` каналу клієнта до `websocket` з'єднання та завершується коли або канал був закритий хабом, або під час передачі повідомлення сталася помилка.

Аби підвищити ефективність додатку під час великого навантаження, `writePump` збиває до купи вихідні не відправлені повідомлення в одне. Таким чином, зменшується кількість системних викликів та кількість даних, що передається мережею.

...


```

func (c *Client) writePump() {
    ticker := time.NewTicker(pingPeriod)
    defer func() {
        ticker.Stop()
        c.conn.Close()
    }()
    for {
        select {
        case message, ok := <-c.send:
            c.conn.SetWriteDeadline(time.Now().Add(writeWait))
            if !ok {
                c.conn.WriteMessage(websocket.CloseMessage, []byte{})
                return
            }

            w, err := c.conn.NextWriter(websocket.TextMessage)
            if err != nil {
                return
            }
            w.Write(message)

            n := len(c.send)
            for i := 0; i < n; i++ {
                w.Write(newline)
                w.Write(<-c.send)
            }

            if err := w.Close(); err != nil {
                return
            }
        case <-ticker.C:
            c.conn.SetWriteDeadline(time.Now().Add(writeWait))
            if err := c.conn.WriteMessage(websocket.PingMessage, nil); err != nil {
                return
            }
        }
    }
}

...

```

Нарешті, HTTP handler викликає клієнтський метод `readPump`, який передає вхідні повідомлення від `webSocket` з'єднання до хабу.

```

...

func (c *Client) readPump() {
    defer func() {
        c.hub.unregister <- c
        c.conn.Close()
    }()
    c.conn.SetReadLimit(maxMessageSize)
    c.conn.SetReadDeadline(time.Now().Add(pongWait))
    c.conn.SetPongHandler(func(string) error {
        c.conn.SetReadDeadline(time.Now().Add(pongWait)); return nil
    })
    for {

```

```

_, message, err := c.conn.ReadMessage()
if err != nil {
    if websocket.IsUnexpectedCloseError(err, websocket.CloseGoingAway,
websocket.CloseAbnormalClosure) {
        log.Printf("error: %v", err)
    }
    break
}
message = bytes.TrimSpace(bytes.Replace(message, newline, space, -1))
c.hub.broadcast <- message
}
}

func serveWs(hub *Hub, w http.ResponseWriter, r *http.Request) {
    conn, err := upgrader.Upgrade(w, r, nil)
    if err != nil {
        log.Println(err)
        return
    }
    client := &Client{hub: hub, conn: conn, send: make(chan []byte, 256)}
    client.hub.register <- client

    go client.writePump()
    go client.readPump()
}

```

2.5 Main

У цьому підпункті зміни будуть вноситися тільки до файлу `main.go`.

```

package main

import (
    "flag"
    "log"
    "net/http"
)

var addr = flag.String("addr", ":8080", "http service address")

func serveHome(w http.ResponseWriter, r *http.Request) {
    log.Println(r.URL)
    if r.URL.Path != "/" {
        http.Error(w, "Not found", http.StatusNotFound)
        return
    }
    if r.Method != "GET" {
        http.Error(w, "Method not allowed", http.StatusMethodNotAllowed)
        return
    }
    http.ServeFile(w, r, "home.html")
}

func main() {
    flag.Parse()
    hub := newHub()
    go hub.run()
    http.HandleFunc("/", serveHome)
}

```

```

http.HandleFunc("/ws", func(w http.ResponseWriter, r *http.Request) {
    serveWs(hub, w, r)
})
err := http.ListenAndServe(*addr, nil)
if err != nil {
    log.Fatal("ListenAndServe: ", err)
}
}

```

Тут ми визначаємо адресу та порт для запуску сервера. Main функція додатку запускає метод run хабу як goroutine, а також функція serveWs клієнта визначається як HTTP handler.

2.6 FrontEnd

У цьому підпункті зміни будуть вноситися тільки до файлу home.html.

Після завантаження документа, скрипт перевіряє чи підтримуються websockets в браузері. Якщо так, то скрипт встановлює зв'язок з сервером та створює колбек-функцію, аби обробляти вхідні повідомлення від сервера. Ця функція додає повідомлення в кінець чату, використовуючи функцію appendLog.

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Chat Example</title>
    <script type="text/javascript">
        window.onload = function () {
            var conn;
            var msg = document.getElementById("msg");
            var log = document.getElementById("log");

            function appendLog(item) {
                var doScroll = log.scrollTop > log.scrollHeight -
log.clientHeight - 1;
                log.appendChild(item);
                if (doScroll) {
                    log.scrollTop = log.scrollHeight - log.clientHeight;
                }
            }

            ...

```

Аби у користувача була можливість власноруч гортати сторінку чату без зміни прокрутки при надходженні нового повідомлення, функція appendLog перевіряє позицію прокрутки перед додаванням нового повідомлення. Якщо чат прокручено вниз, тоді функція прокручує чат донизу після додавання повідомлення. Інакше, прокрутка не змінюється.

```

...

document.getElementById("form").onsubmit = function () {
    if (!conn) {

```

```

        return false;
    }
    if (!msg.value) {
        return false;
    }
    conn.send(msg.value);
    msg.value = "";
    return false;
};

if (window["WebSocket"]) {
    conn = new WebSocket("ws://" + document.location.host + "/ws");
    conn.onclose = function (evt) {
        var item = document.createElement("div");
        item.innerHTML = "<b>Connection closed.</b>";
        appendLog(item);
    };
    conn.onmessage = function (evt) {
        var messages = evt.data.split('\n');
        for (var i = 0; i < messages.length; i++) {
            var item = document.createElement("div");
            item.innerText = messages[i];
            appendLog(item);
        }
    };
} else {
    var item = document.createElement("div");
    item.innerHTML = "<b>Your browser does not support
WebSockets.</b>";
    appendLog(item);
}
};
</script>
<style type="text/css">
    html {
        overflow: hidden;
    }

    body {
        overflow: hidden;
        padding: 0;
        margin: 0;
        width: 100%;
        height: 100%;
        background: gray;
    }

    #log {
        background: white;
        margin: 0;
        padding: 0.5em 0.5em 0.5em 0.5em;
        position: absolute;
        top: 0.5em;
        left: 0.5em;
        right: 0.5em;
        bottom: 3em;
        overflow: auto;
    }

    #form {
        padding: 0 0.5em 0 0.5em;

```

```

        margin: 0;
        position: absolute;
        bottom: 1em;
        left: 0px;
        width: 100%;
        overflow: hidden;
    }

</style>
</head>
<body>
<div id="log"></div>
<form id="form">
    <input type="submit" value="Send" />
    <input type="text" id="msg" size="64" autofocus />
</form>
</body>
</html>

```

Обробник форми надсилає введені дані користувача до websocket та очищує поле вводу.

2.7 Запуск додатку

Для запуску додатку запустіть main функцію у main.go файлі, перейдіть до localhost:8080 у браузері, вводьте повідомлення та натискайте на кнопку “send”.

It's server magic.

