**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №3

«Хранение статических данных PostgreSQL»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Старкин Александр |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г.

### Задание:

Разработать алгоритм поиска пересечения двух расписаний. Под пересечением понимается 2 числа : 1я общая по времени пара и последняя за один день. Алгоритм должен обрабатывать corner кейсы, такие как:

* Выходной у одного из человек
* Расписание не составлено у одной из группы (или по какой-либо другой причине оно пустое )

1. Расписание отдельной группы программа получает путем обращения по адресу https://lks.bmstu.ru/lks-back/api/v1/schedules/groups/" + uid + "/public к API МГТУ им. Баумана.
2. Entity для объекта расписания выглядит как

type IDay struct {

    Day        int         `json:"day"`

    Time       int         `json:"time"`

    Week       string      `json:"week"`

    Groups     []IGroup    `json:"groups"`

    Stream     string      `json:"stream"`

    EndTime    string      `json:"endTime"`

    Teachers   []ITeacher  `json:"teachers"`

    Audiences  []IAudience `json:"audiences"`

    StartTime  string      `json:"startTime"`

    Discipline IDiscipline `json:"discipline"`

    Permission string      `json:"permission"`

}

Текст программы

package notify

import (

    "errors"

    entities "github.com/Robocotik/IT-s\_Friends/internal/models/entities"

)

func GetTimetableIntarsection(meSchedule []entities.IDay, friendSchedule []entities.IDay, day int) (int, int, error) {

    first := 20

    second := -1

*if* len(meSchedule) == 0 || len(friendSchedule) == 0 {

*return* -1, -1, errors.New("schedule empty")

    }

*for* \_, meDay := *range* meSchedule {

*if* meDay.Discipline.Abbr == "Самостоятельная работа" && meDay.Day == day {

*return* -1, -1, errors.New("rest time")

        }

    }

*for* \_, friendDay := *range* friendSchedule {

*if* friendDay.Discipline.Abbr == "Самостоятельная работа" && friendDay.Day == day {

*return* -1, -1, errors.New("rest time")

        }

    }

*for* \_, friendDay := *range* friendSchedule {

*if* friendDay.Day != day {

*continue*

        }

*for* \_, meDay := *range* meSchedule {

*if* friendDay.Day == day && day == meDay.Day {

*if* friendDay.Time == meDay.Time {

*if* friendDay.Time < first {

                        first = friendDay.Time

                    }

*if* meDay.Time > second {

                        second = meDay.Time

                    }

                }

            }

        }

    }

*if* (first == 20 && second == -1){

*return* -1, -1, errors.New("no interaction")

    }

*return* first, second, nil

}