1. HTTP // Habr [Электронный ресурс]. ‒ 2020. ‒ Режим доступа: <https://habr.com/ru/post/215117/>. ‒ Дата доступа: 01.05.2020.
2. Создание сайто // Inprocess [Электронный ресурс]. ‒ 2020. ‒ Режим доступа: <https://inprocess.by/>. ‒ Дата доступа: 01.05.2020.
3. Уникального сайта // croomg [Электронный ресурс]. ‒ 2020. ‒ Режим доступа: <https://www.divier.ru/stati/unikalnyy_dizayn_sayta/>. ‒ Дата доступа: 01.05.2020.

ПРИЛОЖЕНИЕ А

Диаграмма вариантов использования

ПРИЛОЖЕНИЕ Б

Общая схема взаимодействия компонентов

ПРИЛОЖЕНИЕ В

Логическая схема базы данных

# ПРИЛОЖЕНИЕ Г

Листинги кода серверной части

а) Листинг пакета main

package main

import (

"./service"

"flag"

log "github.com/sirupsen/logrus"

)

func main() {

defer func() {

if err := recover(); err != nil {

log.Error(err)  
 }

}()

err := service.InitLog()

if err != nil {

log.Error("Cannot init logs: ", err)

}

log.Info("Starting...")

var cnf = flag.String("c", "conf.json", "Config file name (in a current dir)")

flag.Parse()

log.Info("Loading with config: ", \*cnf)

service.InitConfiguration(\*cnf, service.Conf)

service.RunRest()

}

б) Листинг пакета service

package service

import (

    "../db"

    "../models"

    "bytes"

    "encoding/json"

    "fmt"

    "github.com/gorilla/handlers"

    "github.com/gorilla/mux"

    log "github.com/sirupsen/logrus"

    "io/ioutil"

    "net/http"

    "os"

    "strconv"

    "time"

)

var Conf = &Config{}

func RunRest() {

    r := mux.NewRouter()

    r.HandleFunc("/api/diploms", getDiploms).Methods("GET")

    r.HandleFunc("/api/diploms/{id:[0-9]+}", getDiplom).Methods("GET")

    r.HandleFunc("/api/diploms/{id:[0-9]+}", deleteDiplom).Methods("DELETE")

    r.HandleFunc("/api/chairmans", getChairmans).Methods("GET")

    r.HandleFunc("/api/chairmans", createChairmans).Methods("POST")

    r.HandleFunc("/api/chairmans/{id:[0-9]+}", deleteChairmans).Methods("DELETE")

    r.HandleFunc("/api/chairmans", updateChairmans).Methods("PUT")

    r.HandleFunc("/api/commissions", getCommissions).Methods("GET")

    r.HandleFunc("/api/commissions", createCommissions).Methods("POST")

    r.HandleFunc("/api/commissions/{id:[0-9]+}", deleteCommissions).Methods("DELETE")

    r.HandleFunc("/api/commissions", updateCommissions).Methods("PUT")

    r.HandleFunc("/api/diplomorders", getDiplomorders).Methods("GET")

    r.HandleFunc("/api/diplomorders", createDiplomorders).Methods("POST")

    r.HandleFunc("/api/diplomorders/{id:[0-9]+}", deleteDiplomorders).Methods("DELETE")

    r.HandleFunc("/api/diplomorders", updateDiplomorders).Methods("PUT")

    r.HandleFunc("/api/normcontrollers", getNormcontrollers).Methods("GET")

    r.HandleFunc("/api/normcontrollers", createNormcontrollers).Methods("POST")

    r.HandleFunc("/api/normcontrollers/{id:[0-9]+}", deleteNormcontrollers).Methods("DELETE")

    r.HandleFunc("/api/normcontrollers", updateNormcontrollers).Methods("PUT")

    r.HandleFunc("/api/pms", getPms).Methods("GET")

    r.HandleFunc("/api/pms", createPm).Methods("POST")

    r.HandleFunc("/api/pms/{id:[0-9]+}", deletePm).Methods("DELETE")

    r.HandleFunc("/api/pms", updatePm).Methods("PUT")

    r.HandleFunc("/api/reviewers", getReviewers).Methods("GET")

    r.HandleFunc("/api/reviewers", createReviewers).Methods("POST")

    r.HandleFunc("/api/reviewers/{id:[0-9]+}", deleteReviewers).Methods("DELETE")

    r.HandleFunc("/api/reviewers", updateReviewers).Methods("PUT")

    r.HandleFunc("/api/specialtys", getSpecialtys).Methods("GET")

    r.HandleFunc("/api/specialtys", createSpecialtys).Methods("POST")

    r.HandleFunc("/api/specialtys", updateSpecialtys).Methods("PUT")

    r.HandleFunc("/api/specialtys/{id:[0-9]+}", deleteSpecialtys).Methods("DELETE")

    r.HandleFunc("/api/diploms", createDiplom).Methods("POST")

    r.HandleFunc("/api/diploms", updateDiplom).Methods("PUT")

    r.HandleFunc("/api/doc", createDoc).Methods("POST")

    r.PathPrefix("/").Handler(http.FileServer(http.Dir(Conf.StaticPath)))

    log.Printf("starting REST server on %s", Conf.ListenPort)

    allowedHeaders := handlers.AllowedHeaders([]string{"X-Requested-With", "Content-Type", "Authorization"})

    allowedOrigins := handlers.AllowedOrigins([]string{"\*"})

    allowedMethods := handlers.AllowedMethods([]string{"GET", "POST", "PUT", "DELETE", "OPTIONS"})

    err := http.ListenAndServe(

        Conf.ListenPort,

        handlers.CORS(allowedHeaders, allowedOrigins, allowedMethods)(r))

    if err != nil {

        log.Fatal(err)

    }

}

func getDiploms(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    diplom, err := db.GetAllDiploms()

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(diplom)

}

func getDiplom(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    idStr, ok := mux.Vars(r)["id"]

    if !ok || len(idStr) == 0 {

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    id, err := strconv.Atoi(idStr)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    diplom, err := db.GetDiplom(id)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(diplom)

}

func deleteDiplom(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    idStr, ok := mux.Vars(r)["id"]

    if !ok || len(idStr) == 0 {

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    id, err := strconv.Atoi(idStr)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    err = db.DeleteDiplom(id)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(err)

}

func getChairmans(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    chairman, err := db.GetChairman()

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(chairman)

}

func createChairmans(w http.ResponseWriter, r \*http.Request) {

    var err error

    defer func() {

        if err != nil {

            log.Error(err)

            w.WriteHeader(http.StatusInternalServerError)

            return

        }

    }()

    body := json.NewDecoder(r.Body)

    var teacherModel models.Teacher

    err = body.Decode(&teacherModel)

    if err != nil {

        return

    }

    teacher, err := db.InsertChairman(teacherModel)

    if err != nil {

        return

    }

    err = json.NewEncoder(w).Encode(teacher)

    if err != nil {

        return

    }

}

func deleteChairmans(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    idStr, ok := mux.Vars(r)["id"]

    if !ok || len(idStr) == 0 {

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    id, err := strconv.Atoi(idStr)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    err = db.DeleteChairman(id)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(err)

}

func updateChairmans(w http.ResponseWriter, r \*http.Request) {

    var err error

    defer func() {

        if err != nil {

            log.Error(err)

            w.WriteHeader(http.StatusInternalServerError)

            return

        }

    }()

    body := json.NewDecoder(r.Body)

    var chairmanModel models.Teacher

    err = body.Decode(&chairmanModel)

    if err != nil {

        return

    }

    teacher, err := db.UpdateChairman(chairmanModel)

    if err != nil {

        return

    }

    err = json.NewEncoder(w).Encode(teacher)

    if err != nil {

        return

    }

}

func getCommissions(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    commission, err := db.GetCommissionn()

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(commission)

}

func createCommissions(w http.ResponseWriter, r \*http.Request) {

    var err error

    defer func() {

        if err != nil {

            log.Error(err)

            w.WriteHeader(http.StatusInternalServerError)

            return

        }

    }()

    body := json.NewDecoder(r.Body)

    var teacherModel models.Teacher

    err = body.Decode(&teacherModel)

    if err != nil {

        return

    }

    teacher, err := db.InsertCommission(teacherModel)

    if err != nil {

        return

    }

    err = json.NewEncoder(w).Encode(teacher)

    if err != nil {

        return

    }

}

func deleteCommissions(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    idStr, ok := mux.Vars(r)["id"]

    if !ok || len(idStr) == 0 {

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    id, err := strconv.Atoi(idStr)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    err = db.DeleteCommissionn(id)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(err)

}

func updateCommissions(w http.ResponseWriter, r \*http.Request) {

    var err error

    defer func() {

        if err != nil {

            log.Error(err)

            w.WriteHeader(http.StatusInternalServerError)

            return

        }

    }()

    body := json.NewDecoder(r.Body)

    var teacherModel models.Teacher

    err = body.Decode(&teacherModel)

    if err != nil {

        return

    }

    teacher, err := db.UpdateCommissionn(teacherModel)

    if err != nil {

        return

    }

    err = json.NewEncoder(w).Encode(teacher)

    if err != nil {

        return

    }

}

func getDiplomorders(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    diplomorders, err := db.GetDiplomOrder()

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(diplomorders)

}

func createDiplomorders(w http.ResponseWriter, r \*http.Request) {

    var err error

    defer func() {

        if err != nil {

            log.Error(err)

            w.WriteHeader(http.StatusInternalServerError)

            return

        }

    }()

    body := json.NewDecoder(r.Body)

    var diplomOrderModel models.DiplomOrder

    err = body.Decode(&diplomOrderModel)

    if err != nil {

        return

    }

    diplomOrder, err := db.InsertDiplomOrder(diplomOrderModel)

    if err != nil {

        return

    }

    err = json.NewEncoder(w).Encode(diplomOrder)

    if err != nil {

        return

    }

}

func deleteDiplomorders(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    idStr, ok := mux.Vars(r)["id"]

    if !ok || len(idStr) == 0 {

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    id, err := strconv.Atoi(idStr)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    err = db.DeleteDiplomOrder(id)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(err)

}

func updateDiplomorders(w http.ResponseWriter, r \*http.Request) {

    var err error

    defer func() {

        if err != nil {

            log.Error(err)

            w.WriteHeader(http.StatusInternalServerError)

            return

        }

    }()

    body := json.NewDecoder(r.Body)

    var diplomOrderModel models.DiplomOrder

    err = body.Decode(&diplomOrderModel)

    if err != nil {

        return

    }

    diplomOrder, err := db.UpdateDiplomOrder(diplomOrderModel)

    if err != nil {

        return

    }

    err = json.NewEncoder(w).Encode(diplomOrder)

    if err != nil {

        return

    }

}

func getNormcontrollers(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    normcontrollers, err := db.GetNormcontroller()

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(normcontrollers)

}

func createNormcontrollers(w http.ResponseWriter, r \*http.Request) {

    var err error

    defer func() {

        if err != nil {

            log.Error(err)

            w.WriteHeader(http.StatusInternalServerError)

            return

        }

    }()

    body := json.NewDecoder(r.Body)

    var teacherModel models.Teacher

    err = body.Decode(&teacherModel)

    if err != nil {

        return

    }

    teacher, err := db.InsertNormcontroller(teacherModel)

    if err != nil {

        return

    }

    err = json.NewEncoder(w).Encode(teacher)

    if err != nil {

        return

    }

}

func deleteNormcontrollers(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    idStr, ok := mux.Vars(r)["id"]

    if !ok || len(idStr) == 0 {

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    id, err := strconv.Atoi(idStr)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusBadRequest)

        return

    }

    err = db.DeleteNormcontroller(id)

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(err)

}

func updateNormcontrollers(w http.ResponseWriter, r \*http.Request) {

    var err error

    defer func() {

        if err != nil {

            log.Error(err)

            w.WriteHeader(http.StatusInternalServerError)

            return

        }

    }()

    body := json.NewDecoder(r.Body)

    var teacherModel models.Teacher

    err = body.Decode(&teacherModel)

    if err != nil {

        return

    }

    teacher, err := db.UpdateNormcontroller(teacherModel)

    if err != nil {

        return

    }

    err = json.NewEncoder(w).Encode(teacher)

    if err != nil {

        return

    }

}

func getPms(w http.ResponseWriter, r \*http.Request) {

    w.Header().Set("Content-Type", "application/json")

    pms, err := db.GetPm()

    if err != nil {

        log.Error(err)

        w.WriteHeader(http.StatusInternalServerError)

        return

    }

    json.NewEncoder(w).Encode(pms)

}

в) Листинг пакета db

package db

import (

    "../models"

    "database/sql"

    "fmt"

    \_ "github.com/lib/pq"

    "sort"

)

func createConnectin() \*sql.DB {

    connStr := "user=postgres password=123123 dbname=postgres sslmode=disable"

    db, err := sql.Open("postgres", connStr)

    if err != nil {

        panic(err)

    }

    return db

}

func InsertDiplom(diplom models.Diplom) (models.Diplom, error) {

    db := createConnectin()

    defer db.Close()

    err := db.QueryRow("insert into diplom (fio, topic, completion, score, deadline, queuenumber, pmid, normcontrollerid, reviewerid, chairmanid, diplomorderid, specialtyid, commissionid, execution, type, commissioncomment) values ($1, $2, $3, $4, $5, $6, $7, $8, $9, $10, $11, $12, $13, $14, $15, $16) returning id",

        diplom.Fio, diplom.Topic, diplom.Completion, diplom.Score, diplom.Deadline, diplom.Queuenumber, diplom.PmId, diplom.NormcontrollerId, diplom.ReviewerId, diplom.ChairmanId, diplom.DiplomorderId, diplom.SpecialtyId, diplom.CommissionId, diplom.Execution, diplom.Type, diplom.CommissionComment).Scan(&diplom.Id)

    if err != nil{

        panic(err)

    }

    return diplom, err

}

func DeleteDiplom(id int) error {

    db := createConnectin()

    defer db.Close()

    \_, err := db.Exec("delete from diplom values where id = $1", id)

    if err != nil{

        panic(err)

    }

    return err

}

func UpdateDiplom(diplom models.Diplom) (models.Diplom, error) {

    db := createConnectin()

    defer db.Close()

    if \_, err := db.Exec("update diplom set Fio = $2, Topic = $3, Completion = $4, Score = $5, Deadline = $6, Queuenumber = $7, PmId = $8, NormcontrollerId = $9, ReviewerId = $10, ChairmanId = $11, DiplomorderId = $12, SpecialtyId = $13, CommissionId = $14, execution = $15, type = $16, commissioncomment = $17, time = $18 where id = $1",

        diplom.Id, diplom.Fio,

        diplom.Topic, diplom.Completion,

        diplom.Score, diplom.Deadline,

        diplom.Queuenumber, diplom.PmId,

        diplom.NormcontrollerId, diplom.ReviewerId,

        diplom.ChairmanId, diplom.DiplomorderId,

        diplom.SpecialtyId, diplom.CommissionId,

        diplom.Execution, diplom.Type, diplom.CommissionComment, diplom.Time);

    err != nil {

        return diplom, err

    }

    return diplom, nil

}

func GetAllDiploms() ([]models.Diplom, error){

    db := createConnectin()

    defer db.Close()

    rows, err := db.Query("select \* from diplom order by deadline, queuenumber")

    if err != nil {

        panic(err)

    }

    defer rows.Close()

    diplomAll := []models.Diplom{}

    for rows.Next(){

        p := models.Diplom{}

        err := rows.Scan(&p.Id, &p.Fio, &p.Topic, &p.Completion, &p.Score, &p.Queuenumber, &p.Deadline, &p.PmId, &p.NormcontrollerId, &p.ReviewerId, &p.ChairmanId, &p.DiplomorderId, &p.SpecialtyId, &p.CommissionId, &p.Execution, &p.Type, &p.CommissionComment, &p.Time)

        if err != nil{

            fmt.Println(err)

            continue

        }

        diplomAll = append(diplomAll, p)

    }

    return diplomAll, err

}

func GetDiplom(id int) (models.Diplom, error){

    db := createConnectin()

    defer db.Close()

    p := models.Diplom{}

    err := db.QueryRow("select \* from diplom where id = $1 limit 1", id).Scan(&p.Id, &p.Fio, &p.Topic, &p.Completion, &p.Score, &p.Queuenumber, &p.Deadline, &p.PmId, &p.NormcontrollerId, &p.ReviewerId, &p.ChairmanId, &p.DiplomorderId, &p.SpecialtyId, &p.CommissionId, &p.Execution, &p.Type, &p.CommissionComment, &p.Time)

    return p, err

}

func InsertChairman(chairman models.Teacher) (models.Teacher, error) {

    db := createConnectin()

    defer db.Close()

    err := db.QueryRow("insert into chairman (fio) values ($1) returning id",

        chairman.Fio).Scan(&chairman.Id)

    if err != nil{

        panic(err)

    }

    return chairman, err

}

func GetChairman() ([]models.Teacher, error){

    db := createConnectin()

    defer db.Close()

    rows, err := db.Query("select \* from chairman")

    if err != nil {

        panic(err)

    }

    defer rows.Close()

    chairmanAll := []models.Teacher{}

    for rows.Next(){

        p := models.Teacher{}

        err := rows.Scan(&p.Id, &p.Fio)

        if err != nil{

            fmt.Println(err)

            continue

        }

        chairmanAll = append(chairmanAll, p)

    }

    sort.Slice(chairmanAll, func(i, j int) bool {

        return chairmanAll[i].Id < chairmanAll[j].Id

    })

    return chairmanAll, err

}

func DeleteChairman(id int) error {

    db := createConnectin()

    defer db.Close()

    \_, err := db.Exec("delete from chairman values where id = $1", id)

    if err != nil{

        panic(err)

    }

    return err

}

func UpdateChairman(chairman models.Teacher) (models.Teacher, error) {

    db := createConnectin()

    defer db.Close()

    if \_, err := db.Exec("update chairman set Fio = $2 where id = $1",

        chairman.Id, chairman.Fio);

        err != nil {

        return chairman, err

    }

    return chairman, nil

}

func InsertCommission(commission models.Teacher) (models.Teacher, error) {

    db := createConnectin()

    defer db.Close()

    err := db.QueryRow("insert into commission (fio) values ($1) returning id",

        commission.Fio).Scan(&commission.Id)

    if err != nil{

        panic(err)

    }

    return commission, err

}

func GetCommissionn() ([]models.Teacher, error){

    db := createConnectin()

    defer db.Close()

    rows, err := db.Query("select \* from commission")

    if err != nil {

        panic(err)

    }

    defer rows.Close()

    commissionAll := []models.Teacher{}

    for rows.Next(){

        p := models.Teacher{}

        err := rows.Scan(&p.Id, &p.Fio)

        if err != nil{

            fmt.Println(err)

            continue

        }

        commissionAll = append(commissionAll, p)

    }

    sort.Slice(commissionAll, func(i, j int) bool {

        return commissionAll[i].Id < commissionAll[j].Id

    })

    return commissionAll, err

}

# ПРИЛОЖЕНИЕ Д

Листинги кода клиентской части

import { Component } from '@angular/core';

import {Router} from '@angular/router';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.scss']

})

export class AppComponent {

  tab = 'diplom';

  constructor(

    private router: Router,

  ) {}

  changeTab($event) {

    this.tab = $event.target.getAttribute('rel');

    this.router.navigate([this.tab]);

  }

}

import { Component, OnInit } from '@angular/core';

import {TeacherModel} from '../../common/models/teacher.model';

import {FormBuilder, FormGroup, Validators} from '@angular/forms';

import {DiplomService} from '../../common/services/diplom.service';

import {ToastrService} from 'ngx-toastr';

import {ConfirmDialogComponent} from '../dialogs/confirm-dialog/confirm-dialog.component';

import {MatDialog} from '@angular/material/dialog';

import {DiplomDataService} from "../../common/services/diplom-data.service";

@Component({

  selector: 'app-diplom-chairman',

  templateUrl: './diplom-chairman.component.html',

  styleUrls: ['./diplom-chairman.component.scss']

})

export class DiplomChairmanComponent implements OnInit {

  chairmanList: TeacherModel[];

  chairmanForm: FormGroup;

  buttonTitleAdd = true;

  tab = 'add';

  constructor(

    private diplomService: DiplomService,

    private fb: FormBuilder,

    private toastr: ToastrService,

    public dialog: MatDialog,

    private diplomDataService: DiplomDataService

  ) {}

  ngOnInit(): void {

    this.getPmList();

    this.formInit();

  }

  private getPmList() {

    this.diplomService.getChairmans().subscribe(res => {

      this.chairmanList = res;

    });

  }

  private formInit() {

    this.chairmanForm = this.fb.group({

      Id: [0],

      Fio: ['', Validators.required],

    });

  }

  onSubmit() {

    const сhairman: TeacherModel = this.chairmanForm.getRawValue();

    if (this.buttonTitleAdd) {

      this.diplomService.createChairman(сhairman).subscribe(res => {

        this.toastr.success('Добавлен');

        this.getPmList();

        this.diplomDataService.changeChairman = true;

      });

    } else {

      this.diplomService.updateeChairman(сhairman).subscribe(res => {

        this.toastr.success('Обновлен');

        this.getPmList();

        this.diplomDataService.changeChairman = true;

      });

    }

    this.formInit();

    this.buttonTitleAdd = true;

    this.tab = 'add';

  }

  deleteChairman(id: number) {

    const dialogRef = this.dialog.open(ConfirmDialogComponent, {

      width: '250px',

    });

    dialogRef.afterClosed().subscribe(result => {

      if (result) {

        this.diplomService.deleteChairman(id).subscribe(res => {

          this.toastr.success('Удален');

          this.getPmList();

          this.diplomDataService.changeChairman = true;

        });

      }

      this.formInit();

      this.buttonTitleAdd = true;

      this.tab = 'add';

    });

  }

  changeChairman(id: number) {

    const item = this.chairmanList.find(el => el.Id === id);

    this.chairmanForm.patchValue({

      ...item

    });

    this.buttonTitleAdd = false;

    this.tab = 'change';

  }

  changeTab(ev) {

    this.formInit();

    this.buttonTitleAdd = true;

    this.tab = 'add';

  }

}

import {Component, DoCheck, OnInit, ViewChild} from '@angular/core';

import {DiplomService} from '../../common/services/diplom.service';

import {DiplomModel} from '../../common/models/diplom.model';

import {Observable, zip} from 'rxjs';

import {DiplomDataService} from '../../common/services/diplom-data.service';

import {DiplomInfoModel} from '../../common/models/diplomInfo.model';

import {SpecialytyModel} from "../../common/models/specialyty.model";

@Component({

  selector: 'app-diplom',

  templateUrl: './diplom.component.html',

  styleUrls: ['./diplom.component.scss']

})

export class DiplomComponent implements OnInit, DoCheck {

  isDiplomSelect = false;

  selectDiplomId: number;

  diplomsList: DiplomModel[];

  infoAboutDiplom: DiplomInfoModel;

  load = true;

  constructor(

    private diplomService: DiplomService,

    private diplomDataService: DiplomDataService,

  ) {}

  ngOnInit() {

    const filter = localStorage.getItem('filter');

    if (filter) {

      this.getDiplomsListAndFilter();

    } else {

      this.getDiplomsList();

    }

    zip(

      this.getPmList(),

      this.getNormcontrollerList(),

      this.getReviewerList(),

      this.getChairmanList(),

      this.getDiplomorderList(),

      this.getSpecialtyList(),

      this.getCommissionList(),

    ).subscribe(([pm, normcontroller, reviewer, chairman, diplomorder, specialty, commission]) => {

      this.infoAboutDiplom = {

        pmList: pm,

        normcontrollerList: normcontroller,

        reviewerList: reviewer,

        chairmanList: chairman,

        diplomorderList: diplomorder,

        specialtyList: specialty,

        commissionList: commission,

      };

      this.load = false;

    });

  }

  ngDoCheck() {

    this.selectDiplomId = this.diplomDataService.selectDiplomId;

    this.isDiplomSelect = this.diplomDataService.isDiplomSelect;

    if (this.diplomDataService.isDiplomsUpdate) {

      this.getDiplomsList();

    }

    if (this.diplomDataService.diplomsFilter) {

      this.getDiplomsListAndFilter();

    }

    if (this.diplomDataService.changePm) {

      this.getPmList().subscribe(res => {

        this.infoAboutDiplom.pmList = res;

        this.diplomDataService.changePm = false;

      });

    }

    if (this.diplomDataService.changeReviewer) {

      this.getReviewerList().subscribe(res => {

        this.infoAboutDiplom.reviewerList = res;

        this.diplomDataService.changeReviewer = false;

      });

    }

    if (this.diplomDataService.changeChairman) {

      this.getChairmanList().subscribe(res => {

        this.infoAboutDiplom.chairmanList = res;

        this.diplomDataService.changeChairman = false;

      });

    }

    if (this.diplomDataService.changeNormoconntroller) {

      this.getNormcontrollerList().subscribe(res => {

        this.infoAboutDiplom.normcontrollerList = res;

        this.diplomDataService.changeNormoconntroller = false;

      });

    }

    if (this.diplomDataService.changeOrder) {

      this.getDiplomorderList().subscribe(res => {

        this.infoAboutDiplom.diplomorderList = res;

        this.diplomDataService.changeOrder = false;

      });

    }

    if (this.diplomDataService.changeCommission) {

      this.getCommissionList().subscribe(res => {

        this.infoAboutDiplom.commissionList = res;

        this.diplomDataService.changeCommission = false;

      });

    }

    if (this.diplomDataService.changeSpecialty) {

      this.getSpecialtyList().subscribe(res => {

        this.infoAboutDiplom.specialtyList = res;

        this.diplomDataService.changeSpecialty = false;

      });

    }

    this.diplomDataService.diploms = this.diplomsList;

  }

  private getDiplomsList() {

    this.diplomService.getAllDiploms().subscribe(res => {

      this.diplomsList = res;

      this.diplomDataService.isDiplomsUpdate = false;

    });

  }

  private getDiplomsListAndFilter() {

    this.diplomService.getAllDiploms().subscribe(res => {

      const filter = JSON.parse(localStorage.getItem('filter'));

      Object.keys(filter).forEach(key => {

        if (+/\d+/.exec(filter[key]) && key !== 'Execution' && key !== 'Deadline') {

          filter[key] = +filter[key];

        }

        res = res.filter(el => {

          if (key === 'Execution' || key === 'Deadline') {

            if (this.checkDate(filter[key], el[key])) {

              return el;

            }

          }

          return el[key] === filter[key];

        });

      });

      this.diplomsList = res;

      this.diplomDataService.diplomsFilter = false;

    });

  }

  private checkDate(filter, el) {

    const day = (+filter.substring(8, 10) + 1) < 10 ? `0${+filter.substring(8, 10) + 1}` : (+filter.substring(8, 10) + 1).toString();

    return filter.substring(0, 4) === el.substring(0, 4) && filter.substring(5, 7) === el.substring(5, 7) && day === el.substring(8, 10);

  }

  private getPmList() {

    return this.diplomService.getPms();

  }

  private getNormcontrollerList() {

    return this.diplomService.getNormcontrollers();

  }

  private getReviewerList() {

    return this.diplomService.getReviewers();

  }

  private getChairmanList() {

    return this.diplomService.getChairmans();

  }

  private getDiplomorderList() {

    return this.diplomService.getDiplomorders();

  }

  private getSpecialtyList(): Observable<SpecialytyModel[]> {

    return this.diplomService.getSpecialtys();

  }

  private getCommissionList() {

    return this.diplomService.getCommissions();

  }

}

import { Component, OnInit } from '@angular/core';

import {TeacherModel} from '../../common/models/teacher.model';

import {FormBuilder, FormControl, FormGroup, Validators} from '@angular/forms';

import {DiplomService} from '../../common/services/diplom.service';

import {ToastrService} from 'ngx-toastr';

import {DiplomorderModel} from '../../common/models/diplomorder.model';

import \* as \_moment from 'moment';

import {MomentDateAdapter, MAT\_MOMENT\_DATE\_ADAPTER\_OPTIONS} from '@angular/material-moment-adapter';

import {DateAdapter, MAT\_DATE\_FORMATS, MAT\_DATE\_LOCALE} from '@angular/material/core';

import {MatDialog} from "@angular/material/dialog";

import {ConfirmDialogComponent} from "../dialogs/confirm-dialog/confirm-dialog.component";

import {DiplomDataService} from "../../common/services/diplom-data.service";

const moment = \_moment;

export const MY\_FORMATS = {

  parse: {

    dateInput: 'D.MM.YYYY'

  },

  display: {

    dateInput: 'DD.MM.YYYY',

    monthYearLabel: 'MMMM Y',

    dateA11yLabel: 'LL',

    monthYearA11yLabel: 'MMMM Y'

  }

};

@Component({

  selector: 'app-diplom-order',

  templateUrl: './diplom-order.component.html',

  styleUrls: ['./diplom-order.component.scss'],

  providers: [{

    provide: DateAdapter,

    useClass: MomentDateAdapter,

    deps: [MAT\_DATE\_LOCALE, MAT\_MOMENT\_DATE\_ADAPTER\_OPTIONS]

  },

    {provide: MAT\_DATE\_FORMATS, useValue: MY\_FORMATS},

  ],

})

export class DiplomOrderComponent implements OnInit {

  diplomOrderList: DiplomorderModel[];

  diplomOrderForm: FormGroup;

  buttonTitleAdd = true;

  tab = 'add';

  constructor(

    private diplomService: DiplomService,

    private fb: FormBuilder,

    private toastr: ToastrService,

    private dialog: MatDialog,

    private diplomDataService: DiplomDataService

  ) {

  }

  ngOnInit(): void {

    this.getReviewerList();

    this.formInit();

  }

  private getReviewerList() {

    this.diplomService.getDiplomorders().subscribe(res => {

      this.diplomOrderList = res;

    });

  }

  private formInit() {

    this.diplomOrderForm = this.fb.group({

      Id: [0],

      Name: ['', Validators.required],

      Dateorder: [new FormControl(moment()), Validators.required],

    });

  }

  onSubmit() {

    const diplomOrder: DiplomorderModel = this.diplomOrderForm.getRawValue();

    if (diplomOrder.Dateorder === '') {

      diplomOrder.Dateorder = null;

    } else {

      diplomOrder.Dateorder = moment(diplomOrder.Dateorder).add(4, 'hours').toISOString();

    }

    if (this.buttonTitleAdd) {

      this.diplomService.createDiplomorder(diplomOrder).subscribe(res => {

        this.toastr.success('Добавлен');

        this.getReviewerList();

        this.diplomDataService.changeOrder = true;

      });

    } else {

      this.diplomService.updateeDiplomorder(diplomOrder).subscribe(res => {

        this.toastr.success('Обновлен');

        this.getReviewerList();

        this.diplomDataService.changeOrder = true;

      });

    }

    this.formInit();

    this.buttonTitleAdd = true;

    this.tab = 'add';

  }

  deleteDiplomOrder(id: number) {

    const dialogRef = this.dialog.open(ConfirmDialogComponent, {

      width: '250px',

    });

    dialogRef.afterClosed().subscribe(result => {

      if (result) {

        this.diplomService.deleteDiplomorder(id).subscribe(res => {

          this.toastr.success('Удален');

          this.getReviewerList();

          this.diplomDataService.changeOrder = true;

        }, error => {

          this.toastr.error('Нарушение ограничений целостности');

        });

      }

      this.formInit();

      this.buttonTitleAdd = true;

      this.tab = 'add';

    });

  }

  changeDiplomOrder(id: number) {

    const item = this.diplomOrderList.find(el => el.Id === id);

    this.diplomOrderForm.patchValue({

      ...item

    });

    this.buttonTitleAdd = false;

    this.tab = 'change';

  }

  changeTab(ev) {

    this.formInit();

    this.buttonTitleAdd = true;

    this.tab = 'add';

  }

  strToDate(str: string) {

    const date = new Date(str);

    const day = date.getDate();

    const month = date.getMonth() + 1;

    const year = date.getFullYear();

    return `${day < 10 ? `0${day}` : day}.${month < 10 ? `0${month}` : month}.${year}`;

  }

}

import {Component, OnInit} from '@angular/core';

import {DiplomService} from '../../common/services/diplom.service';

import {TeacherModel} from '../../common/models/teacher.model';

import {FormBuilder, FormGroup, Validators} from '@angular/forms';

import {ToastrService} from 'ngx-toastr';

import {DiplomModel} from '../../common/models/diplom.model';

import {MatDialog} from "@angular/material/dialog";

import {ConfirmDialogComponent} from "../dialogs/confirm-dialog/confirm-dialog.component";

import {DiplomDataService} from "../../common/services/diplom-data.service";

@Component({

  selector: 'app-pm',

  templateUrl: './pm.component.html',

  styleUrls: ['./pm.component.scss']

})

export class PmComponent implements OnInit {

  pmList: TeacherModel[];

  pmForm: FormGroup;

  buttonTitleAdd = true;

  tab = 'add';

  constructor(

    private diplomService: DiplomService,

    private fb: FormBuilder,

    private toastr: ToastrService,

    private dialog: MatDialog,

    private diplomDataService: DiplomDataService

  ) {}

  ngOnInit(): void {

    this.getPmList();

    this.formInit();

  }

  private getPmList() {

    this.diplomService.getPms().subscribe(res => {

      this.pmList = res;

    });

  }

  private formInit() {

    this.pmForm = this.fb.group({

      Id: [0],

      Fio: ['', Validators.required],

    });

  }

  onSubmit() {

    const pm: TeacherModel = this.pmForm.getRawValue();

    if (this.buttonTitleAdd) {

      this.diplomService.createPm(pm).subscribe(res => {

        this.toastr.success('Добавлен');

        this.getPmList();

        this.diplomDataService.changePm = true;

      });

    } else {

      this.diplomService.updateePm(pm).subscribe(res => {

        this.toastr.success('Обновлен');

        this.getPmList();

        this.diplomDataService.changePm = true;

      });

    }

    this.formInit();

    this.buttonTitleAdd = true;

  }

  deletePm(id: number) {

    const dialogRef = this.dialog.open(ConfirmDialogComponent, {

      width: '250px',

    });

    dialogRef.afterClosed().subscribe(result => {

      if (result) {

        this.diplomService.deletePm(id).subscribe(res => {

          this.toastr.success('Удален');

          this.getPmList();

          this.diplomDataService.changePm = true;

        }, error => {

          this.toastr.error('Нарушение ограничений целостности');

        });

      }

      this.formInit();

      this.buttonTitleAdd = true;

      this.tab = 'add';

    });

  }

  changePm(id: number) {

    const item = this.pmList.find(el => el.Id === id);

    this.pmForm.patchValue({

      ...item

    });

    this.buttonTitleAdd = false;

    this.tab = 'change';

  }

  changeTab(ev) {

    this.formInit();

    this.buttonTitleAdd = true;

    this.tab = 'add';

  }

}

import {Component, Input, OnInit, DoCheck} from '@angular/core';

import {DiplomService} from '../../../common/services/diplom.service';

import {DiplomModel} from '../../../common/models/diplom.model';

import {DiplomDataService} from '../../../common/services/diplom-data.service';

import {DiplomInfoModel} from '../../../common/models/diplomInfo.model';

import {Router} from '@angular/router';

import {PasswordDialogComponent} from '../../dialogs/password-dialog/password-dialog.component';

import {ConfirmDialogComponent} from '../../dialogs/confirm-dialog/confirm-dialog.component';

import {MatDialog} from '@angular/material/dialog';

import {ToastrService} from "ngx-toastr";

@Component({

  selector: 'app-diplom-list',

  templateUrl: './diplom-list.component.html',

  styleUrls: ['./diplom-list.component.scss']

})

export class DiplomListComponent implements OnInit, DoCheck {

  @Input() diplomsList: DiplomModel[];

  @Input() infoAboutDiplom: DiplomInfoModel;

  isAdmin: boolean;

  constructor(

    private diplomDataService: DiplomDataService,

    private diplomService: DiplomService,

    private router: Router,

    private toastr: ToastrService,

    public dialog: MatDialog

  ) {

  }

  ngOnInit(): void {

    const filterItem = JSON.parse(localStorage.getItem('filter'));

    if (filterItem) {

      Object.keys(filterItem).forEach(key => {

        const keyObj = `${key}Check`;

        filterItem[keyObj] = true;

      });

    }

  }

  ngDoCheck() {

    this.isAdmin = this.diplomDataService.isAdmin;

  }

  changeDiplom(e, id: number) {

    e.preventDefault();

    e.stopPropagation();

    this.diplomDataService.selectDiplomId = id;

    this.diplomDataService.isDiplomSelect = true;

  }

  createTitleString(diplom: DiplomModel) {

    const diplomOrder = this.infoAboutDiplom.diplomorderList.find((el) => el.Id === diplom.DiplomorderId);

    const specialty = this.infoAboutDiplom.specialtyList.find((el) => el.Id === diplom.SpecialtyId);

    const pm = this.infoAboutDiplom.pmList.find((el) => el.Id === diplom.PmId);

    return `[${diplomOrder.Name}, ${this.strToDate(diplomOrder.Dateorder)}]-${diplom.Queuenumber}-${specialty.Name}-${diplom.Fio}/${pm.Fio}`;

  }

  getNormcontroller(normcontrollerId: number) {

    const normcontroller = this.infoAboutDiplom.normcontrollerList.find((el) => el.Id === normcontrollerId);

    return normcontroller.Fio;

  }

  getReviewer(reviewerId: number) {

    const reviewer = this.infoAboutDiplom.reviewerList.find((el) => el.Id === reviewerId);

    return reviewer.Fio;

  }

  getCommission(commissionId: number) {

    const commission = this.infoAboutDiplom.commissionList.find((el) => el.Id === commissionId);

    return commission.Fio;

  }

  getChairman(chairmanId: number) {

    const chairman = this.infoAboutDiplom.chairmanList.find((el) => el.Id === chairmanId);

    return chairman.Fio;

  }

  deleteDiplom(id: number) {

    const dialogRef = this.dialog.open(ConfirmDialogComponent, {

      width: '250px',

    });

    dialogRef.afterClosed().subscribe(result => {

      if (result) {

        this.diplomService.deleteDiplom(id).subscribe(res => {

          this.toastr.success('Удален');

          this.diplomDataService.isDiplomsUpdate = true;

        }, error => {

          this.toastr.error('Нарушение ограничений целостности');

        });

      }

    });

  }

  openDiplom(id: number) {

    this.router.navigate(['/diplom-components-info/' + id]);

  }

  strToDate(str: string) {

    const date = new Date(str);

    const day = date.getDate();

    const month = date.getMonth() + 1;

    const year = date.getFullYear();

    return `${day < 10 ? `0${day}` : day}.${month < 10 ? `0${month}` : month}.${year}`;

  }

}

import {Component, OnDestroy, OnInit} from '@angular/core';

import {ActivatedRoute, Router} from '@angular/router';

import {DiplomService} from '../../../common/services/diplom.service';

import {DiplomModel} from '../../../common/models/diplom.model';

import {zip} from 'rxjs';

import { timer } from 'rxjs';

@Component({

  selector: 'app-diplom-id',

  templateUrl: './diplom-id.component.html',

  styleUrls: ['./diplom-id.component.scss']

})

export class DiplomIdComponent implements OnInit, OnDestroy {

  constructor(

    private router: Router,

    private route: ActivatedRoute,

    private diplomService: DiplomService

  ) {

  }

  diplomId: number;

  pm = '';

  normcontroller = '';

  reviewer = '';

  chairman = '';

  diplomOrder = '';

  specialty = '';

  commission = '';

  diplom = new DiplomModel();

  subscribeTimer: any;

  timeLeft = 20;

  timeEnd = 20;

  timerText: string;

  color = '#212529';

  timer = false;

  interval;

  checkInterval;

  ngOnInit(): void {

    this.diplomId = this.route.snapshot.params.id;

    this.getDiplom();

    this.startTimer();

  }

  ngOnDestroy() {

    clearInterval(this.interval);

    clearInterval(this.checkInterval);

  }

  getDiplom() {

    this.diplomService.getDiplomById(this.diplomId).subscribe(res => {

      if (res) {

        this.diplom = res;

        if (this.diplom.Time !== 0) {

          this.timeLeft = this.diplom.Time;

          this.start();

        }

        this.getDiplomInfo(res);

      }

    });

  }

  getDiplomInfo(diplom: DiplomModel) {

    zip(

      this.getPmList(),

      this.getNormcontrollerList(),

      this.getReviewerList(),

      this.getChairmanList(),

      this.getDiplomorderList(),

      this.getSpecialtyList(),

      this.getCommissionList(),

    ).subscribe(([pm, normcontroller, reviewer, chairman, diplomorder, specialty, commission]) => {

      this.pm = pm.find(el => el.Id === diplom.PmId).Fio;

      this.normcontroller = normcontroller.find(el => el.Id === diplom.NormcontrollerId).Fio;

      this.reviewer = reviewer.find(el => el.Id === diplom.ReviewerId).Fio;

      this.chairman = chairman.find(el => el.Id === diplom.ChairmanId).Fio;

      const diplomOrderModel = diplomorder.find(el => el.Id === diplom.DiplomorderId);

      this.diplomOrder = `${diplomOrderModel.Name} ${this.strToDate(diplomOrderModel.Dateorder)}`;

      this.specialty = specialty.find(el => el.Id === diplom.SpecialtyId).Name;

      this.commission = commission.find(el => el.Id === diplom.CommissionId).Fio;

    });

  }

  private getPmList() {

    return this.diplomService.getPms();

  }

  private getNormcontrollerList() {

    return this.diplomService.getNormcontrollers();

  }

  private getReviewerList() {

    return this.diplomService.getReviewers();

  }

  private getChairmanList() {

    return this.diplomService.getChairmans();

  }

  private getDiplomorderList() {

    return this.diplomService.getDiplomorders();

  }

  private getSpecialtyList() {

    return this.diplomService.getSpecialtys();

  }

  private getCommissionList() {

    return this.diplomService.getCommissions();

  }

  private updateDiplom(diplom: DiplomModel) {

    if (this.timer) {

      this.diplomService.updateDiplom(diplom).subscribe(res => {});

    }

  }

  onMain() {

    this.router.navigate(['/']);

  }

  stop() {

    clearInterval(this.interval);

    this.diplom.Time = 0;

    this.updateDiplom(this.diplom);

    this.timerText = '';

    this.color = '#212529';

    this.timer = false;

  }

  start() {

    if (!this.interval) {

      this.interval = setInterval(() => {

        this.diplom.Time = this.timeLeft;

        this.timer = true;

        this.updateDiplom(this.diplom);

        console.log('1')

        if (this.timeLeft > 0) {

          this.timeLeft--;

          this.timerText = this.secToMin(this.timeLeft.toString());

        } else {

          this.timeLeft--;

          this.color = 'red';

          this.timerText = this.secToMin((this.timeLeft \* -1).toString());

        }

      }, 1000);

    }

  }

  startTimer() {

    this.checkInterval = setInterval(() => {

      console.log('2')

      this.diplomService.getDiplomById(this.diplomId).subscribe(res => {

        if (res.Time !== 0) {

          clearInterval(this.checkInterval);

          this.start();

        }

      });

    }, 2000);

  }

  secToMin(sec: string) {

    const min = Math.floor(+sec / 60);

    let seconds = (+sec - min \* 60).toString();

    if (+seconds < 10) {

      seconds = `0${seconds}`;

    }

    return `${min}:${seconds}`;

  }

  strToDate(str: string) {

    const date = new Date(str);

    const day = date.getDate();

    const month = date.getMonth() + 1;

    const year = date.getFullYear();

    return `${day < 10 ? `0${day}` : day}.${month < 10 ? `0${month}` : month}.${year}`;

  }

}

ПРИЛОЖЕНИЕ Е

Таблица экономических показателей