ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**Отчет о прохождении учебной практики**

|  |
| --- |
| Танашкин Валерий Александрович |

*(Ф.И.О. обучающегося)*

|  |
| --- |
| 2 курс, гр. 3530901/80003 |

|  |
| --- |
| 09.03.01 «Информатика и вычислительная техника» |

*(Направление подготовки (код и наименование)*

|  |
| --- |
| **Место прохождения практики**: Высшая школа информационных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ» с использованием электронного обучения и дистанционных образовательных технологий. |

|  |
| --- |
| **Сроки практики:** с 22 июня по 18 июля 2020 г. |

|  |
| --- |
| **Руководитель практики:** |

|  |
| --- |
| Жвариков В. А., к.т.н., доцент ВШИСиСТ ИКНТ |

*(Ф.И.О., уч. степень, должность)*

|  |
| --- |
| **Оценка (зачет): \_\_\_\_\_\_\_\_\_\_\_** |

|  |
| --- |
|  |

|  |
| --- |
| Руководитель практики : Жвариков В. А. |

|  |
| --- |
| Обучающийся: Танашкин. В. А |

|  |
| --- |
| Дата: |
|  |

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**ИНДИВИДУАЛЬНЫЙ ПЛАН (ЗАДАНИЕ И ГРАФИК)**

**ПРОВЕДЕНИЯ ПРАКТИКИ**

|  |
| --- |
| Ф.И.О. обучающегося Танашкин Валерий Александрович |

|  |
| --- |
| **Направление подготовки** (код/наименование): 09.03.01, «Информатика и вычислительная техника» |
| **Профиль** (код/наименование): 09.03.01 |
| **Вид практики:** учебная |
| **Тип практики:** ознакомительная |
| **Место прохождения практики**: Высшая школа информационных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ» с использованием электронного обучения и дистанционных образовательных технологий. |

|  |
| --- |
|  |
| Руководитель практики: Жвариков В. А., к.т.н., доцент ВШИСиСТ ИКНТ |
| *(Ф.И.О., уч.степень, должность)* |

**Рабочий график проведения учебной практики**

Сроки практики: с **22.06.2020** г. по **18.07.2020** г.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Этапы (периоды) практики | Вид работ | Сроки прохождения этапа (периода) практики |
| 1 | Организационный этап | Установочная лекция (вебинар) для разъяснения целей, задач, содержания и порядка прохождения практики, выдача сопроводительных документов по практике  Основная тема: разработка приложения для учета рабочей активности сотрудников цеха на Java/Kotlin | 22.06.2020 г. |
| 2 | Основной  этап | Изучение сторонних библиотек для создания клиент-серверного приложения.  Сбор информации, обработка, систематизация и анализ фактического и теоретического материала.  Содержание практики: разработка клиент-серверного приложения для компьютера с графическим интерфейсом.  Планируемые результаты прохождения практики: улучшение навыков создания клиент-серверных приложений для персональных компьютеров и ноутбуков с разными операционными системами | 23.06.2020 - 15.07.2020 г. |
| 3 | Заключительный этап | Подготовка отчета | 16.07.- 17.07.2020 г. |
| Защита отчета по практике (сдача зачета) | 17.07 - 18.07.2020 г |

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Танашкин В. А./

Руководитель практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /Жвариков В. А.

Содержание

[**Техническое задание**](#_Toc45654964) **4-5**

[**Метод решения 6**](#_Toc45654966)

База данных[:](#_Toc45654967) 6

Backend[: 6](#_Toc45654968)

Frontend[. 7](#_Toc45654969)

[**Скриншоты программы 8**](#_Toc45654970)

[**Список использованных источников 1**](#_Toc45654971)**5**

[**Приложение 1**](#_Toc45654972)**6**

**Техническое задание**

Цель летней практики: разработка backend и frontend частей описанного ниже сайта, снятие хостинга и домена и выведение сайта в работу.

Описание frontend части:

Сайт – база пациентов определенного врача.

Перечисление страниц сайта:

· Home (главная страница)

· Поиск по пациентам

· Профиль врача

· Регистрация

Home page

Главная страница на которой будет размещена информация о сайте. Требования:

· Новости сайта в формате блога.

· Минимальная информация о сайте, его функционале.

· Логотип сайта.

Профиль врача

Страница с информацией о враче.

Требования:

· На самой странице будет указана информация, которую каждый врач посчитает нужным указать о себе.

· Возможность изменения информации.

· Возможность удаления профиля.

Поиск по пациентам

Страница на которой будет список всех пациентов определенного врача

Требования:

· Указывается минимальная информация, по которой можно идентифицировать пациента

· Поиск по ФИО, дате рождения, организации

· Фильтры по возрасту, полу

· Сортировка по ФИО, дате добавления.

Регистрация

Регистрация врача Требования:

· Форма регистрации

· Подтверждение регистрации по Email

Требования к backend части:

Технологии:

· ЯП и библиотеки - Java, Spring, Postgresql, Hibernate, Liquibase

· IDE - IntellijIdea

Цели:

· обеспечение работы всех описанных выше функций frontend части

· обеспечение шифрования “sensitive” данных, таких как пароли

· обеспечение защиты персональных данные пациентов и врачей с использованием JWT и https.

# <https://github.com/NastyPill/patientbase-frontend> - клиент на React.js

# <https://github.com/NastyPill/patientbase> - основная сслыка

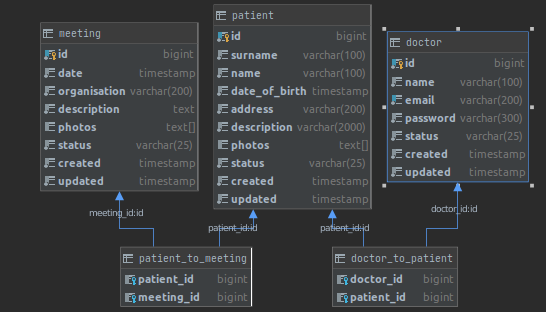
# Метод решения

Приложение можно разделить на две программы:

1. Backend
   1. Services
   2. Controllers
   3. Models
   4. Repositories
2. Frontend
   1. Components

## **База данных****:**

Я использовал PostgreSQL в 3ей нормальной форме, с использованием связывающих таблиц.



Для версионирования БД была использована библиотека liquibase.

## **Backend:**

## Мной был использован Spring Framework: Spring Boot, Spring MVC, Spring Data. Использовал RESTful сервисы, клиент-серверное взаимодействие проходило с помощью HTTP requests/responces. Также для уменьшения кол-ва кода была использована библиотека lombok, которая генерирует setters/getters/constructors/e.t.c с помощью аннтаций. Само взаимодействие происходило синхронно. Взаимодействие сервера и БД было с помощью JPA, HQL и Criteria.

## **Frontend:**

## Был написан на html/css/js(React.js). Были созданы различные компоненты. Связь с backend поддерживалась с помощью ajax и http response/requests.

# Скриншоты программы

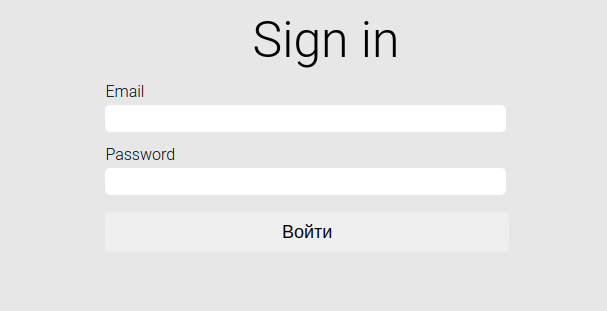


Рис. 1 Окно авторизации

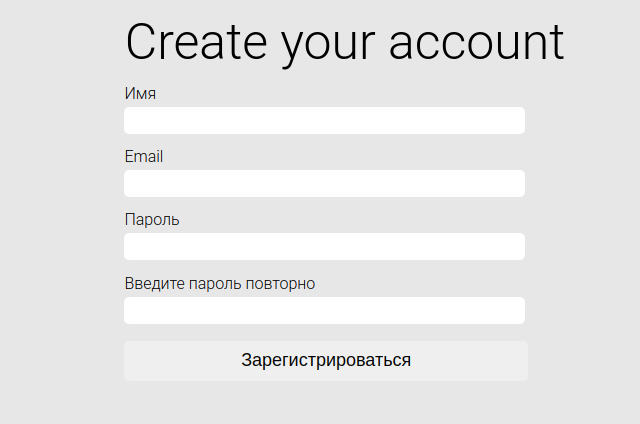


Рис. 2 Окно регистрации

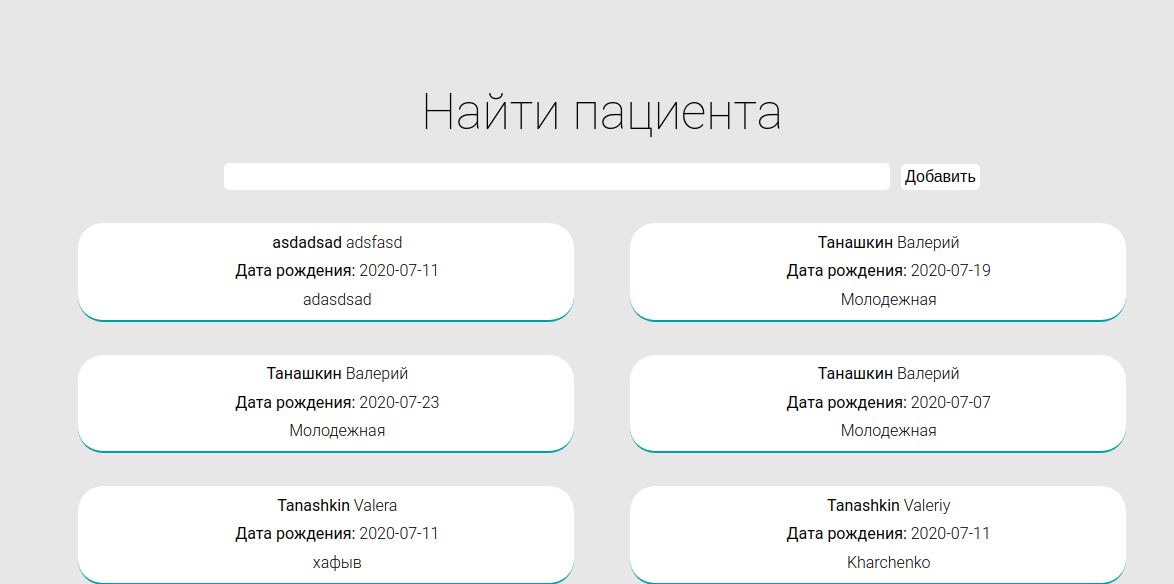


Рис. 3 Список пациентов

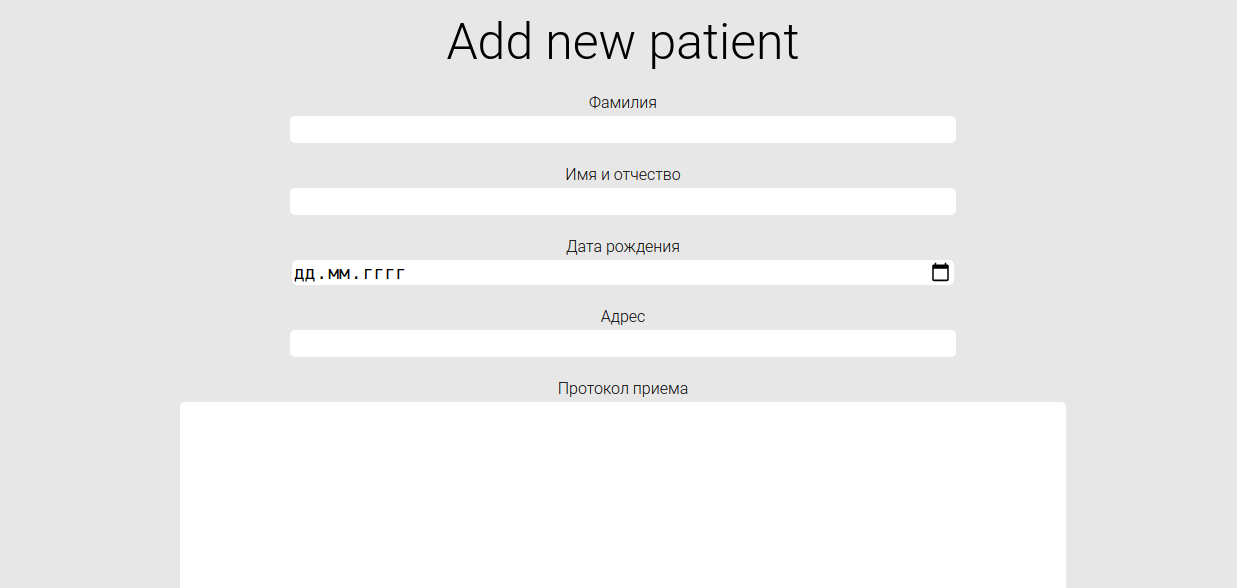


Рис. 4 Добавить пациента

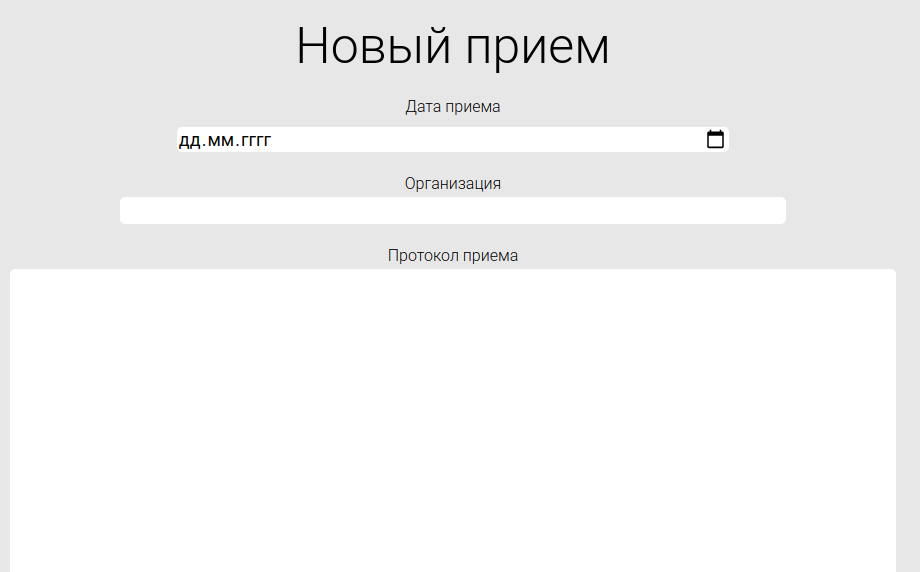


Рис. 5 Добавить прием

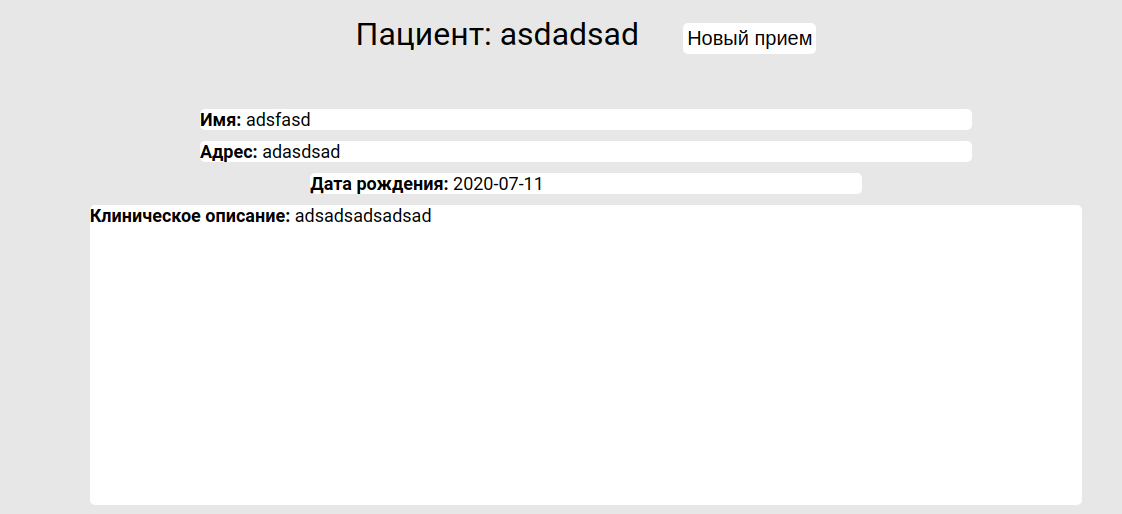


Рис. 6 Информация о пациенте

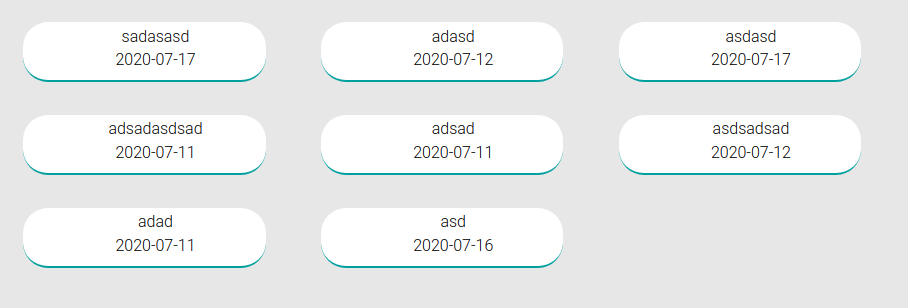


Рис. 7 Список приемов пациента

**Список использованных источников**

1. [https://](https://tornadofx.io/)stackoveflow.com – поиск ответов на вопросы.

2. [https://youtube.com](https://youtube.com/) — гайды по React,.js

# Приложение

# Backend:

package ru.patientbase.mainAPI.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.crypto.factory.PasswordEncoderFactories;

import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

public class JavaConfig {

@Bean

public PasswordEncoder passwordEncoder() {

return PasswordEncoderFactories.*createDelegatingPasswordEncoder*(); //Password encoder bean.

}

}

package ru.patientbase.mainAPI.dto;

import lombok.Data;

import ru.patientbase.mainAPI.entity.Doctor;

import ru.patientbase.mainAPI.entity.Status;

import java.sql.Date;

import java.util.List;

import java.util.Objects;

import java.util.stream.Collectors;

//class for translate object to JSON

@Data

public class DoctorDTO {

private Long id;

private String name;

private String email;

private List<MiniPatientDTO> patientList;

private String status;

private Date created;

private Date updated;

public static DoctorDTO translateToDto(Doctor doctor) {

DoctorDTO doctorDTO = new DoctorDTO();

doctorDTO.setId(doctor.getId());

doctorDTO.setName(doctor.getName());

doctorDTO.setEmail(doctor.getEmail());

doctorDTO.setPatientList(doctor

.getPatientList()

.stream()

.map(MiniPatientDTO::*translate*)

.collect(Collectors.*toList*()));

doctorDTO.setStatus(doctor.getStatus().toString());

doctorDTO.setCreated(doctor.getCreated());

doctorDTO.setUpdated(doctor.getUpdated());

return doctorDTO;

}

public static Doctor getDoctor(DoctorDTO doctorDTO) {

Doctor doctor = new Doctor();

doctor.setId(doctorDTO.getId());

doctor.setName(doctorDTO.getName());

doctor.setEmail(doctorDTO.getEmail());

doctor.setStatus(Status.*valueOf*(doctorDTO.getStatus()));

doctor.setCreated(doctorDTO.getCreated());

doctor.setUpdated(doctorDTO.getUpdated());

return doctor;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

DoctorDTO doctorDTO = (DoctorDTO) o;

return Objects.equals(id, doctorDTO.id) &&

Objects.equals(name, doctorDTO.name) &&

Objects.equals(email, doctorDTO.email);

}

@Override

public int hashCode() {

return Objects.hash(id, name, email, patientList, status, created, updated);

}

}

package ru.patientbase.mainAPI.dto;

import lombok.Data;

//class for translate object to JSON

@Data

public class LoginDto {

private String email;

private String password;

}

package ru.patientbase.mainAPI.dto;

import lombok.Data;

import ru.patientbase.mainAPI.entity.Meeting;

import ru.patientbase.mainAPI.entity.Status;

import java.sql.Date;

import java.util.List;

import java.util.Objects;

//class for translate object to JSON

@Data

public class MeetingDTO {

private Long id;

private Date date;

private String organisation;

private String description;

private List<String> links;

private String status;

private MiniPatientDTO patientDTO;

public static Meeting getMeeting(MeetingDTO meetingDto) {

Meeting meeting = new Meeting();

meeting.setId(meetingDto.getId());

meeting.setDate(meetingDto.getDate());

meeting.setOrganisation(meetingDto.getOrganisation());

meeting.setDescription(meetingDto.getDescription());

meeting.setLinks(meetingDto.getLinks());

meeting.setStatus(Status.*valueOf*(meetingDto.getStatus()));

return meeting;

}

public static MeetingDTO translateToDto(Meeting meeting) {

MeetingDTO meetingDTO = new MeetingDTO();

meetingDTO.setId(meeting.getId());

meetingDTO.setDate(meeting.getDate());

meetingDTO.setOrganisation(meeting.getOrganisation());

meetingDTO.setDescription(meeting.getDescription());

meetingDTO.setLinks(meeting.getLinks());

meetingDTO.setStatus(meeting.getStatus().toString());

meetingDTO.setPatientDTO(MiniPatientDTO.*translate*(meeting.getPatient()));

return meetingDTO;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

MeetingDTO that = (MeetingDTO) o;

return Objects.equals(id, that.id) &&

Objects.equals(date, that.date) &&

Objects.equals(organisation, that.organisation);

}

@Override

public int hashCode() {

return Objects.hash(id, date, organisation);

}

}

package ru.patientbase.mainAPI.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

import ru.patientbase.mainAPI.entity.Meeting;

import java.sql.Date;

//class for translate object to JSON

@Data

@AllArgsConstructor

public class MiniMeetingDto {

private Long id;

private String organisation;

private Date date;

public static MiniMeetingDto translate(Meeting meeting) {

return new MiniMeetingDto(meeting.getId(),

meeting.getOrganisation(),

meeting.getDate());

}

}

package ru.patientbase.mainAPI.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

import ru.patientbase.mainAPI.entity.Patient;

import java.sql.Date;

//class for translate object to JSON

@Data

@AllArgsConstructor

public class MiniPatientDTO {

private Long id;

private String surname;

private String name;

private Date dateOfBirth;

private String address;

public static MiniPatientDTO translate(Patient patient) {

return new MiniPatientDTO(patient.getId(),

patient.getSurname(),

patient.getName(),

patient.getDateOfBirth(),

patient.getAddress());

}

}

package ru.patientbase.mainAPI.dto;

import lombok.Data;

import ru.patientbase.mainAPI.entity.Patient;

import java.sql.Date;

import java.util.List;

import java.util.Objects;

import java.util.Set;

import java.util.stream.Collectors;

//class for translate object to JSON

@Data

public class PatientDTO {

private Long id;

private String surname;

private String name;

private Date dateOfBirth;

private String address;

private String description;

private List<String> links;

private Set<MiniMeetingDto> meetings;

private String status;

public static Patient getPatient(PatientDTO patientDTO) {

Patient patient = new Patient();

patient.setId(patientDTO.getId());

patient.setSurname(patientDTO.getSurname());

patient.setName(patientDTO.getName());

patient.setDateOfBirth(patientDTO.getDateOfBirth());

patient.setAddress(patientDTO.getAddress());

patient.setDescription(patientDTO.getDescription());

patient.setLinks(patientDTO.getLinks());

return patient;

}

public static PatientDTO translateToDto(Patient patient) {

PatientDTO patientDTO = new PatientDTO();

patientDTO.setId(patient.getId());

patientDTO.setSurname(patient.getSurname());

patientDTO.setName(patient.getName());

patientDTO.setDateOfBirth(patient.getDateOfBirth());

patientDTO.setAddress(patient.getAddress());

patientDTO.setDescription(patient.getDescription());

patientDTO.setLinks(patient.getLinks());

patientDTO.setStatus(patient.getStatus().toString());

if (patient.getMeetingList() != null)

patientDTO.setMeetings(patient.getMeetingList().stream()

.map(MiniMeetingDto::*translate*)

.collect(Collectors.*toSet*()));

return patientDTO;

}

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

PatientDTO that = (PatientDTO) o;

return Objects.*equals*(id, that.id) &&

Objects.*equals*(surname, that.surname) &&

Objects.*equals*(name, that.name) &&

Objects.*equals*(dateOfBirth, that.dateOfBirth) &&

Objects.equals(address, that.address);

}

@Override

public int hashCode() {

return Objects.hash(id, surname, name, dateOfBirth, address);

}

}

package ru.patientbase.mainAPI.dto;

import lombok.Data;

import ru.patientbase.mainAPI.entity.Doctor;

//class for translate object to JSON

@Data

public class RegisterDto {

private String name;

private String email;

private String password;

public static Doctor getDoctor(RegisterDto registerDto) {

Doctor doctor = new Doctor();

doctor.setEmail(registerDto.getEmail());

doctor.setName(registerDto.getName());

doctor.setPassword(registerDto.getPassword());

return doctor;

}

}

package ru.patientbase.mainAPI.entity;

import lombok.Data;

import javax.persistence.\*;

import java.sql.Date;

//Entity superclass

@MappedSuperclass

@Data

public class BaseEntity {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*)

private Long id;

@Enumerated(EnumType.*STRING*)

@Column(name = "status")

private Status status;

@Column(name = "created")

private Date created;

@Column(name = "updated")

private Date updated;

}

package ru.patientbase.mainAPI.entity;

import lombok.Data;

import lombok.Getter;

import lombok.Setter;

import javax.persistence.\*;

import java.util.List;

import java.util.Objects;

//Doctor table class

@Entity

@Table(name = "DOCTOR")

@Getter

@Setter

public class Doctor extends BaseEntity {

@Column(name = "name")

private String name;

@Column(name = "email")

private String email;

@Column(name = "password")

private String password;

@OneToMany(mappedBy = "doctor", fetch = FetchType.*LAZY*)

private List<Patient> patientList;

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

if (!super.equals(o)) return false;

Doctor doctor = (Doctor) o;

return email.equals(doctor.email);

}

@Override

public int hashCode() {

return Objects.*hash*(super.hashCode(), email);

}

@Override

public String toString() {

return "Doctor{" +

"name='" + name + '\'' +

", email='" + email + '\'' +

", patientList=" + patientList +

'}';

}

}

package ru.patientbase.mainAPI.entity;

import com.vladmihalcea.hibernate.type.array.ListArrayType;

import lombok.Data;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.Type;

import org.hibernate.annotations.TypeDef;

import javax.persistence.\*;

import java.sql.Date;

import java.util.List;

import java.util.Objects;

//Meeting table class

@Entity

@Getter

@Setter

@Table(name = "MEETING")

@TypeDef(

name = "list-array",

typeClass = ListArrayType.class

)

public class Meeting extends BaseEntity {

@Column(name = "date")

private Date date;

@Column(name = "organisation")

private String organisation;

@Column(name = "description")

private String description;

@Type(type = "list-array")

@Column(

name = "photos",

columnDefinition = "text[]"

)

private List<String> links;

@ManyToOne

@JoinTable(name = "patient\_to\_meeting",

inverseJoinColumns = {@JoinColumn(name = "patient\_id", referencedColumnName = "id")},

joinColumns = {@JoinColumn(name = "meeting\_id", referencedColumnName = "id")}

)

private Patient patient;

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

if (!super.equals(o)) return false;

Meeting meeting = (Meeting) o;

return Objects.*equals*(date, meeting.date) &&

Objects.*equals*(organisation, meeting.organisation) &&

Objects.*equals*(description, meeting.description) &&

Objects.*equals*(patient, meeting.patient);

}

@Override

public int hashCode() {

return Objects.*hash*(super.hashCode(), date, organisation, description, patient);

}

@Override

public String toString() {

return "Meeting{" +

"date=" + date +

", organisation='" + organisation + '\'' +

", description='" + description + '\'' +

", patient=" + patient.getSurname() + " " + patient.getName() +

'}';

}

}

package ru.patientbase.mainAPI.entity;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.vladmihalcea.hibernate.type.array.ListArrayType;

import lombok.Data;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.Type;

import org.hibernate.annotations.TypeDef;

import javax.persistence.\*;

import java.sql.Date;

import java.util.List;

import java.util.Objects;

//Patient table class

@Entity

@Getter

@Setter

@Table(name = "PATIENT")

@TypeDef(

name = "list-array",

typeClass = ListArrayType.class

)

public class Patient extends BaseEntity {

@Column(name = "surname")

private String surname;

@Column(name = "name")

private String name;

@Column(name = "date\_of\_birth")

private Date dateOfBirth;

@Column(name = "address")

private String address;

@Column(name = "description")

private String description;

@Type(type = "list-array")

@Column(

name = "photos",

columnDefinition = "text[]"

)

private List<String> links;

@JsonIgnore

@ManyToOne

@JoinTable(name = "doctor\_to\_patient",

joinColumns = {@JoinColumn(name = "patient\_id", referencedColumnName = "id")},

inverseJoinColumns = {@JoinColumn(name = "doctor\_id", referencedColumnName = "id")}

)

private Doctor doctor;

@OneToMany(mappedBy = "patient", fetch = FetchType.*LAZY*)

private List<Meeting> meetingList;

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

if (!super.equals(o)) return false;

Patient patient = (Patient) o;

return Objects.*equals*(surname, patient.surname) &&

Objects.*equals*(name, patient.name) &&

Objects.*equals*(dateOfBirth, patient.dateOfBirth) &&

Objects.equals(address, patient.address);

}

@Override

public int hashCode() {

return Objects.hash(super.hashCode(), surname, name, dateOfBirth, address);

}

@Override

public String toString() {

return "Patient{" +

"surname='" + surname + '\'' +

", name='" + name + '\'' +

", dateOfBirth=" + dateOfBirth +

", address='" + address + '\'' +

", description='" + description + '\'' +

", meetingList=" + meetingList +

'}';

}

}

package ru.patientbase.mainAPI.entity;

//Status of meeting/patients/etc

public enum Status {

*ACTIVE*, *NOT\_ACTIVE*, *DELETED*

}

package ru.patientbase.mainAPI.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Modifying;

import org.springframework.data.jpa.repository.Query;

import org.springframework.data.repository.query.Param;

import ru.patientbase.mainAPI.entity.Doctor;

import java.util.Optional;

//Interface for database queries

public interface DoctorRepository extends JpaRepository<Doctor, Long> {

//find doctor by email method

Optional<Doctor> findByEmail(String email);

//modify doctor

@Modifying

@Query("update Doctor u set u.name = :name, u.email = :email where u.id = :id")

Integer modify(@Param("name") String name, @Param("email") String email, @Param("id") Long id);

}

package ru.patientbase.mainAPI.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import ru.patientbase.mainAPI.entity.Meeting;

//Interface for database queries

public interface MeetingRepository extends JpaRepository<Meeting, Long> {

}

package ru.patientbase.mainAPI.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import ru.patientbase.mainAPI.entity.Doctor;

import ru.patientbase.mainAPI.entity.Patient;

import java.util.List;

//Interface for database queries

public interface PatientRepository extends JpaRepository<Patient, Long> {

//Get all patients by doctor

List<Patient> findAllByDoctor(Doctor doctor);

}

package ru.patientbase.mainAPI.rest.exceptionAdvice;

import lombok.extern.slf4j.Slf4j;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

import java.util.NoSuchElementException;

@ControllerAdvice

@Slf4j

public class RestControllerAdvice {

//Exception handler for NoSuchElementExc

@ExceptionHandler({NoSuchElementException.class})

public ResponseEntity<String> handleNoSuchElementException(NoSuchElementException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND).body("\"" + ex.getMessage() + "\"");

}

//Exception handler for IllegalArgumentExc

@ExceptionHandler({IllegalArgumentException.class})

public ResponseEntity<String> handleIllegalArgumentException(IllegalArgumentException ex) {

return ResponseEntity.status(HttpStatus.NOT\_ACCEPTABLE).body("\"" + ex.getMessage() + "\"");

}

//Exception handler for IllegalAccessExc

@ExceptionHandler({IllegalAccessException.class})

public ResponseEntity<String> handleIllegalAccessException(IllegalAccessException ex) {

return ResponseEntity.status(HttpStatus.FORBIDDEN).body("\"" + ex.getMessage() + "\"");

}

}

package ru.patientbase.mainAPI.rest;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.web.bind.annotation.\*;

import ru.patientbase.mainAPI.dto.DoctorDTO;

import ru.patientbase.mainAPI.dto.LoginDto;

import ru.patientbase.mainAPI.dto.RegisterDto;

import ru.patientbase.mainAPI.entity.Doctor;

import ru.patientbase.mainAPI.service.DoctorService;

//Doctor REST Controller class

@RestController

@RequestMapping("/api/v1.0/")

@Slf4j

public class DoctorRestController {

final DoctorService doctorService;

final PasswordEncoder passwordEncoder;

@Autowired

public DoctorRestController(DoctorService doctorService, PasswordEncoder passwordEncoder) {

this.doctorService = doctorService;

this.passwordEncoder = passwordEncoder;

}

//register new doc method

@PostMapping("register")

public void register(@RequestBody RegisterDto registerDto) {

registerDto.setPassword(passwordEncoder.encode(registerDto.getPassword()));

doctorService.register(RegisterDto.*getDoctor*(registerDto));

}

//login method

@PostMapping("login")

public ResponseEntity<DoctorDTO> login(@RequestBody LoginDto loginDto) {

Doctor doctor = doctorService.getDoctorByEmail(loginDto.getEmail());

if(passwordEncoder.matches(loginDto.getPassword(), doctor.getPassword())) {

return ResponseEntity.*ok*(DoctorDTO.*translateToDto*(doctor));

} else {

throw new IllegalArgumentException("Invalid username/password");

}

}

//get doc by id method

@GetMapping("doctor/{id}")

public ResponseEntity<DoctorDTO> getDoctorById(@PathVariable Long id) {

Doctor doctor = doctorService.getDoctorById(id);

*log*.info("getDoctorById with id: " + id);

return ResponseEntity.*ok*(DoctorDTO.*translateToDto*(doctor));

}

//modify doc method

@PatchMapping("doctor/modify")

public void modifyDoctor(@RequestBody DoctorDTO doctorDTO) {

Doctor doctor = DoctorDTO.*getDoctor*(doctorDTO);

doctorService.modifyDoctor(doctor);

}

}

package ru.patientbase.mainAPI.rest;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import ru.patientbase.mainAPI.dto.MeetingDTO;

import ru.patientbase.mainAPI.entity.Meeting;

import ru.patientbase.mainAPI.service.impl.MeetingServiceImpl;

//Meeting REST Controller

@RestController

@RequestMapping("api/v1.0")

public class MeetingRestController {

final

MeetingServiceImpl meetingService;

@Autowired

public MeetingRestController(MeetingServiceImpl meetingService) {

this.meetingService = meetingService;

}

//get meeting by ID

@GetMapping("/meeting/{id}")

public ResponseEntity<MeetingDTO> getMeeting(@PathVariable Long id) {

return ResponseEntity.*ok*(MeetingDTO.*translateToDto*(meetingService.find(id)));

}

//add meeting

@PostMapping("meeting/new/{id}")

public ResponseEntity<MeetingDTO> add(@RequestBody MeetingDTO meetingDTO, @PathVariable Long id) {

meetingDTO.setStatus("ACTIVE");

return ResponseEntity.*ok*(MeetingDTO

.*translateToDto*(

meetingService

.add(id, MeetingDTO.*getMeeting*(meetingDTO))));

}

}

package ru.patientbase.mainAPI.rest;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import ru.patientbase.mainAPI.dto.PatientDTO;

import ru.patientbase.mainAPI.entity.Doctor;

import ru.patientbase.mainAPI.entity.Patient;

import ru.patientbase.mainAPI.entity.Status;

import ru.patientbase.mainAPI.service.DoctorService;

import ru.patientbase.mainAPI.service.PatientService;

import java.sql.Date;

import java.util.List;

import java.util.NoSuchElementException;

import java.util.stream.Collectors;

//Patient REST Controller

@RestController

@Slf4j

@RequestMapping("api/v1.0/")

public class PatientRestController {

final PatientService patientService;

final DoctorService doctorService;

@Autowired

public PatientRestController(DoctorService doctorService, PatientService patientService) {

this.patientService = patientService;

this.doctorService = doctorService;

}

//get all patuents by doctorID

@GetMapping("patient/all/{id}")

public ResponseEntity<List<PatientDTO>> getAll(@PathVariable Long id) {

log.info("Get all patients was invoked");

return ResponseEntity.ok(patientService

.getAll(id)

.stream()

.map(PatientDTO::translateToDto)

.collect(Collectors.toList())

);

}

//add patient for doctor with ID = doctorId

@PostMapping("patient/add/{doctorId}")

public ResponseEntity<PatientDTO> createPatient(@PathVariable Long doctorId, @RequestBody PatientDTO patientDTO) {

Doctor doctor = doctorService.getDoctorById(doctorId);

log.info("CreatePatient was invoked");

if (doctor != null) {

Patient patient = PatientDTO.getPatient(patientDTO);

patient.setDoctor(doctorService.getDoctorById(doctorId));

patient.setStatus(Status.ACTIVE);

patient.setCreated(new Date(System.currentTimeMillis()));

patient.setUpdated(new Date(System.currentTimeMillis()));

Patient saved = patientService.addPatient(patient);

return ResponseEntity.ok(PatientDTO.translateToDto(saved));

} else {

throw new NoSuchElementException("Doctor with id: " + doctorId + " wasn't found");

}

}

//modify patient for doctor with id = doctorId

@PatchMapping("patient/modify/{doctorId}")

public ResponseEntity<PatientDTO> modifyPatient(@PathVariable Long doctorId, @RequestBody PatientDTO patientDTO) throws IllegalAccessException {

Patient patient = patientService.findPatient(patientDTO.getId());

if (!patientService.findPatient(patientDTO.getId()).getDoctor().getId().equals(doctorId)) {

throw new IllegalAccessException("Patient doctor\_id and doctor\_id aren't equal");

}

if (patient != null) {

patient.setName(patientDTO.getName());

patient.setSurname(patientDTO.getSurname());

patient.setLinks(patientDTO.getLinks());

patient.setDescription(patientDTO.getDescription());

patient.setDateOfBirth(patientDTO.getDateOfBirth());

patient.setAddress(patientDTO.getAddress());

patient.setUpdated(new Date(System.currentTimeMillis()));

patientService.modifyPatient(patient);

return ResponseEntity.ok(PatientDTO.translateToDto(patient));

} else {

throw new NoSuchElementException("Patient with id: " + patientDTO.getId() + " wasn't found");

}

}

//Delete patient

@DeleteMapping("patient/delete/{id}")

public void deletePatient(@PathVariable Long id) {

patientService.deletePatient(id);

}

//get patient

@GetMapping("patient/{id}")

public ResponseEntity<PatientDTO> getPatient(@PathVariable Long id) {

return ResponseEntity.ok(PatientDTO.translateToDto(patientService.findPatient(id)));

}

}

package ru.patientbase.mainAPI.service.impl;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import ru.patientbase.mainAPI.entity.Doctor;

import ru.patientbase.mainAPI.entity.Status;

import ru.patientbase.mainAPI.repository.DoctorRepository;

import ru.patientbase.mainAPI.service.DoctorService;

import javax.transaction.Transactional;

import java.sql.Date;

import java.util.NoSuchElementException;

import java.util.Optional;

//Implementation of interface

@Service

@Slf4j

public class DoctorServiceImpl implements DoctorService {

private final DoctorRepository doctorRepository;

@Autowired

public DoctorServiceImpl(DoctorRepository doctorRepository) {

this.doctorRepository = doctorRepository;

}

//get doc service method

@Override

public Doctor getDoctorById(Long id) throws NoSuchElementException {

Optional<Doctor> doctor = doctorRepository.findById(id);

if (doctor.isPresent()) {

return doctor.get();

} else {

throw new NoSuchElementException("Such doctor wasn't found");

}

}

//get doc by email service method

@Override

public Doctor getDoctorByEmail(String email) throws NoSuchElementException {

Optional<Doctor> doctor = doctorRepository.findByEmail(email);

if (doctor.isPresent()) {

return doctor.get();

} else {

throw new NoSuchElementException("Such doctor wasn't found");

}

}

//modify doc service method

@Override

@Transactional

public void modifyDoctor(Doctor doctor) throws NoSuchElementException {

Optional<Doctor> doc = doctorRepository.findById(doctor.getId());

if (doc.isPresent()) {

doctorRepository.modify(doctor.getName(), doctor.getEmail(), doctor.getId());

} else {

throw new NoSuchElementException("Such doctor wasn't found");

}

}

//reg doc service method

@Override

public void register(Doctor doctor) {

Optional<Doctor> optionalDoctor = doctorRepository.findByEmail(doctor.getEmail());

if(optionalDoctor.isPresent()) {

throw new IllegalArgumentException("Doctor with such email already exists!");

} else {

doctor.setCreated(new Date(System.currentTimeMillis()));

doctor.setUpdated(new Date(System.currentTimeMillis()));

doctor.setStatus(Status.ACTIVE);

doctorRepository.save(doctor);

}

}

}

package ru.patientbase.mainAPI.service.impl;

import lombok.extern.slf4j.Slf4j;

import org.springframework.stereotype.Service;

import ru.patientbase.mainAPI.entity.Meeting;

import ru.patientbase.mainAPI.entity.Patient;

import ru.patientbase.mainAPI.entity.Status;

import ru.patientbase.mainAPI.repository.MeetingRepository;

import ru.patientbase.mainAPI.repository.PatientRepository;

import ru.patientbase.mainAPI.service.MeetingService;

import java.sql.Date;

import java.util.NoSuchElementException;

import java.util.Optional;

//Implementation of meeting interface

@Service

@Slf4j

public class MeetingServiceImpl implements MeetingService {

final PatientRepository patientRepository;

final MeetingRepository meetingRepository;

public MeetingServiceImpl(PatientRepository patientRepository, MeetingRepository meetingRepository) {

this.patientRepository = patientRepository;

this.meetingRepository = meetingRepository;

}

//add new meeting

@Override

public Meeting add(Long patientId, Meeting meeting) {

Optional<Patient> optionalPatient = patientRepository.findById(patientId);

if (!optionalPatient.isPresent()) {

throw new NoSuchElementException("There is no patient with id: " + patientId);

}

meeting.setCreated(new Date(System.currentTimeMillis()));

meeting.setUpdated(new Date(System.currentTimeMillis()));

Patient patient = optionalPatient.get();

meeting.setPatient(patient);

Meeting saved = meetingRepository.save(meeting);

patient.getMeetingList().add(saved);

System.*out*.println(patient.getMeetingList());

patientRepository.save(patient);

return saved;

}

//delete meeting

@Override

public void delete(Long id) {

Optional<Meeting> optionalMeeting = meetingRepository.findById(id);

if (optionalMeeting.isPresent()) {

Meeting meeting = optionalMeeting.get();

meeting.setStatus(Status.*DELETED*);

meetingRepository.save(meeting);

} else {

throw new NoSuchElementException("Meeting with id: " + id + " wasn't found");

}

}

//modify meeting

@Override

public void modify(Meeting meeting) {

Optional<Meeting> optionalMeeting = meetingRepository.findById(meeting.getId());

if (optionalMeeting.isPresent()) {

meetingRepository.save(meeting);

} else {

throw new NoSuchElementException("There is no meeting with id: " + meeting.getId());

}

}

//find meeting by ID

@Override

public Meeting find(Long id) {

Optional<Meeting> optionalMeeting = meetingRepository.findById(id);

if (optionalMeeting.isPresent()) {

return optionalMeeting.get();

} else {

throw new NoSuchElementException("There is no meeting with id: " + id);

}

}

}

package ru.patientbase.mainAPI.service.impl;

import lombok.extern.slf4j.Slf4j;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import ru.patientbase.mainAPI.entity.Doctor;

import ru.patientbase.mainAPI.entity.Patient;

import ru.patientbase.mainAPI.entity.Status;

import ru.patientbase.mainAPI.repository.DoctorRepository;

import ru.patientbase.mainAPI.repository.PatientRepository;

import ru.patientbase.mainAPI.service.PatientService;

import java.util.List;

import java.util.NoSuchElementException;

import java.util.Optional;

//Implementation of PatientService interface

@Service

@Slf4j

public class PatientServiceImpl implements PatientService {

final PatientRepository patientRepository;

final DoctorRepository doctorRepository;

@Autowired

public PatientServiceImpl(DoctorRepository doctorRepository, PatientRepository patientRepository) {

this.patientRepository = patientRepository;

this.doctorRepository = doctorRepository;

}

//add new Patient

@Override

public Patient addPatient(Patient patient) {

if (!doctorRepository.findById(patient.getDoctor().getId()).isPresent()) {

throw new NoSuchElementException("Doctor wasn't found");

}

if (patient.getName() == null || patient.getSurname() == null ||

patient.getAddress() == null || patient.getDescription() == null ||

patient.getDateOfBirth() == null) {

throw new IllegalArgumentException("Some fields are null!");

}

return patientRepository.save(patient);

}

//modify patient

@Override

public void modifyPatient(Patient patient) {

if (patientRepository.findById(patient.getId()).isPresent()) {

patientRepository.save(patient);

} else {

throw new NoSuchElementException("Such patient wasn't found");

}

}

//Delete patient

@Override

public void deletePatient(Long id) {

Optional<Patient> optionalPatient = patientRepository.findById(id);

if (optionalPatient.isPresent()) {

Patient patient = optionalPatient.get();

patient.setStatus(Status.*DELETED*);

patientRepository.save(patient);

} else {

throw new NoSuchElementException("Such patient wasn't found");

}

}

//Find patient by id

@Override

public Patient findPatient(Long id) {

Optional<Patient> patient = patientRepository.findById(id);

if (patient.isPresent()) {

return patient.get();

} else {

throw new NoSuchElementException("Patient with id: " + id + " wasn't found");

}

}

//get all patients

@Override

public List<Patient> getAll(Long id) {

Optional<Doctor> doctor = doctorRepository.findById(id);

if (doctor.isPresent()) {

return patientRepository.findAllByDoctor(doctor.get());

} else {

throw new NoSuchElementException("Doctor with id: " + id + " wasn't found");

}

}

}

package ru.patientbase.mainAPI.service;

import ru.patientbase.mainAPI.entity.Doctor;

import java.util.NoSuchElementException;

//interface for DOC service

public interface DoctorService {

Doctor getDoctorById(Long id) throws NoSuchElementException;

Doctor getDoctorByEmail(String email) throws NoSuchElementException;

void modifyDoctor(Doctor doctor) throws NoSuchElementException;

void register(Doctor doctor);

}

package ru.patientbase.mainAPI.service;

import ru.patientbase.mainAPI.entity.Meeting;

// interface for Meeting Service

public interface MeetingService {

Meeting add(Long patientId, Meeting meeting);

void delete(Long id);

void modify(Meeting meeting);

Meeting find(Long id);

}

package ru.patientbase.mainAPI.service;

import ru.patientbase.mainAPI.entity.Patient;

import java.util.List;

//interface for Patint service

public interface PatientService {

Patient addPatient(Patient patient);

void modifyPatient(Patient patient);

void deletePatient(Long id);

Patient findPatient(Long id);

List<Patient> getAll(Long id);

}

package ru.patientbase.mainAPI;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

//Main class

@SpringBootApplication

public class MainApiApplication {

public static void main(String[] args) {

SpringApplication.*run*(MainApiApplication.class, args);

}

}

server.port=8075

spring.datasource.url=jdbc:postgresql://localhost:5432/patientbase

spring.datasource.driver-class-name=org.postgresql.Driver

spring.datasource.username=patientbase

spring.datasource.password=patientbase

spring.liquibase.url=jdbc:postgresql://localhost:5432/patientbase

spring.liquibase.change-log=classpath:db/changelog/changelog-master.xml

spring.liquibase.user=patientbase

spring.liquibase.password=patientbase

plugins **{**

id 'org.springframework.boot' version '2.3.1.RELEASE'

id 'io.spring.dependency-management' version '1.0.9.RELEASE'

id 'java'

**}**

group = 'ru.patientbase'

version = '0.0.1-SNAPSHOT'

sourceCompatibility = '11'

configurations **{**

compileOnly **{**

extendsFrom annotationProcessor

**}**

**}**

repositories **{**

mavenCentral()

**}**

dependencies **{**

implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

implementation 'org.springframework.boot:spring-boot-starter-mail'

implementation 'org.springframework.boot:spring-boot-starter-web'

implementation 'org.liquibase:liquibase-core'

compileOnly 'org.projectlombok:lombok'

developmentOnly 'org.springframework.boot:spring-boot-devtools'

runtimeOnly 'org.postgresql:postgresql'

annotationProcessor 'org.springframework.boot:spring-boot-configuration-processor'

annotationProcessor 'org.projectlombok:lombok'

testImplementation('org.springframework.boot:spring-boot-starter-test') **{**

exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'

**}**

// https://mvnrepository.com/artifact/junit/junit

testCompile group: 'junit', name: 'junit', version: '4.13'

// https://mvnrepository.com/artifact/com.vladmihalcea/hibernate-types-52

compile group: 'com.vladmihalcea', name: 'hibernate-types-52', version: '2.9.12'

// https://mvnrepository.com/artifact/org.springframework.security/spring-security-crypto

compile group: 'org.springframework.security', name: 'spring-security-crypto', version: '5.3.2.RELEASE'

testImplementation 'org.springframework.security:spring-security-test'

**}**

jar **{**

manifest.attributes("Main-Class": 'ru.patientbase.mainAPI.MainApiApplication');

**}**

test **{**

useJUnitPlatform()

**}**

# Frontend:

import React, {Component} from 'react';

import '../styles/register.css';

import '../styles/addPatient.css';

import {postData} from "../logic/ServerConnection";

class AddMeeting extends Component {

//This file contains component wich allows to add new Meeting

constructor(props) {

super(props);

this.state = {

error: "",

date: "",

organisation: "",

description: "",

links: []

}

}

//on submit send data to server

handleSubmit = (event) => {

event.preventDefault();

let state = this.state;

if (state.organisation === "" || state.description === "" || state.date === "") {

this.setState({error: "Не все поля заполнены!"});

} else {

postData('/api/v1.0/meeting/new/' + this.props.match.params.id, JSON.stringify(this.state))

.then(res => {

if(res.id > 0) {

alert("Новый прием добавлен!");

this.props.history.push("/meeting/" + res.id);

} else {

console.log(res);

}

});

}

return false;

}

handleChange = event => {

this.setState({

[event.target.id]: event.target.value

});

};

render() {

return (

<div className="registerForm">

<form onSubmit={this.handleSubmit}>

<h1 id="signUp">Новый прием</h1>

<div className="form">

<p id="regFormText">Дата приема</p>

<input type="date" id="date" value={this.state.date}

onChange={this.handleChange}/>

<p id="regFormText">Организация</p>

<input type="text" id="organisation" size="50" value={this.state.organisation}

onChange={this.handleChange}/>

<p id="regFormText">Протокол приема</p>

<textarea id="description" onChange={this.handleChange}>{this.state.description}</textarea>

<p id="error">{this.state.error}</p>

<input type="submit" id="submit" value="Отправить"/>

</div>

</form>

</div>

);

}

}

export default AddMeeting;

import React, {Component} from 'react';

import '../styles/register.css';

import '../styles/addPatient.css';

import {postData} from "../logic/ServerConnection";

class AddPatient extends Component {

//this file contains AddPatient component logic

constructor(props) {

super(props);

this.label = "";

this.state = {

error: "",

surname: "",

name: "",

dateOfBirth: "",

address: "",

description: "",

links: []

}

}

//on submit send data to server

handleSubmit = (event) => {

event.preventDefault();

let state = this.state;

if (state.surname === "" || state.name === "" || state.dateOfBirth === "" || state.address === "" || state.description === "") {

this.setState({error: "Не все поля заполнены!"});

} else {

postData('/api/v1.0/patient/add/' + this.props.docId, ***JSON***.stringify(this.state))

.then(res => {

if(res.id > 0) {

alert("Пациент добавлен!");

this.props.history.push("/patient/" + res.id);

} else {

***console***.log(res);

}

});

}

return false;

}

handleChange = event => {

this.setState({

[event.target.id]: event.target.value

});

};

render() {

return (

<div className="registerForm">

<form onSubmit={this.handleSubmit}>

<h1 id="signUp">Add new patient</h1>

<div className="form">

<p id="regFormText">Фамилия</p>

<input type="text" id="surname" size="50" value={this.state.surname}

onChange={this.handleChange}/>

<p id="regFormText">Имя и отчество</p>

<input type="text" id="name" size="50" value={this.state.name} onChange={this.handleChange}/>

<p id="regFormText">Дата рождения</p>

<input type="date" id="dateOfBirth" value={this.state.dateOfBirth}

onChange={this.handleChange}/>

<p id="regFormText">Адрес</p>

<input type="text" id="address" size="50" value={this.state.address}

onChange={this.handleChange}/>

<p id="regFormText">Протокол приема</p>

<textarea id="description" onChange={this.handleChange}>{this.state.description}</textarea>

<p id="error">{this.state.error}</p>

<h2>{this.label}</h2>

<input type="submit" id="submit" value="Отправить"/>

</div>

</form>

</div>

);

}

}

export default AddPatient;

import React, { Component } from 'react';

import '../styles/content.css'

import Register from "./Register";

import Login from "./Login";

import AddPatient from "./AddPatient";

import Patient from "./Patient";

import Meeting from "./Meeting";

import ListOfPatients from "./ListOfPatients";

import Doctor from './Doctor';

import ModifyDoctor from './ModifyDoctor';

import Info from "./Info";

import {Route, Switch} from "react-router-dom";

import AddMeeting from "./AddMeeting";

class Content extends Component {

//Main Component for various types of content

render() {

return (

<div className="content">

<Switch>

<Route exact path='/' component={Info}/>

<Route path='/doctor/:id' component={Doctor}/>

<Route path='/list' render={props => (<ListOfPatients {...props} docId={this.props.docId}/>)}/>

<Route exact path='/patient/:id' component={Patient}/>

<Route path='/login' render={props => (<Login {...props} setId={this.props.setId}/>)}/>

<Route path='/register' component={Register}/>

<Route path='/add' render={props => (<AddPatient {...props} docId={this.props.docId}/>)}/>

<Route exact path='/meeting/:id' component={Meeting}/>

<Route path='/meeting/add/:id' component={AddMeeting}/>

</Switch>

</div>

);

}

}

export default Content;

import React, { Component } from 'react';

import '../styles/doctor.css';

class Doctor extends Component {

//Component which represents doctors info

state = {

id: this.props.match.params.id,

name: "",

email: "",

status: "",

created: ""

};

componentDidMount() {

fetch("/api/v1.0/doctor/" + this.state.id)

.then(res => res.json())

.then(

(result) => {

this.setState(result);

},

(error) => {

console.log(error);

}

)

}

render() {

return (

<div className="doctor">

<h1 id="header">Информация о вас</h1>

<p id="text">Имя: {this.state.name}</p>

<p id="text">Email: {this.state.email}</p>

<p id="text">Статус: {this.state.status}</p>

<p id="text">Учетная запись создана: {this.state.created}</p>

<button id="modifyButton">Изменить данные</button>

</div>

);

}

}

export default Doctor;

import React, { Component } from 'react';

import '../styles/footer.css'

class Footer extends Component {

//Footer component

render() {

return (

<div className="footer">

<p id="email">Email: DJPonnYash.Vinyl@gmail.com<a id="git" href="https://github.com/nastypill">Мой Github</a></p>

</div>

);

}

}

export default Footer;

import React, { Component } from 'react';

class Info extends Component {

//Info about web-site

render() {

return (

<div className="info">

<div id="info">

<p>Данный сайт предоставляет возможность врачам разных профилей и специализаций

вести простой и понятный учет больных без использования бумажных носителей.

Легко и просто добавить пацеинета, указать всю необходимую информацию о нем,

прикрепить сканы документов, которые вам необходимы, и вы будете всегда знать

кто к вам пришел, и приходил ли он ранее. Это сильно облегчит процесс постановки точного

диагноза, а также позволит сократить время приема без потери его эффективности!</p>

</div>

</div>

);

}

}

export default Info;

import React, {Component} from 'react';

import "../styles/link.css"

import "../styles/linksItem.css"

import {***Link***} from "react-router-dom";

import LogLink from "./LogLink";

import RegLink from "./RegLink";

class Links extends Component {

//Links for endpoints

render() {

return (

<div className="links">

{/\*<Link to="/">\*/}

{/\* <button key="1" className="linkButton">\*/}

{/\* <h3 className="linkDescription">Main</h3>\*/}

{/\* </button>\*/}

{/\*</Link>\*/}

<Link to={this.props.isLogged ? '/list' : '/login'}>

<button key="2" className="linkButton">

<h3 className="linkDescription">Список пациентов</h3>

</button>

</Link>

<Link to={this.props.isLogged ? '/doctor/' + this.props.docId : '/login'}>

<button key="3" className="linkButton">

<h3 className="linkDescription">Профиль</h3>

</button>

</Link>

{this.props.isLogged ? "" : <LogLink/>}

{this.props.isLogged ? "" : <RegLink/>}

</div>

);

}

}

export default Links;

import React, {Component} from 'react';

import '../styles/miniMeeting.css'

import '../styles/listOfPatients.css'

import Patient from "./Patient";

class ListOfPatients extends Component {

//Component for listOfPatients

//Из listOfPatients делаю запрос в базу, создаю экземпляры miniPatient передавая данные в props

constructor(props) {

super(props);

this.state = {

search: "",

list: [],

sublist: []

}

}

//When mount ask server for data

componentDidMount() {

fetch("/api/v1.0/patient/all/" + this.props.docId)

.then(res => res.json())

.then(

(result) => {

***console***.log(result);

this.setState({list: result, sublist: result});

},

(error) => {

***console***.log(error);

}

)

}

//on change of TF filter patients by this TF

handleChange = event => {

this.setState({

[event.target.id]: event.target.value

});

let val = event.target.value.trim().toLowerCase()

this.setState({sublist: this.state.list.filter(elem => elem.surname.toLowerCase().startsWith(val))})

console.log(this.state.sublist)

};

handleClick(event, id){

console.log(id);

this.props.history.push('/patient/' + id);

}

handleAdd = (e) => {

this.props.history.push("/add")

}

render() {

return (

<div className="list">

<p id="searchHeader">Найти пациента</p>

<div className="inp">

<input id="search" type="text" value={this.state.search} onChange={this.handleChange}/>

<button id="plusButton" onClick={this.handleAdd}>Добавить</button>

</div>

{this.state.sublist.map((patient) => {

return <div className="miniPatient" key={patient.id} onClick={e => this.handleClick(e, patient.id)}>

<p id="miniDivText"><b id="miniB" key={patient.id}>{patient.surname}</b> {patient.name}</p>

<p id="miniDivText"><b id="miniB" key={patient.id}>Дата рождения:</b> {patient.dateOfBirth}</p>

<p id="miniDivText" key={patient.id}>{patient.address}</p>

</div>

})}

</div>

);

}

}

export default ListOfPatients;

import React, {Component} from 'react';

import '../styles/register.css'

import {postData} from "../logic/ServerConnection";

class Login extends Component {

//Login form component

constructor(props) {

super(props);

this.state = {

email: "",

password: "",

error: ""

}

}

handleChange = event => {

this.setState({

[event.target.id]: event.target.value

});

};

//On submit ask server for such doctor

handleSubmit = event => {

event.preventDefault();

postData("/api/v1.0/login", ***JSON***.stringify(this.state))

.then(res => {

if (res.id > 0) {

this.props.history.push("/doctor/" + res.id)

this.props.setId(res.id);

} else {

this.setState({error: res});

}

}).catch((e) => {

***console***.log(e);

})

}

render() {

return (

<div className="registerForm">

<form onSubmit={this.handleSubmit}>

<h1 id="signUp">Sign in</h1>

<div className="fields">

<p id="regFormText">Email</p>

<input type="text" size="50" id="email" value={this.state.name} onChange={this.handleChange}/>

<p id="regFormText">Password</p>

<input type="password" size="50" id="password" value={this.state.password}

onChange={this.handleChange}/>

<p id="passError">{this.state.error}</p>

<input type="submit" id="regButton" value="Войти"/>

</div>

</form>

</div>

);

}

}

export default Login;

import React, {Component} from 'react';

import '../styles/linksItem.css'

import {***Link***} from "react-router-dom";

class LogRegLink extends Component {

//Login link component

render() {

return (

<Link to="/login">

<button key="4" className="linkButton">

<h3 className="linkDescription">Войти</h3>

</button>

</Link>

);

}

}

export default LogRegLink;

import React, {Component} from 'react';

import "../styles/patient.css"

class Meeting extends Component {

//Meeting represents component

constructor(props) {

super(props);

this.state = {

id: this.props.match.params.id,

surname: "",

name: "",

dateOfBirth: "",

address: "",

date: "",

organisation: "",

description: "",

photos: []

}

}

//When mount ask server for meeting's data

componentDidMount() {

fetch("/api/v1.0/meeting/" + this.state.id)

.then(res => res.json())

.then(

(result) => {

console.log(result);

this.setState({

surname: result.patientDTO.surname,

name: result.patientDTO.name,

dateOfBirth: result.patientDTO.dateOfBirth,

address: result.patientDTO.address,

date: result.date,

organisation: result.organisation,

description: result.description

})

if(result.links !== null) {

this.setState({photos: result.links})

}

},

(error) => {

console.log(error);

}

)

}

render() {

return (

<div className="meeting">

<div id="info">

<div id="header">Пациент: {this.state.surname}</div>

<div id="text"><b>Имя:</b> {this.state.name}</div>

<div id="text"><b>Адрес:</b> {this.state.address}</div>

<div id="date"><b>Дата рождения:</b> {this.state.dateOfBirth}</div>

<div id="date"><b>Дата приема:</b> {this.state.date}</div>

<div id="text"><b>Организация:</b> {this.state.organisation}</div>

<div id="clinicDesc"><b>Клиническое описание: </b>{this.state.description} </div>

<div id="text"><b>Сканы документов: </b></div>

<div id="images">

{this.state.photos.map((photo) => {

return <img className="photo" key={photo} src={photo}/>

})}

</div>

</div>

</div>

);

}

}

export default Meeting;

import React, {Component} from 'react';

import '../styles/register.css'

import {patchData} from "../logic/ServerConnection";

class Register extends Component {

//Form for doctor's modifying query

constructor(props) {

super(props);

this.state = {

id: 1,

name: "",

email: "",

status: "ACTIVE",

error: ""

}

}

handleChange = event => {

this.setState({

[event.target.id]: event.target.value

});

};

//On submit if all fields were filled -> send to server

handleSubmit = event => {

event.preventDefault();

if (this.state.name === "" || this.state.email === "") {

this.setState({error: "Не все поля заполнены!"})

} else {

patchData('/api/v1.0/doctor/modify', JSON.stringify(this.state));

}

}

render() {

return (

<div className="registerForm">

<form onSubmit={this.handleSubmit}>

<h1 id="signUp">Введите изменения</h1>

<div className="fields">

<p id="regFormText">Имя</p>

<input type="text" size="50" id="name" value={this.state.name} onChange={this.handleChange}/>

<p id="regFormText">Email</p>

<input type="text" size="50" id="email" value={this.state.email} onChange={this.handleChange}/>

<p id="error">{this.state.error}</p>

<input type="submit" id="regButton" value="Готово!"/>

</div>

</form>

</div>

);

}

}

export default Register;

import React, {Component} from 'react';

import '../styles/patient.css'

import '../styles/miniMeeting.css'

import '../styles/register.css'

import {***Link***} from "react-router-dom";

class Patient extends Component {

//PAtient component

constructor(props) {

super(props);

this.state = {

id: this.props.match.params.id,

surname: "",

name: "",

dateOfBirth: "",

address: "",

description: "",

photos: [],

meetings: []

}

}

//When mount ask server for patient's data

componentDidMount() {

fetch("/api/v1.0/patient/" + this.state.id)

.then(res => res.json())

.then(

(result) => {

this.setState(result);

},

(error) => {

console.log(error);

}

)

}

handleClick(e) {

}

render() {

return (

<div className="patient">

<div id="info">

<div id="header">Пациент: {this.state.surname}<button id="add" onClick={this.handleAddClick}>Новый прием</button></div>

<div id="text"><b>Имя:</b> {this.state.name}</div>

<div id="text"><b>Адрес:</b> {this.state.address}</div>

<div id="date"><b>Дата рождения:</b> {this.state.dateOfBirth}</div>

<div id="clinicDesc"><b>Клиническое описание: </b>{this.state.description} </div>

<div className="miniMeetings" onClick={e => this.handleClick(e)}>

{this.state.meetings.map((meeting) => {

return <div className="miniMeeting" key={meeting.id}>

<Link to={"/meeting/" + meeting.id}>

<p id="miniDivText">{meeting.organisation}</p>

<p id="miniDivText">{meeting.date}</p>

</Link>

</div>

})}

</div>

<div id="text"><b>Сканы документов: </b></div>

<div id="images">

{this.state.photos.map((photo) => {

return <img className="photo" key={photo} src={photo}/>

})}

</div>

</div>

</div>

);

}

handleAddClick = event => {

this.props.history.push("/meeting/add/" + this.state.id);

}

}

export default Patient;

import React, {Component} from 'react';

import '../styles/register.css'

import {postData} from "../logic/ServerConnection";

class Register extends Component {

//Register Form

constructor(props) {

super(props);

this.state = {

name: "",

email: "",

password: "",

label: ""

}

}

handleChange = event => {

this.setState({

[event.target.id]: event.target.value

});

};

//Check passwords equality

handlePassword = event => {

console.log(this.props.password);

console.log(event.target.value);

if (this.state.password.localeCompare(event.target.value) === 0) {

this.setState({

label: ""

})

} else {

this.setState({

label: "Пароли не совпадают"

})

}

}

//on submit send data to server if all OK

handleSubmit = event => {

event.preventDefault();

if (this.state.name === "" || this.state.email === "" || this.state.password === "") {

this.setState({label: "Every field should be filled"});

} else {

if (this.state.password.length < 8) {

this.setState({label: "Password shouldn't be shorter then 8 letters"})

} else {

let data = {

name: this.state.name,

email: this.state.email,

password: this.state.password

};

postData("api/v1.0/register", JSON.stringify(data))

.then(res => {

if (res === null || res === "") {

this.props.history.push("/login")

} else {

this.setState({label: res})

}

}).catch(e => {

this.props.history.push("/login")

});

}

}

}

render() {

return (

<div className="registerForm">

<form onSubmit={this.handleSubmit}>

<h1 id="signUp">Create your account</h1>

<div className="fields">

<p id="regFormText">Имя</p>

<input type="text" size="50" id="name" value={this.state.name} onChange={this.handleChange}/>

<p id="regFormText">Email</p>

<input type="text" size="50" id="email" value={this.state.email} onChange={this.handleChange}/>

<p id="regFormText">Пароль</p>

<input type="password" size="50" id="password" value={this.state.password}

onChange={this.handleChange}/>

<p id="regFormText">Введите пароль повторно</p>

<input type="password" size="50" id="password" onChange={this.handlePassword}/>

<p id="passError">{this.state.label}</p>

<input type="submit" id="regButton" value="Зарегистрироваться"/>

</div>

</form>

</div>

);

}

}

export default Register;

import React, {Component} from 'react';

import '../styles/linksItem.css'

import {***Link***} from "react-router-dom";

class LogRegLink extends Component {

//Register Link component

render() {

return (

<Link to="/register">

<button key="5" className="linkButton">

<h3 className="linkDescription">Зарегистрироваться</h3>

</button>

</Link>

);

}

}

export default LogRegLink;

import React, { Component } from 'react';

import Header from './Header';

import "../styles/wrapper.css"

import Content from "./Content";

import Footer from "./Footer";

class Wrapper extends Component {

//Wrapper component, contains footer, header, content.

constructor() {

super();

this.state = {

docId: -1,

isLogged: false

}

this.setDocId = this.setDocId.bind(this)

}

setDocId(docId) {

this.setState({docId: docId, isLogged: true});

}

render() {

return (

<div className="wrapper">

<Header docId={this.state.docId} isLogged={this.state.isLogged}/>

<Content setId={this.setDocId} docId={this.state.docId} isLogged={this.state.isLogged}/>

<Footer/>

</div>

);

}

}

export default Wrapper;

export async function postData(url = '', data = {}) {

// Default options are marked with \*

const response = await fetch(url, {

method: 'POST', // \*GET, POST, PUT, DELETE, etc.

headers: {

'Content-Type': 'application/json'

// 'Content-Type': 'application/x-www-form-urlencoded',

},

body: data // body data type must match "Content-Type" header

});

return await response.json(); // parses JSON response into native JavaScript objects

}

export async function patchData(url = '', data = {}) {

// Default options are marked with \*

const response = await fetch(url, {

method: 'PATCH', // \*GET, POST, PUT, DELETE, etc.

headers: {

'Content-Type': 'application/json'

// 'Content-Type': 'application/x-www-form-urlencoded',

},

body: data // body data type must match "Content-Type" header

});

return await response.json(); // parses JSON response into native JavaScript objects

}

#description {

height: 400px;

}

p {

margin-top: 2%;

}

.registerForm {

margin-top: -5%;

}

#error{

color: red;

}

.content{

min-height: 90%;

margin-top: 50px;

margin-right: 15%;

margin-left: 15%;

height: auto;

width: auto;

background-color: #e7e7e7;

padding-top: 5%;

padding-left: 5%;

padding-right: 5%;

}

#text {

width: 70%;

max-width: 70%;

background-color: #ffffff;

border-width: 1px;

}

#modifyButton {

margin: auto;

width: 70%;

height: 30px;

margin-top: 5%;

font-size: 22px;

background-color: #ffffff;

border-style: solid;

border-radius: 5px;

border-color: #9f9f9f;

border-width: 1px;

}

#modifyButton,

#modifyButton:active,

#modifyButton:focus {

outline: none;

}

.doctor {

margin-top: 5%;

padding: 2%;

text-align: center;

border-style: solid;

border-width: 2px;

border-color: #909090;

border-radius: 5px;

background-color: #f0f0f0;

}

.footer {

background-color: #e0e0e0 ;

height: 50px;

width: auto;

margin-top: 0;

padding: 5px;

}

#git {

margin-left: 60%;

}

@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap');

.header {

display: inline-list-item;

width: 100%;

min-width: 700px;

height: 50px;

background-color: #e0e0e0;

position: fixed;

z-index: 101;

margin-top: -50px;

border-bottom-style: solid;

border-bottom-width: 2px;

border-bottom-color: #00a0a0;

}

#headerText {

font-family: 'Roboto', sans-serif;

font-weight: 300;

margin-top: 0.5%;

font-size: 26px;

margin-left: 5%;

margin-bottom: 0px;

float: left;

}

.links {

float: right;

width: auto;

height: 100%;

margin-right: 15%;

}

@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap');

.linkButton {

margin-left: 2px;

width: auto;

height: 100%;

background-color: #f0f0f0;

border-style: none;

padding-left: 20px;

padding-right: 20px;

cursor: pointer;

}

.linkButton:hover {

background-color: #eaeaea;

border-bottom-style: solid;

border-bottom-color: #606060;

border-bottom-width: 2px;

border-top-style: none;

}

.linkButton:active{

background-color: #dadada;

}

.linkButton,

.linkButton:active,

.linkButton:focus {

outline: none;

}

.linkDescription {

font-family: 'Roboto', sans-serif;

font-weight: 300;

}

@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap');

#searchHeader{

font-family: 'Roboto', sans-serif;

font-weight: 200;

font-size: 50px;

margin-top: -1%;

}

#search{

margin-top: 2%;

margin-bottom: 3%;

}

.list{

text-align: center;

}

#plusButton {

width: auto;

height: auto;

padding: 4px;

border-style: none;

border-radius: 5px;

background-color: #ffffff;

margin-left: 1%;

font-size: 16px;

}

#plusButton,

#plusButton:focus,

#plusButton:active {

outline: none;

}

#plusButton:hover {

background-color: #e0e0e0;

}

.miniPatient, .miniMeeting {

padding-left: 2%;

padding-bottom: 1%;

width: 20%;

min-height: 40px;

border-bottom-style: solid;

border-width: 2px;

border-radius: 25px;

border-color: #00a0a0;

background-color: #ffffff;

float: left;

margin-right: 2.5%;

margin-left: 2.5%;

margin-bottom: 3%;

}

.miniPatient:hover, .miniMeeting:hover {

background-color: #dadada;

cursor: pointer;

}

.list, .miniMeetings {

margin-top: 3%;

display: inline-block;

width: 100%;

}

.miniPatient {

width: 43%;

}

#miniB {

font-weight: 400;

}

@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap');

.meeting, .patient, .info {

width: auto;

text-align: center;

}

#header{

font-family: 'Roboto', sans-serif;

font-weight: 400;

font-size: 32px;

margin-bottom: 5%;

}

#text, #date, #clinicDesc{

text-align: left;

margin: auto;

max-width: 50%;

border-style: none;

border-radius: 5px;

background-color: #ffffff;

font-family: 'Roboto', sans-serif;

font-weight: 400;

font-size: 18px;

margin-top: 1%;

word-wrap: break-word;

}

#clinicDesc {

max-width: 90%;

min-height: 300px;

text-align: left;

}

#images {

margin-top: 3%;

display: inline-block;

width: 95%;

}

.photo {

transition: 1s;

width: 40%;

margin-right: 5%;

}

.photo:hover {

transform: scale(3);

}

#add {

margin-left: 4%;

width: auto;

height: auto;

padding: 4px;

border-style: none;

border-radius: 5px;

background-color: #ffffff;

font-size: 20px;

font-weight: 300;

}

#add,

#add:focus,

#add:active {

outline: none;

}

#add:hover {

background-color: #e0e0e0;

}

a {

text-decoration: none;

color: #000000;

font-family: 'Roboto', sans-serif;

font-weight: 400;

font-size: 16px;

}

@import url('https://fonts.googleapis.com/css2?family=Roboto:wght@100&display=swap');

#passError {

color: red;

}

p {

font-family: 'Roboto', sans-serif;

font-weight: 300;

margin-top: 4%;

}

input {

width: 60%;

height: 25px;

font-size: 18px;

border-style: hidden;

border-radius: 5px;

margin-top: 4px;

}

input,

input:active,

input:focus {

outline: none;

}

textarea {

resize: none;

width: 80%;

height: 250px;

font-size: 18px;

border-style: hidden;

border-radius: 5px;

margin-top: 4px;

}

textarea,

textarea:active,

textarea:focus {

outline: none;

}

.registerForm {

text-align: center;

margin-top: 10%;

}

#regButton {

height: 40px;

width: 61%;

border-radius: 5px;

}

#regButton:hover{

border-bottom-style: solid;

border-bottom-width: 3px;

height: 43px;

}

#regButton:active{

background-color: #e0e0e0;

border-style: none;

border-bottom-style: solid;

border-bottom-width: 3px;

height: 43px;

}

#regButton,

#regButton:active,

#regButton:focus {

outline: none;

}

#signUp {

font-family: 'Roboto', sans-serif;

font-weight: 300;

font-size: 50px;

}

.fields {

text-align: left;

margin-left: 30%;

width: 60%;

height: 30%;

}

.wrapper{

display: block;

width: auto;

min-width: 700px;

max-height: 98vh;

height: 100vh;

}

\* {

margin: 0;

}

import React, {Component} from 'react';

import { ***BrowserRouter*** } from 'react-router-dom';

import './App.css';

import Wrapper from "./components/Wrapper";

class App extends Component {

//MAIN component

render() {

return (

<BrowserRouter>

<Wrapper/>

</BrowserRouter>

);

}

}

export default App;

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

it('renders without crashing', () => {

const div = ***document***.createElement('div');

ReactDOM.render(<App />, div);

});

body {

margin: 0;

padding: 0;

font-family: sans-serif;

}

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import './index.css';

//Main file, starts all APP

ReactDOM.render(

<App />,

***document***.getElementById('root')

);

{

"name": "patientbase",

"version": "0.1.0",

"private": true,

"dependencies": {

"express": "^4.17.1",

"react": "^16.13.1",

"react-dom": "^16.13.1",

"react-scripts": "0.9.5"

},

"devDependencies": {

"react-router-dom": "^5.2.0"

},

"scripts": {

"start": "react-scripts start",

"build": "react-scripts build",

"test": "react-scripts test --env=jsdom",

"eject": "react-scripts eject"

},

"proxy": "http://localhost:8075"

}