Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский экономический университет имени Г.В. Плеханова»

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ**

специальность 09.02.07 «Информационные системы и программирование»

Квалификация: Программист

ИНДИВИДУАЛЬНЫЙ ПРОЕКТ

УП 04.01 ВППО

|  |  |
| --- | --- |
| Выполнил студент  группы П50-6-20  Баранов Андрей Викторович | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_ Д.В. Серяк  «\_\_\_» \_\_\_\_\_ 2023 года |

Москва 2023

ОГЛАВЛЕНИЕ

[1. СОДЕРЖАНИЕ 3](#_Toc154631085)

[2. ТЕМА 4](#_Toc154631086)

[3. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc154631087)

[4. СХЕМА БАЗЫ ДАННЫХ 6](#_Toc154631088)

[5. СЛОВАРЬ ДАННЫХ 8](#_Toc154631089)

[6. СКРИПТ БАЗЫ ДАННЫХ 9](#_Toc154631090)

[7. КОД ПРОГРАММЫ 11](#_Toc154631091)

[8. ДЕМОНСТРАЦИЯ ПРОГРАММЫ 31](#_Toc154631092)

[9. ВЫВОД 37](#_Toc154631093)

# СОДЕРЖАНИЕ

Разработка реляционной базы данных с REST API, программного интерфейса, и веб приложения на языке программирования Kotlin с помощью фреймворка Spring.

# ТЕМА

Индивидуальные проект написан на тему «Форум».

# ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

Форум – это веб-платформа, предназначенная для обмена мнениями и информацией между пользователями по различным темам. Основная структура форума включает в себя разделы и подразделы, которые организованы по тематическим категориям. В каждом разделе пользователи могут создавать темы (треды), в которых они обсуждают специфические вопросы или идеи.

Каждая тема на форуме представляет собой цепочку сообщений (постов), размещенных пользователями. Пользователи могут отвечать на сообщения других участников, цитировать их, а также использовать различные средства форматирования текста для выделения ключевых моментов своего сообщения. Кроме того, на форумах часто присутствует система рейтинга, позволяющая оценивать посты других участников.

Форумы могут быть открытыми, где регистрация не требуется для просмотра и участия в обсуждениях, или закрытыми, где доступ возможен только после регистрации. На многих форумах присутствует роль модераторов и администраторов, которые следят за соблюдением правил форума, управляют контентом и решают конфликтные ситуации между пользователями.

Форумы играют важную роль в сетевом общении, позволяя пользователям обмениваться информацией, получать помощь, делиться опытом и мнениями по интересующим их темам. Они могут быть посвящены самым разнообразным темам: от технологий и программирования до хобби и развлечений.

# СХЕМА БАЗЫ ДАННЫХ

* 1. Логическая схема базы данных

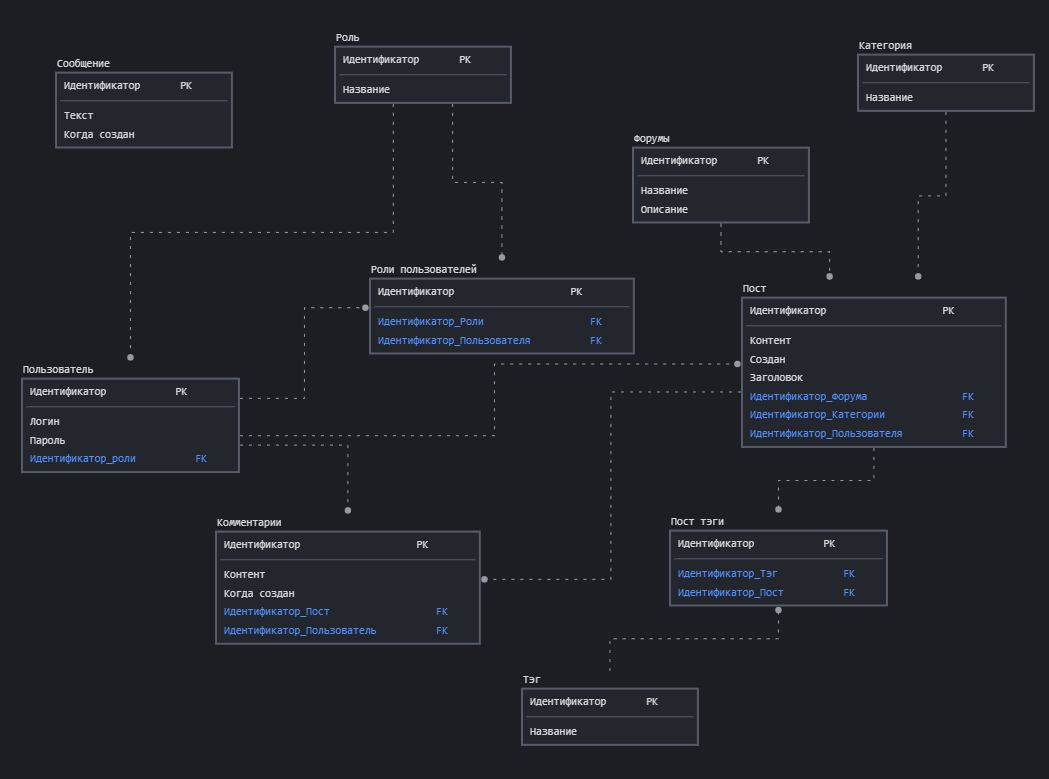


Рисунок – Логическая схема данных

* 1. Физическая схема базы данных

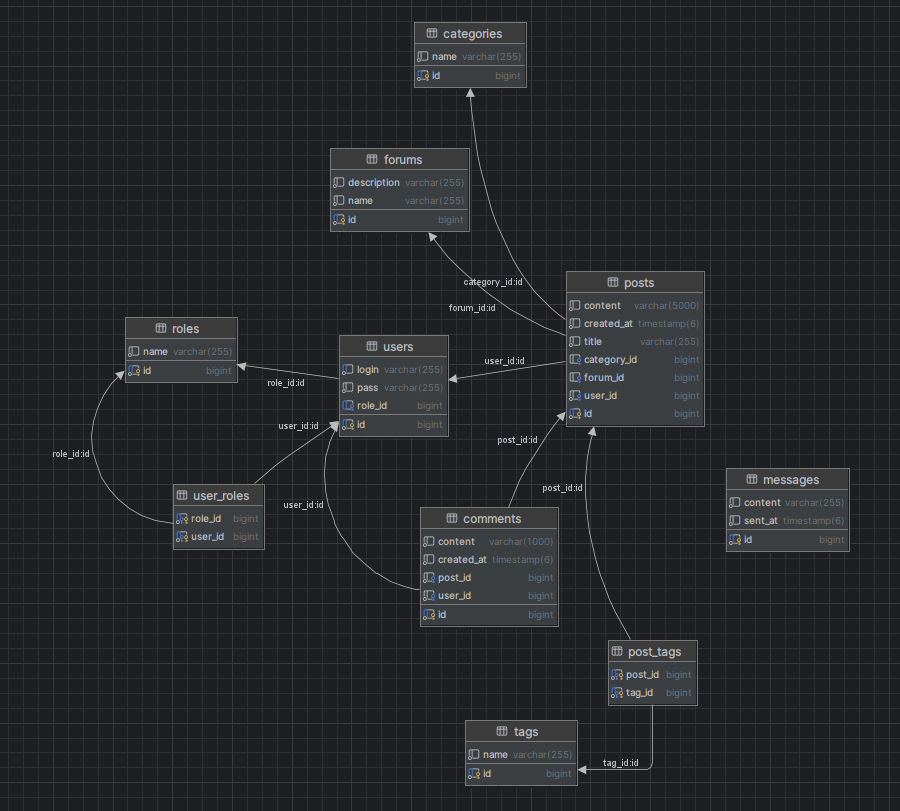


Рисунок – Физическая схема данных

# СЛОВАРЬ ДАННЫХ

Таблица – Словарь данных

|  |  |  |  |
| --- | --- | --- | --- |
| Ключ | Наименование | Тип данных | Примечание |
| PK | id | bigint | Уникальный идентификатор |
|  | name | varchar(255) | Название пользователя/категории/тега/форума |
|  | description | varchar(255) | Описание форума/категории |
|  | login | varchar(255) | Логин пользователя |
|  | pass | varchar(255) | Пароль пользователя |
|  | title | varchar(255) | Заголовок поста |
|  | content | varchar(5000) | Содержимое поста/комментария/сообщения |
|  | createdAt | timestamp(6) | Время создания поста/комментария |
|  | sent\_at | timestamp(6) | Время отправки сообщения |
| FK | user\_id | bigint | Внешний ключ, ссылка на пользователя |
| FK | post\_id | bigint | Внешний ключ, ссылка на пост |
| FK | role\_id | bigint | Внешний ключ, ссылка на роль |
| FK | forum\_id | bigint | Внешний ключ, ссылка на форум |
| FK | category\_id | bigint | Внешний ключ, ссылка на категорию |
| FK | tag\_id | bigint | Внешний ключ, ссылка на тег |
| M2M | user\_roles |  | Связь многие-ко-многим между пользователями и ролями |
| M2M | post\_tags |  | Связь многие-ко-многим между постами и тегами |

# СКРИПТ БАЗЫ ДАННЫХ

create table if not exists public.categories

(

id bigserial

primary key,

name varchar(255) not null

);

alter table public.categories

owner to postgres;

create table if not exists public.forums

(

id bigserial

primary key,

description varchar(255) not null,

name varchar(255) not null

);

alter table public.forums

owner to postgres;

create table if not exists public.messages

(

id bigserial

primary key,

content varchar(255) not null,

sent\_at timestamp(6) not null

);

alter table public.messages

owner to postgres;

create table if not exists public.roles

(

id bigserial

primary key,

name varchar(255) not null

);

alter table public.roles

owner to postgres;

create table if not exists public.tags

(

id bigserial

primary key,

name varchar(255) not null

);

alter table public.tags

owner to postgres;

create table if not exists public.users

(

id bigserial

primary key,

login varchar(255) not null

constraint uk\_ow0gan20590jrb00upg3va2fn

unique,

pass varchar(255) not null,

role\_id bigint

constraint uk\_krvotbtiqhudlkamvlpaqus0t

unique

constraint fkp56c1712k691lhsyewcssf40f

references public.roles

);

alter table public.users

owner to postgres;

create table if not exists public.posts

(

id bigserial

primary key,

content varchar(5000) not null,

created\_at timestamp(6) not null,

title varchar(255) not null,

category\_id bigint

constraint fkijnwr3brs8vaosl80jg9rp7uc

references public.categories,

forum\_id bigint

constraint fk9bleycktuep8yrcvuuveugtqf

references public.forums,

user\_id bigint not null

constraint fk5lidm6cqbc7u4xhqpxm898qme

references public.users

);

alter table public.posts

owner to postgres;

create table if not exists public.comments

(

id bigserial

primary key,

content varchar(1000) not null,

created\_at timestamp(6) not null,

post\_id bigint not null

constraint fkh4c7lvsc298whoyd4w9ta25cr

references public.posts,

user\_id bigint not null

constraint fk8omq0tc18jd43bu5tjh6jvraq

references public.users

);

alter table public.comments

owner to postgres;

create table if not exists public.post\_tags

(

post\_id bigint not null

constraint fkkifam22p4s1nm3bkmp1igcn5w

references public.posts,

tag\_id bigint not null

constraint fkm6cfovkyqvu5rlm6ahdx3eavj

references public.tags,

primary key (post\_id, tag\_id)

);

alter table public.post\_tags

owner to postgres;

create table if not exists public.user\_roles

(

role\_id bigint not null

constraint fkh8ciramu9cc9q3qcqiv4ue8a6

references public.roles,

user\_id bigint not null

constraint fkhfh9dx7w3ubf1co1vdev94g3f

references public.users,

primary key (role\_id, user\_id)

);

alter table public.user\_roles

owner to postgres;

# КОД ПРОГРАММЫ

1. CategoryController.kt

package org.example.itog.controller

import org.example.itog.model.Category

import org.example.itog.model.UserRole

import org.example.itog.repositories.CategoryRepository

import org.springframework.stereotype.Controller

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.bind.annotation.RequestParam

import org.springframework.web.servlet.ModelAndView

@Controller

@RequestMapping("/cat")

class CategoryController(val catRepo:CategoryRepository) {

@GetMapping

fun index() : ModelAndView {

val modelAndView = ModelAndView("category")

modelAndView.addObject("cats", catRepo.findAll())

return modelAndView

}

@PostMapping

@RequestMapping("/create")

fun createRole(@RequestParam name:String) : String {

catRepo.save(Category().apply { this.name = name })

return "redirect:/cat"

}

@PostMapping

@RequestMapping("/change", params = ["delete"])

fun deleteRole(@RequestParam roleId:Long) : String {

catRepo.deleteById(roleId)

return "redirect:/cat"

}

@PostMapping

@RequestMapping("/change", params = ["edit"])

fun updateRole(@RequestParam roleId:Long, @RequestParam name:String) : String {

//roleRepo.deleteById(roleId)

catRepo.findById(roleId).get().let {

it.name = name

catRepo.save(it)

}

return "redirect:/cat"

}

}

2. ForumController.kt

package org.example.itog.controller

import jakarta.websocket.server.PathParam

import org.example.itog.model.Comment

import org.example.itog.model.Forum

import org.example.itog.model.Post

import org.example.itog.repositories.ForumRepository

import org.example.itog.repositories.PostRepository

import org.example.itog.repositories.UserRepository

import org.springframework.security.core.annotation.AuthenticationPrincipal

import org.springframework.security.core.context.SecurityContextHolder

import org.springframework.security.core.userdetails.UserDetails

import org.springframework.stereotype.Controller

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PathVariable

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.bind.annotation.RequestParam

import org.springframework.web.servlet.ModelAndView

import java.security.Principal

import kotlin.jvm.optionals.getOrElse

import kotlin.jvm.optionals.getOrNull

@Controller

@RequestMapping("/forum")

class ForumController(

val forumRepo:ForumRepository,

val postRepo:PostRepository,

val userRepo:UserRepository,

val commentRepo:UserRepository,

) {

@GetMapping

fun index() : ModelAndView {

val modelAndView = ModelAndView("forums")

modelAndView.addObject("forums", forumRepo.findAll())

return modelAndView

}

@GetMapping

@RequestMapping("/{id}")

fun getForum(@PathVariable id:Long) : ModelAndView {

val modelAndView = ModelAndView("forum")

modelAndView.addObject("forum", forumRepo.findById(id).getOrElse {

throw IllegalArgumentException("No forum $id")

})

return modelAndView

}

@PostMapping

@RequestMapping("/add\_comment")

fun addComment(

@RequestParam forumId:Long,

@RequestParam postId:Long,

@RequestParam content:String

) : String {

val user = userRepo.getByLogin(SecurityContextHolder.getContext().authentication.name)

val post = postRepo.findById(postId).get()

user!!.comments.add(Comment().apply {

this.user = user

this.post = post

this.content = content

})

userRepo.save(user)

return "redirect:/forum/${forumId}"

}

@PostMapping

@RequestMapping("/post")

fun createPost(

@RequestParam forumId: Long,

@RequestParam title: String,

@RequestParam content: String

) : String {

//println("Cred: ${SecurityContextHolder.getContext().authentication.name}")

val user = userRepo.getByLogin(SecurityContextHolder.getContext().authentication.name)

//println("USere: $user")

user!!.posts.add(Post().apply {

this.title = title

this.content = content

this.user = user

this.forum = forumRepo.findById(forumId).get()

})

userRepo.save(user)

return "redirect:/forum/${forumId}"

}

@PostMapping

@RequestMapping("/create")

fun create(@RequestParam name:String, @RequestParam desc:String) : String {

forumRepo.save(Forum().apply {

this.name = name

this.description = desc

})

return "redirect:/forums"

}

}

3. MainController.kt

package org.example.itog.controller

import org.springframework.stereotype.Controller

import org.springframework.web.bind.annotation.GetMapping

@Controller

class MainController {

@GetMapping

fun index() : String {

return "index"

}

}

4. MessageController.kt

package org.example.itog.controller

import org.example.itog.model.Message

import org.example.itog.repositories.MessageRepository

import org.springframework.stereotype.Controller

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.servlet.ModelAndView

@Controller

@RequestMapping("message")

class MessageController(

val msgRepo:MessageRepository

) {

@GetMapping

fun index() : ModelAndView {

val mav = ModelAndView("message")

mav.addObject("messages", msgRepo.findAll())

return mav

}

@PostMapping

@RequestMapping("/send")

fun sendMsg(msg:String) : String {

val msg = Message().apply {

this.content = msg

}

msgRepo.save(msg)

return "redirect:/message"

}

}

5. RegisterController.kt

package org.example.itog.controller

import org.example.itog.model.User

import org.example.itog.repositories.RoleRepository

import org.example.itog.repositories.UserRepository

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

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

import org.springframework.stereotype.Controller

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.bind.annotation.RequestParam

@Controller

@RequestMapping("register")

class RegisterController(val repo: UserRepository,

val hasher: PasswordEncoder = BCryptPasswordEncoder(),

val roleRepo:RoleRepository

) {

@GetMapping

fun index() : String {

return "register"

}

@PostMapping

fun register(@RequestParam login:String, @RequestParam password:String) : String {

val neww = User()

neww.login = login

neww.role = roleRepo.findByName("User")

neww.pass = hasher.encode(password)

repo.save(neww)

return "redirect:login"

}

}

6. RoleController.kt

package org.example.itog.controller

import org.example.itog.model.UserRole

import org.example.itog.repositories.RoleRepository

import org.example.itog.repositories.UserRepository

import org.springframework.stereotype.Controller

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.bind.annotation.RequestParam

import org.springframework.web.servlet.ModelAndView

@Controller

@RequestMapping("role")

class RoleController(

val userRepository: UserRepository,

val roleRepo:RoleRepository

) {

@GetMapping

fun index() : ModelAndView {

val users = userRepository.findAll()

val modelAndView = ModelAndView("role-edit")

modelAndView.addObject("users", users)

modelAndView.addObject("roles", roleRepo.findAll())

return modelAndView

}

@PostMapping

@RequestMapping("/create")

fun createRole(@RequestParam name:String) : String {

roleRepo.save(UserRole().apply { this.name = name })

return "redirect:/role"

}

@PostMapping

@RequestMapping("/change", params = ["delete"])

fun deleteRole(@RequestParam roleId:Long) : String {

roleRepo.deleteById(roleId)

return "redirect:/role"

}

@PostMapping

@RequestMapping("/change", params = ["edit"])

fun updateRole(@RequestParam roleId:Long, @RequestParam name:String) : String {

//roleRepo.deleteById(roleId)

roleRepo.findById(roleId).get().let {

it.name = name

roleRepo.save(it)

}

return "redirect:/role"

}

@PostMapping

@RequestMapping("/change-user-role")

fun changeUserRole(

@RequestParam("userId") userId: Long,

@RequestParam("roleId") roleId: Long?

): String {

val user = userRepository.findById(userId).orElse(null)

//val role = roleRepository.findById(roleId).orElse(null)

if (user != null && roleId != null) {

user.role = roleRepo.findById(roleId).get()

userRepository.save(user)

}

return "redirect:/role-edit"

}

}

7. TagController.kt

package org.example.itog.controller

import org.example.itog.model.Tag

import org.example.itog.repositories.TagRepository

import org.springframework.stereotype.Controller

import org.springframework.web.bind.annotation.GetMapping

import org.springframework.web.bind.annotation.PostMapping

import org.springframework.web.bind.annotation.RequestMapping

import org.springframework.web.bind.annotation.RequestParam

import org.springframework.web.servlet.ModelAndView

@Controller

@RequestMapping("/tag")

class TagController(

val tagRepo:TagRepository

) {

@GetMapping

fun index() : ModelAndView {

val modelAndView = ModelAndView("tags")

modelAndView.addObject("tags", tagRepo.findAll())

return modelAndView

}

@PostMapping

@RequestMapping("/create")

fun createRole(@RequestParam name:String) : String {

tagRepo.save(Tag().apply { this.name = name })

return "redirect:/tag"

}

@PostMapping

@RequestMapping("/change", params = ["delete"])

fun deleteRole(@RequestParam roleId:Long) : String {

tagRepo.deleteById(roleId)

return "redirect:/tag"

}

@PostMapping

@RequestMapping("/change", params = ["edit"])

fun updateRole(@RequestParam roleId:Long, @RequestParam name:String) : String {

//roleRepo.deleteById(roleId)

tagRepo.findById(roleId).get().let {

it.name = name

tagRepo.save(it)

}

return "redirect:/tag"

}

}

8. ItogApplication.kt

package org.example.itog

import org.springframework.boot.autoconfigure.SpringBootApplication

import org.springframework.boot.runApplication

@SpringBootApplication

class ItogApplication

fun main(args: Array<String>) {

runApplication<ItogApplication>(\*args)

}

9. Category.kt

package org.example.itog.model

import jakarta.persistence.\*

import java.time.LocalDateTime

@Entity

@Table(name = "categories")

class Category {

@Id @GeneratedValue(strategy = GenerationType.IDENTITY)

val id: Long = 0

@Column(nullable = false)

var name: String = ""

@OneToMany(mappedBy = "category", cascade = [CascadeType.ALL], fetch = FetchType.LAZY)

val posts: List<Post> = mutableListOf()

}

10. Comment.kt

package org.example.itog.model

import jakarta.persistence.\*

import java.time.LocalDateTime

@Entity

@Table(name = "comments")

class Comment {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

var id: Long = 0

@Column(nullable = false, length = 1000)

var content: String = ""

@Column(nullable = false)

var createdAt: LocalDateTime = LocalDateTime.now()

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "post\_id", nullable = false)

var post: Post? = null

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "user\_id", nullable = false)

var user: User? = null

}

11. Forum.kt

package org.example.itog.model

import jakarta.persistence.\*

@Entity

@Table(name = "forums")

class Forum {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

val id: Long = 0

@Column(nullable = false)

var name: String = ""

@Column(nullable = false)

var description: String = ""

@OneToMany(mappedBy = "forum", cascade = [CascadeType.ALL], fetch = FetchType.LAZY)

val posts: List<Post> = mutableListOf()

}

12. Message.kt

package org.example.itog.model

import jakarta.persistence.\*

import java.time.LocalDateTime

@Entity

@Table(name = "messages")

class Message {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

val id: Long = 0

@Column(nullable = false)

var content: String = ""

@Column(nullable = false)

val sentAt: LocalDateTime = LocalDateTime.now()

}

13. Post.kt

package org.example.itog.model

import jakarta.persistence.\*

import java.time.LocalDateTime

@Entity

@Table(name = "posts")

class Post {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

val id: Long = 0

@Column(nullable = false)

var title: String = ""

@Column(nullable = false, length = 5000)

var content: String = ""

@Column(nullable = false)

val createdAt: LocalDateTime = LocalDateTime.now()

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "user\_id", nullable = false)

var user: User? = null

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "category\_id")

var category: Category? = null

@ManyToOne(fetch = FetchType.LAZY)

@JoinColumn(name = "forum\_id")

var forum: Forum? = null

@OneToMany(mappedBy = "post", cascade = [CascadeType.ALL], fetch = FetchType.LAZY)

val comments: MutableList<Comment> = mutableListOf()

@ManyToMany(fetch = FetchType.LAZY)

@JoinTable(

name = "post\_tags",

joinColumns = [JoinColumn(name = "post\_id")],

inverseJoinColumns = [JoinColumn(name = "tag\_id")]

)

val tags: MutableSet<Tag> = mutableSetOf()

}

14. Tag.kt

package org.example.itog.model

import jakarta.persistence.\*

import java.time.LocalDateTime

@Entity

@Table(name = "tags")

class Tag {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

val id: Long = 0

@Column(nullable = false)

var name: String = ""

@ManyToMany(mappedBy = "tags", cascade = [CascadeType.ALL], fetch = FetchType.LAZY)

val posts: Set<Post> = mutableSetOf()

}

15. User.kt

package org.example.itog.model

import jakarta.persistence.\*

import org.springframework.security.core.GrantedAuthority

import org.springframework.security.core.authority.SimpleGrantedAuthority

import org.springframework.security.core.userdetails.UserDetails

@Entity

@Table(name = "users")

class User : UserDetails {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

val id: Long = 0

constructor()

constructor(login:String, password:String, role:UserRole){

this.login = login

this.pass = password

this.role = role

}

@Column(nullable = false, unique = true)

var login: String = ""

@Column(nullable = false)

var pass: String = ""

@OneToMany(mappedBy = "user", cascade = [CascadeType.ALL], fetch = FetchType.LAZY)

val posts: MutableList<Post> = mutableListOf()

@OneToMany(mappedBy = "user", cascade = [CascadeType.ALL], fetch = FetchType.LAZY)

val comments: MutableList<Comment> = mutableListOf()

@OneToOne(fetch = FetchType.EAGER, cascade = [CascadeType.ALL])

var role: UserRole? = null

override fun getAuthorities(): MutableCollection<out GrantedAuthority> {

val a = role?.name ?: "GUEST"

println("A: $a")

return mutableListOf(SimpleGrantedAuthority(a))

}

override fun getPassword(): String {

return pass

}

override fun getUsername(): String {

return login

}

override fun isAccountNonExpired(): Boolean {

return true

}

override fun isAccountNonLocked(): Boolean {

return true

}

override fun isCredentialsNonExpired(): Boolean {

return true

}

override fun isEnabled(): Boolean {

return true

}

}

16. UserRole.kt

package org.example.itog.model

import jakarta.persistence.\*

@Entity

@Table(name = "roles")

class UserRole {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

var id: Long = 0

@Column(nullable = false)

var name: String = ""

}

17. CategoryRepository.kt

package org.example.itog.repositories

import org.example.itog.model.Category

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

interface CategoryRepository : JpaRepository<Category, Long> {

}

18. CommentRepository.kt

package org.example.itog.repositories

import org.example.itog.model.Comment

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

interface CommentRepository : JpaRepository<Comment, Long> {

}

19. ForumRepository.kt

package org.example.itog.repositories

import org.example.itog.model.Forum

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

interface ForumRepository : JpaRepository<Forum, Long> {

}

20. MessageRepository.kt

package org.example.itog.repositories

import org.example.itog.model.Message

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

interface MessageRepository : JpaRepository<Message, Long> {

}

21. PostRepository.kt

package org.example.itog.repositories

import org.example.itog.model.Post

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

interface PostRepository : JpaRepository<Post, Long> {

}

22. RoleRepository.kt

package org.example.itog.repositories

import org.example.itog.model.UserRole

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

interface RoleRepository : JpaRepository<UserRole, Long> {

fun findByName(name:String) : UserRole?

}

23. TagRepository.kt

package org.example.itog.repositories

import org.example.itog.model.Tag

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

interface TagRepository : JpaRepository<Tag, Long> {

}

24. UserRepository.kt

package org.example.itog.repositories

import org.example.itog.model.User

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

interface UserRepository : JpaRepository<User, Long> {

fun getByLogin(login: String): User?

}

25. WebSecurityConfig.kt

package org.example.itog

import org.example.itog.model.User

import org.example.itog.repositories.RoleRepository

import org.example.itog.repositories.UserRepository

import org.springframework.context.annotation.Configuration

import org.springframework.security.config.annotation.web.builders.HttpSecurity

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity

import org.springframework.security.core.userdetails.UserDetailsService

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

import org.springframework.context.annotation.Bean

import org.springframework.security.config.Customizer.withDefaults

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder

import org.springframework.security.core.userdetails.UserDetails

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

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

import org.springframework.security.web.DefaultSecurityFilterChain

import org.springframework.stereotype.Service

@Service

class PersonDetailsService(

private val userRepository: UserRepository,

private val roleRepo: RoleRepository,

) : UserDetailsService {

override fun loadUserByUsername(username: String): UserDetails {

if(username == "root")

return User(

"root",

BCryptPasswordEncoder().encode("root"),

roleRepo.findByName("Root")!!

)

val user = userRepository.getByLogin(username)

if(user!=null){

return User(

user.login,

user.pass,

user.role!!

)

}else{

return User(

"guest",

BCryptPasswordEncoder().encode("guest"),

roleRepo.findByName("Guest")!!

)

}

}

}

@Configuration

@EnableWebSecurity

class WebSecurityConfig(

private val userDetailsService: PersonDetailsService,

) {

@Bean

protected fun passwordEncoder(): PasswordEncoder {

return BCryptPasswordEncoder()

}

@Autowired

fun configureGlobal(auth: AuthenticationManagerBuilder) {

auth.userDetailsService(userDetailsService)

}

@Bean

fun configure(http: HttpSecurity): DefaultSecurityFilterChain? {

http

.authorizeHttpRequests {

it.requestMatchers("/register", "/login", "/logout").permitAll()

.requestMatchers(

"/",

"/index",

"/message",

"/forum",

"/register",

).permitAll()

.requestMatchers(

"/forum/\*\*",

"/forum/post/\*\*",

"/message/\*\*",

"/message/send",

).hasAnyAuthority("User", "Admin")

.requestMatchers(

"/role",

"/role/\*\*",

"/cat/\*\*",

"/tag/\*\*").hasAnyAuthority("Admin")

.requestMatchers("/role-edit", "/change-user-role").hasAnyAuthority("Admin")

.anyRequest().hasAuthority("Admin")

}.formLogin(withDefaults())

.logout {

it.logoutUrl("/logout") // URL, на который отправляется запрос для выхода

.logoutSuccessUrl("/login?logout") // URL, на который перенаправляется пользователь после выхода

.permitAll()

}

return http.build()

}

}

26. category.html

<h1>Category edit</h1>

<form action="/cat/create" th:method="POST">

<input type="hidden" name="\_csrf" th:value="${\_csrf.token}" />

<label>

<input name="name" type="text"/>

</label>

<button type="submit">Create New Role</button>

</form>

<form action="/cat/change" th:type="POST" th:each="cat : ${cats}">

<input type="hidden" name="roleId" th:value="${cat.getId()}">

<label>

<input type="text" name="name" th:value="${cat.getName()}">

</label>

<input type="submit" name="delete" th:value="Delete"/>

<input type="submit" name="edit" th:value="Edit"/>

</form>

27. forum.html

<h1 th:text="${'Форум ' + forum.name}"></h1>

<h6 th:text="${'Описание: ' + forum.description}"></h6>

<div th:each="post : ${forum.getPosts()}">

<span style="color: green; border-left: 2px solid green;" th:text="${post.getTitle()}"></span>

<div style="margin-left: 20px;" th:each="comment : ${post.getComments()}">

<span style="font-size: 10px;color: white; border: 2px solid darkseagreen; border-radius: 4px; background-color: darkseagreen"

th:text="${comment.getUser().getUsername() + '(' + comment.getUser().getRole().getName() + ')'}"></span>

<span style="font-size: 10px;" th:text="${comment.content}"></span>

</div>

<form th:type="POST" action="/forum/add\_comment">

<input type="hidden" name="\_csrf" th:value="${\_csrf.token}" />

<input type="hidden" name="forumId" th:value="${forum.getId()}" />

<input type="hidden" name="postId" th:value="${post.getId()}">

<span>Comment</span>

<input type="text" name="content"/>

<input type="submit" value="New comment"/>

</form>

</div>

<form th:type="POST" action="/forum/post">

<input type="hidden" name="\_csrf" th:value="${\_csrf.token}" />

<input type="hidden" name="forumId" th:value="${forum.getId()}" />

<span>Title</span>

<input type="text" name="title"/>

<span>Content</span>

<input type="text" name="content"/>

<input type="submit" value="Create post"/>

</form>

28. forums.html

<h1>Forums</h1>

<div th:each="forum : ${forums}">

<a th:href="${'/forum/' + forum.getId()}" th:text="${forum.getName()}"></a>

</div>

<h2>Create new forum</h2>

<form th:type="POST" action="/forum/create">

<input type="hidden" name="\_csrf" th:value="${\_csrf.token}" />

<label>

<input type="text" name="name"/>

</label>

<label>

<input type="text" name="desc"/>

</label>

<input type="submit" value="Create forum">

</form>

29. index.html

<h1>Forum</h1>

<form th:type="GET" action="/forum">

<input type="submit" value="Форумы"/>

</form>

<form th:type="GET" action="/message">

<input type="submit" value="Глобальный чат"/>

</form>

<form th:type="GET" action="/role">

<input type="submit" value="Роли"/>

</form>

<form th:type="GET" action="/tag">

<input type="submit" value="Тэги"/>

</form>

<form th:type="GET" action="/cat">

<input type="submit" value="Категории"/>

</form>

30. message.html

<h1>Глобальный чат</h1>

<div style="margin-left: 20px;" th:each="msg : ${messages}">

<span style="font-size: 10px;" th:text="${msg.content}"></span>

</div>

<form th:type="POST" action="/message/send">

<input type="hidden" name="\_csrf" th:value="${\_csrf.token}" />

<input type="text" name="msg"/>

<input type="submit"/>

</form>

31. register.html

<h1>Register page</h1>

<form th:action="@{/register}" th:method="POST">

<label>

<input type="text" name="login" />

</label>

<label>

<input type="password" name="password" />

</label>

<input type="submit" value="Register" />

</form>

32. role-edit.html

<!DOCTYPE html>

<html>

<head>

<title>Role Edit</title>

</head>

<body>

<h2>Edit User Role</h2>

<form action="/role/change-user-role" th:method="POST">

<input type="hidden" name="\_csrf" th:value="${\_csrf.token}" />

<select name="userId">

<option value="">Select User</option>

<!-- Здесь должны быть добавлены элементы для каждого пользователя -->

<option th:each="user : ${users}" th:value="${user.id}" th:text="${user.username}"></option>

</select>

<select name="roleId">

<option value="">Select Role</option>

<!-- Здесь должны быть добавлены элементы для каждой роли -->

<option th:each="role : ${roles}" th:value="${role.id}" th:text="${role.name}"></option>

</select>

<button type="submit">Change Role</button>

</form>

<form action="/role/create" th:method="POST">

<input type="hidden" name="\_csrf" th:value="${\_csrf.token}" />

<label>

<input name="name" type="text"/>

</label>

<button type="submit">Create New Role</button>

</form>

<form action="/role/change" th:type="POST" th:each="role : ${roles}">

<input type="hidden" name="roleId" th:value="${role.getId()}">

<label>

<input type="text" name="name" th:value="${role.getName()}">

</label>

<input type="submit" name="delete" th:value="Delete"/>

<input type="submit" name="edit" th:value="Edit"/>

</form>

</body>

</html>

33. tags.html

<h1>Tags edit</h1>

<form action="/tag/create" th:method="POST">

<input type="hidden" name="\_csrf" th:value="${\_csrf.token}" />

<label>

<input name="name" type="text"/>

</label>

<button type="submit">Create New Role</button>

</form>

<form action="/tag/change" th:type="POST" th:each="tag : ${tags}">

<input type="hidden" name="roleId" th:value="${tag.getId()}">

<label>

<input type="text" name="name" th:value="${tag.getName()}">

</label>

<input type="submit" name="delete" th:value="Delete"/>

<input type="submit" name="edit" th:value="Edit"/>

</form>

34. ItogApplicationTests.kt

package org.example.itog

import org.junit.jupiter.api.Test

import org.springframework.boot.test.context.SpringBootTest

@SpringBootTest

class ItogApplicationTests {

@Test

fun contextLoads() {

}

}

# ДЕМОНСТРАЦИЯ ПРОГРАММЫ



Рисунок – Страница авторизации

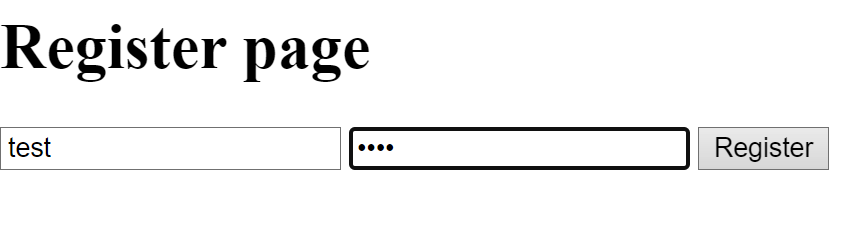


Рисунок – Страница регистрации

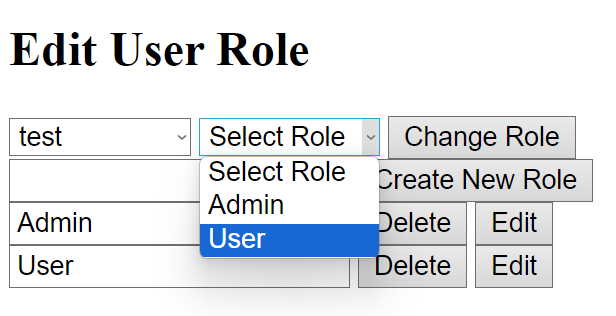


Рисунок – Страница управления ролями

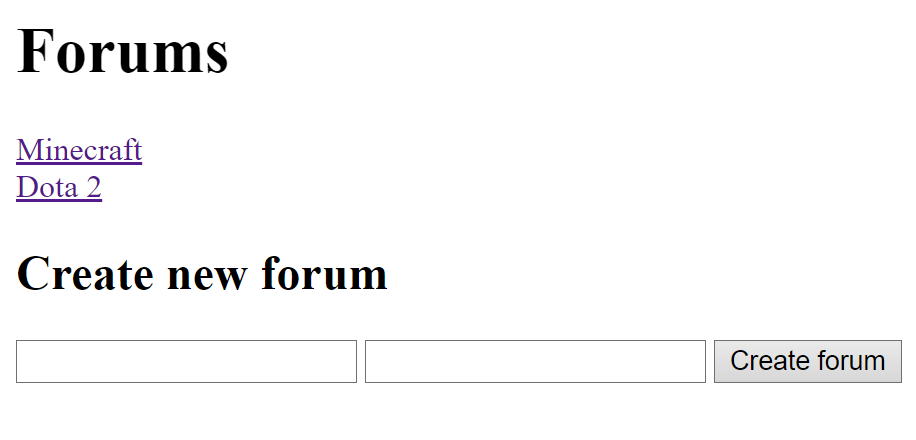


Рисунок – Страница всех форумов, Создание форума

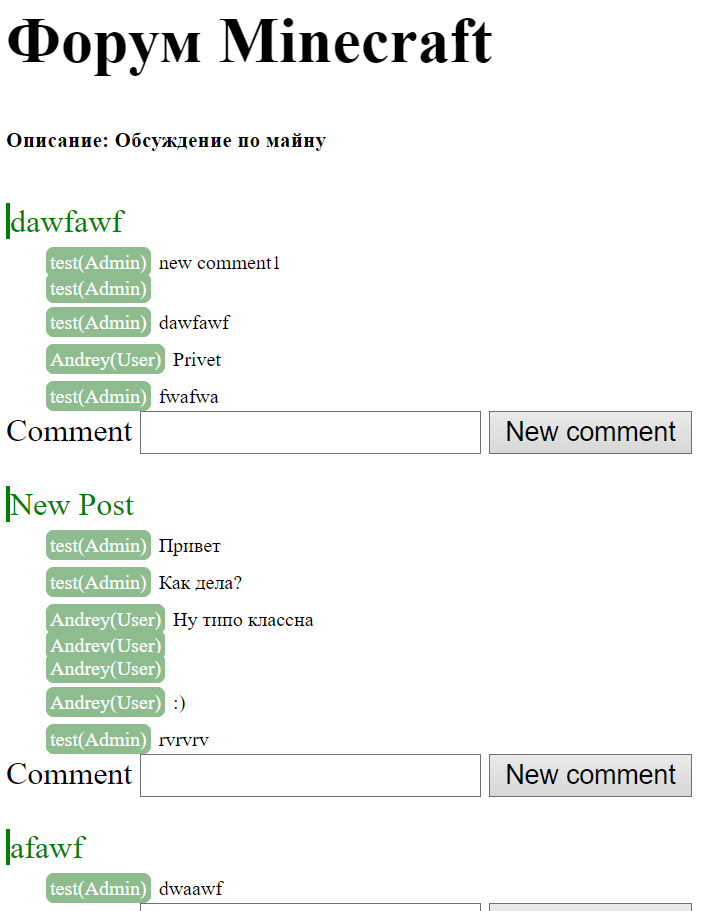


Рисунок – Страница форума, добавление поста, добавление комментарий

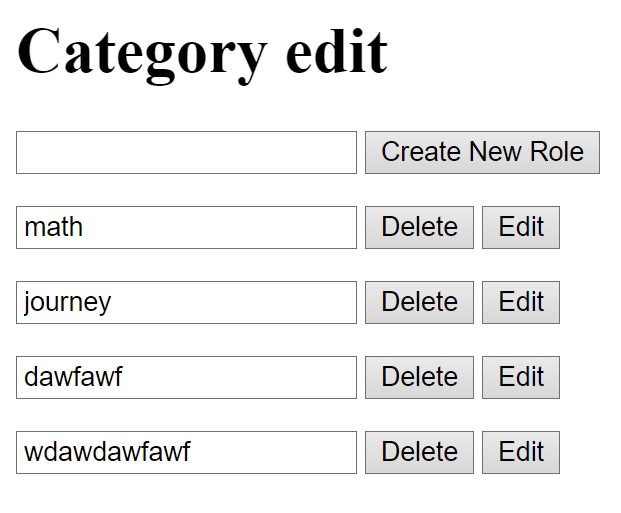


Рисунок – Страница редактирования категорий

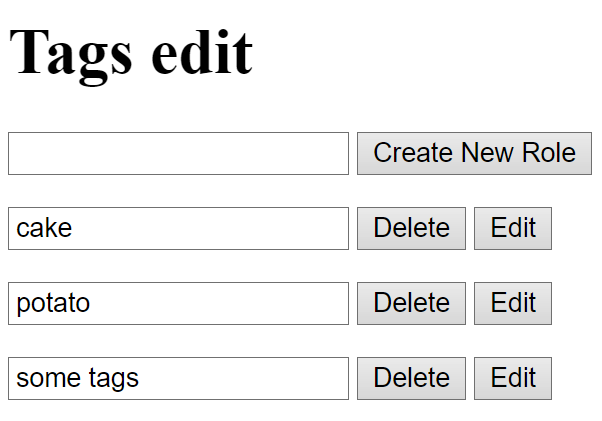


Рисунок – Страница редактирования тэгов

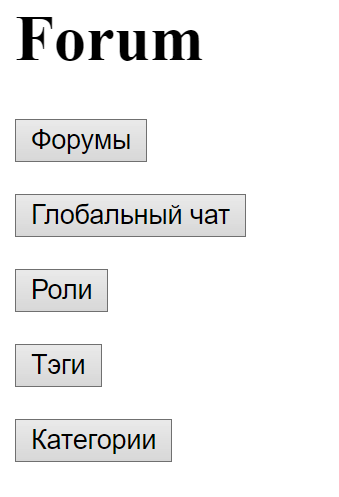


Рисунок – Главная страница

# ВЫВОД

В рамках завершенного проекта форума, он продемонстрировал высочайший уровень профессионализма и глубокие знания в современных технологиях. Использование Kotlin и Spring Framework позволило создать мощное и эффективное приложение, которое обеспечивает отличную производительность и простоту поддержки. Применение JPA и Hibernate для работы с базой данных обеспечило возможность реализации сложных запросов и операций с данными, при этом гарантируя их целостность и надежность.

Интеграция Spring Security гарантировала многоуровневую систему защиты приложения, включая аутентификацию и авторизацию, что критически важно для форума с безопасным управлением аккаунтами и контентом пользователями. Использование Thymeleaf для серверного рендеринга страниц позволило создать динамичные и взаимодействующие веб-страницы, которые легко масштабируются и адаптируются к различным устройствам и размерам экранов.

Все компоненты системы были тщательно спроектированы и оптимизированы, что привело к высокой скорости работы и отзывчивости приложения. Его внимание к деталям, чистота кода и следование принципам SOLID и лучшим практикам программирования подтверждают его глубокие технические знания и стремление к совершенству.

Завершив этот проект, он не только укрепил свои навыки в области бэкенд-разработки, но и продемонстрировал способность к созданию современных веб-приложений, соответствующих самым высоким стандартам качества и безопасности. Этот проект стал отражением его способности к инновациям и решению сложных задач в области веб-разработки.

Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский экономический университет имени Г.В. Плеханова»

**МОСКОВСКИЙ ПРИБОРОСТРОИТЕЛЬНЫЙ ТЕХНИКУМ**

специальность 09.02.07 «Информационные системы и программирование»

Квалификация: Программист

ПРАКТИЧЕСКИЕ РАБОТЫ

УП 04.01 ВППО

|  |  |
| --- | --- |
| Выполнил студент  группы П50-6-20  Баранов Андрей Викторович | Проверил преподаватель  \_\_\_\_\_\_\_\_\_\_\_ Д.В. Серяк  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2023 года |

Москва 2023

ОГЛАВЛЕНИЕ

[ПРАКТИЧЕСКАЯ РАБОТА №1 3](#_Toc154545059)

[Калькулятор и конвертер валют 3](#_Toc154545060)

[ПРАКТИЧЕСКАЯ РАБОТА №2 10](#_Toc154545061)

[Работа с паттерном DAO 10](#_Toc154545062)

[ПРАКТИЧЕСКАЯ РАБОТА №3 14](#_Toc154545063)

[Работа с JPA и Validator 14](#_Toc154545064)

[ПРАКТИЧЕСКАЯ РАБОТА №4 18](#_Toc154545065)

[Связи 18](#_Toc154545066)

[ПРАКТИЧЕСКАЯ РАБОТА №5 21](#_Toc154545067)

[Авторизация и регистрация 21](#_Toc154545068)

[ПРАКТИЧЕСКАЯ РАБОТА №6 26](#_Toc154545069)

[Разделение прав доступа 26](#_Toc154545070)

# ПРАКТИЧЕСКАЯ РАБОТА №1

## Калькулятор и конвертер валют

Цель работы: создать веб-приложение с калькулятором и конвертером валют на Spring Boot.

1. Создание проекта

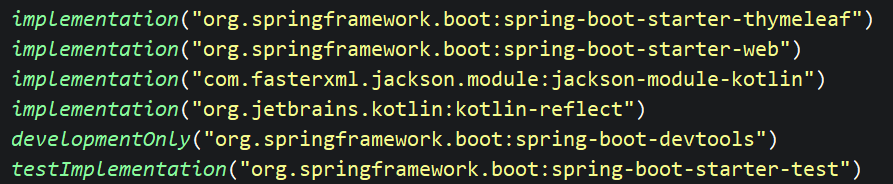


Рисунок 1 – Подключение зависимостей

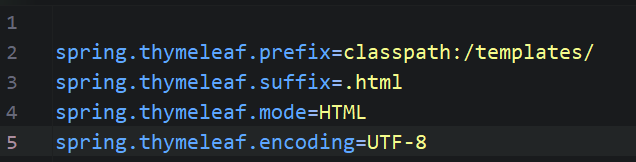


Рисунок 2 – Конфигурация Thymeleaf

1. Главный сайт

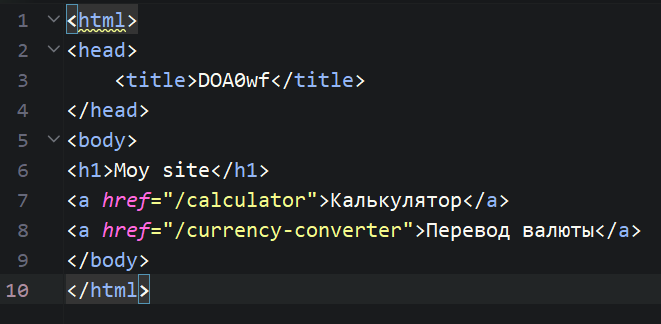


Рисунок 3 – Index.html

1. Конвертер, через form идёт запрос http в Spring.

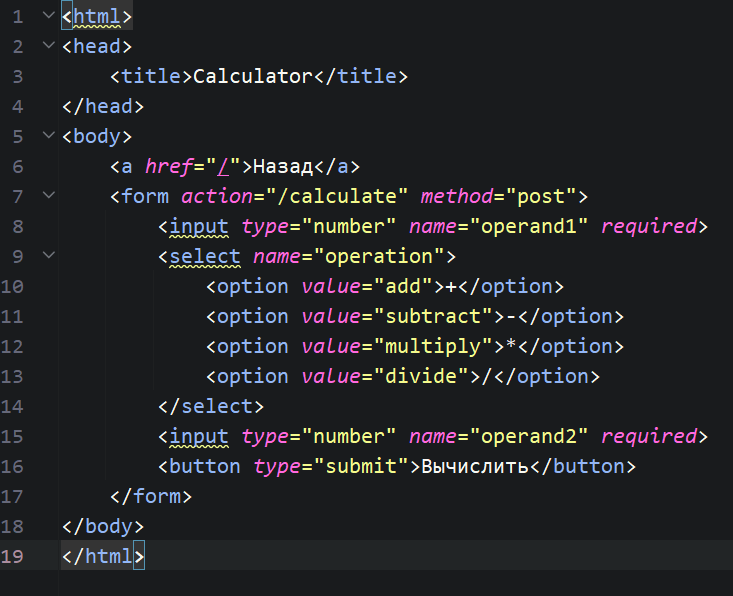


Рисунок 4 – Конвертер валюты

1. Конвертер контроллер, получение страницы, и вычисление конвертации.

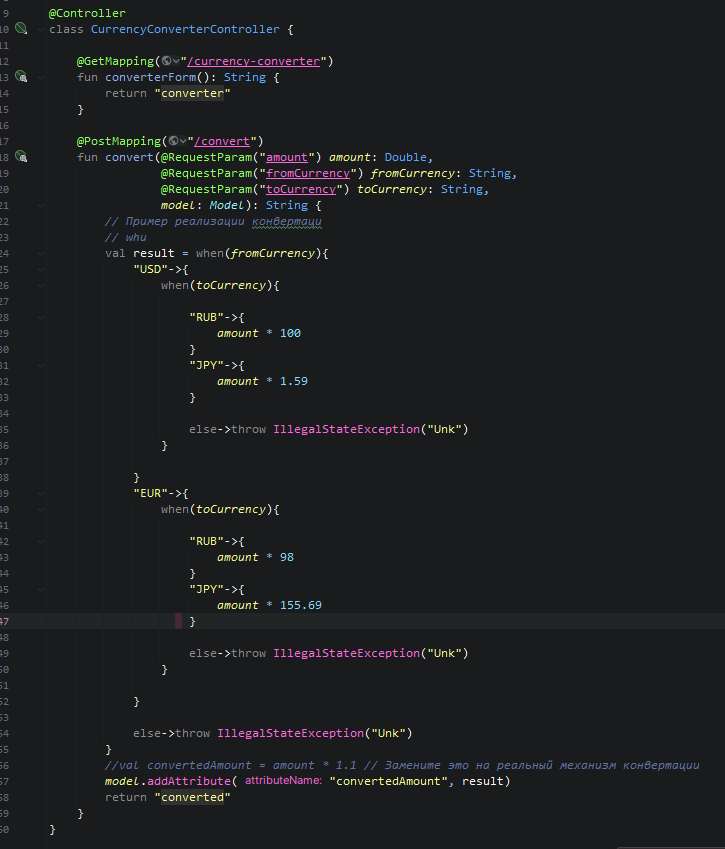


Рисунок 5 – Контроллер для валюты

1. Калькулятор он прибавляет складывает, минусует, значения и так далее.



Рисунок 6 – Страница калькулятора



Рисунок 7 – Контроллер калькулятора

1. Результат работы

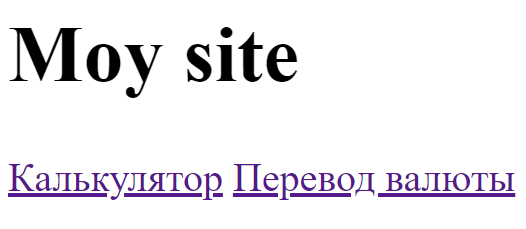


Рисунок 8 – Главная страница

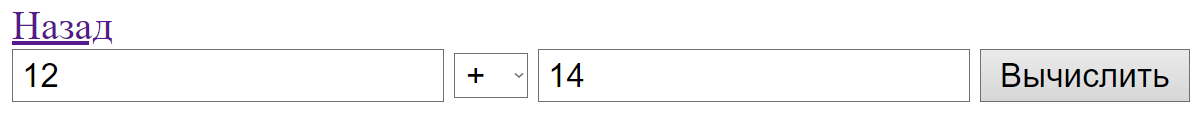


Рисунок 9 – Калькулятор

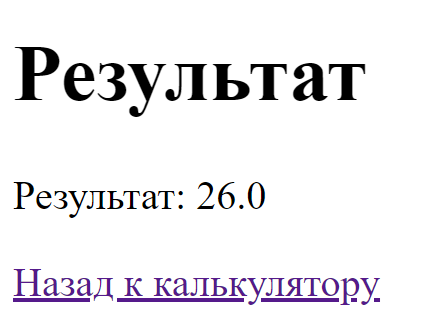


Рисунок 10 – Результат

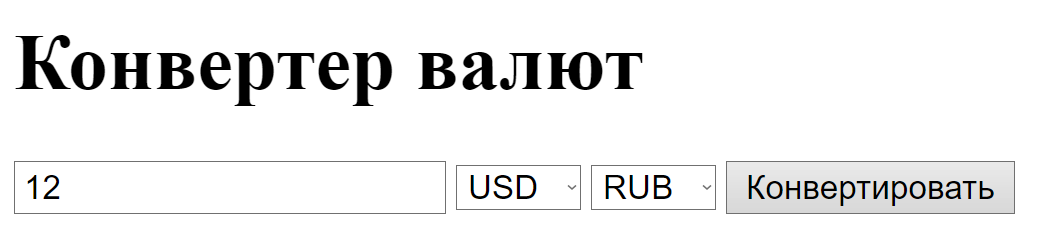


Рисунок 11 – Конвертер валют

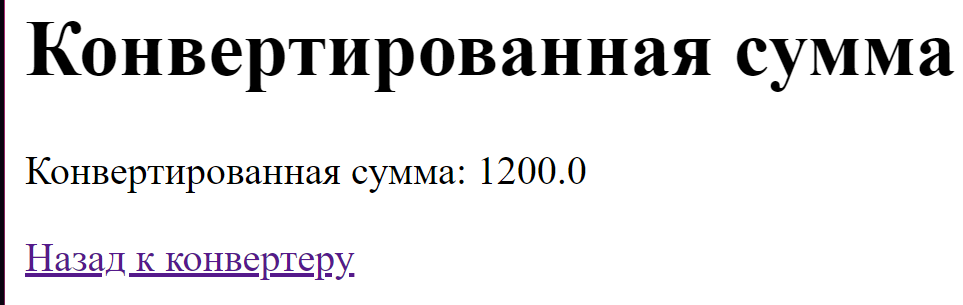


Рисунок 12 – Конвертированная сумма

Вывод: Приложение демонстрирует использование Spring MVC для обработки запросов и динамического отображения результатов.

# ПРАКТИЧЕСКАЯ РАБОТА №2

## Работа с паттерном DAO

Цель работы: разработка паттерна Data Access Object (DAO) для управления данными, создание моделей и контроллеров для взаимодействия с этими данными, а также разработка шаблонов страниц, которые будут загружаться через Endpoints

1. Все модели будут имплементировать этот интерфейс.

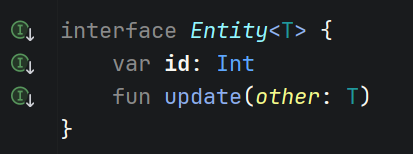


Рисунок 13 - Интерфейс

1. Импровизация бд с дженериком по интерфейсу.

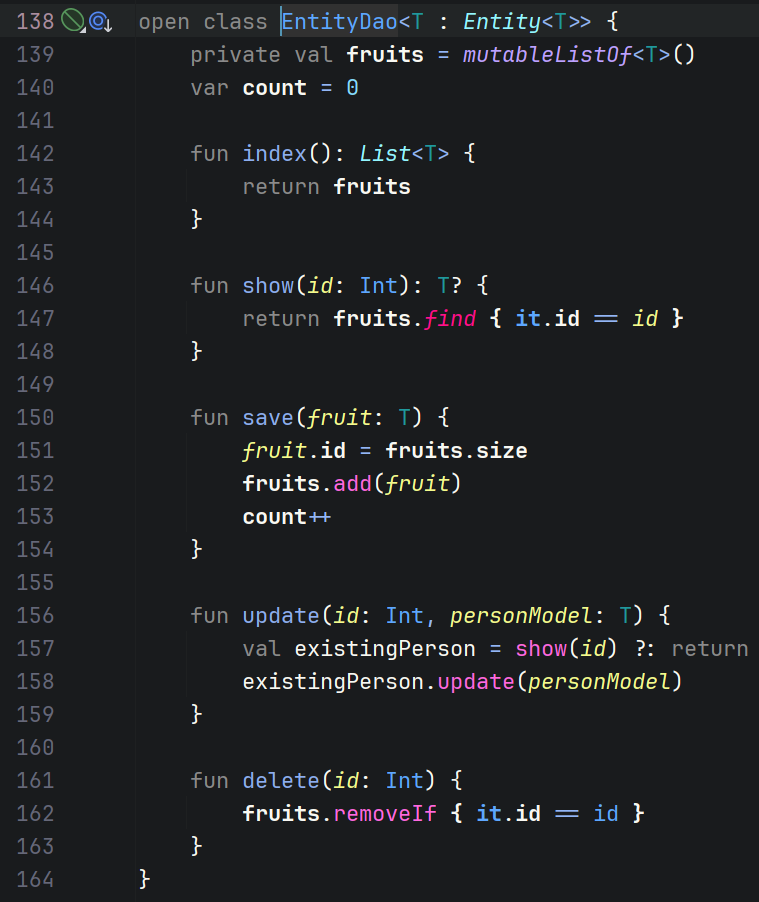


Рисунок 14 – GenericDAO

1. Создание моделей



Рисунок 15 - Модели

1. Заполняем данными, и @Component нужен для создания bean, для Dependency Injection.



Рисунок 16 – BookDAO

1. Создание контроллера, который подключает к себе компонент DaoBook, индекс для загрузки страницы, отображение, добавление, изменение, удаление.



Рисунок 17 – Контроллер Book

1. Пишем много HTML, подключаем туда Bootstrap для стилей. Проделываем так для каждого Entity.

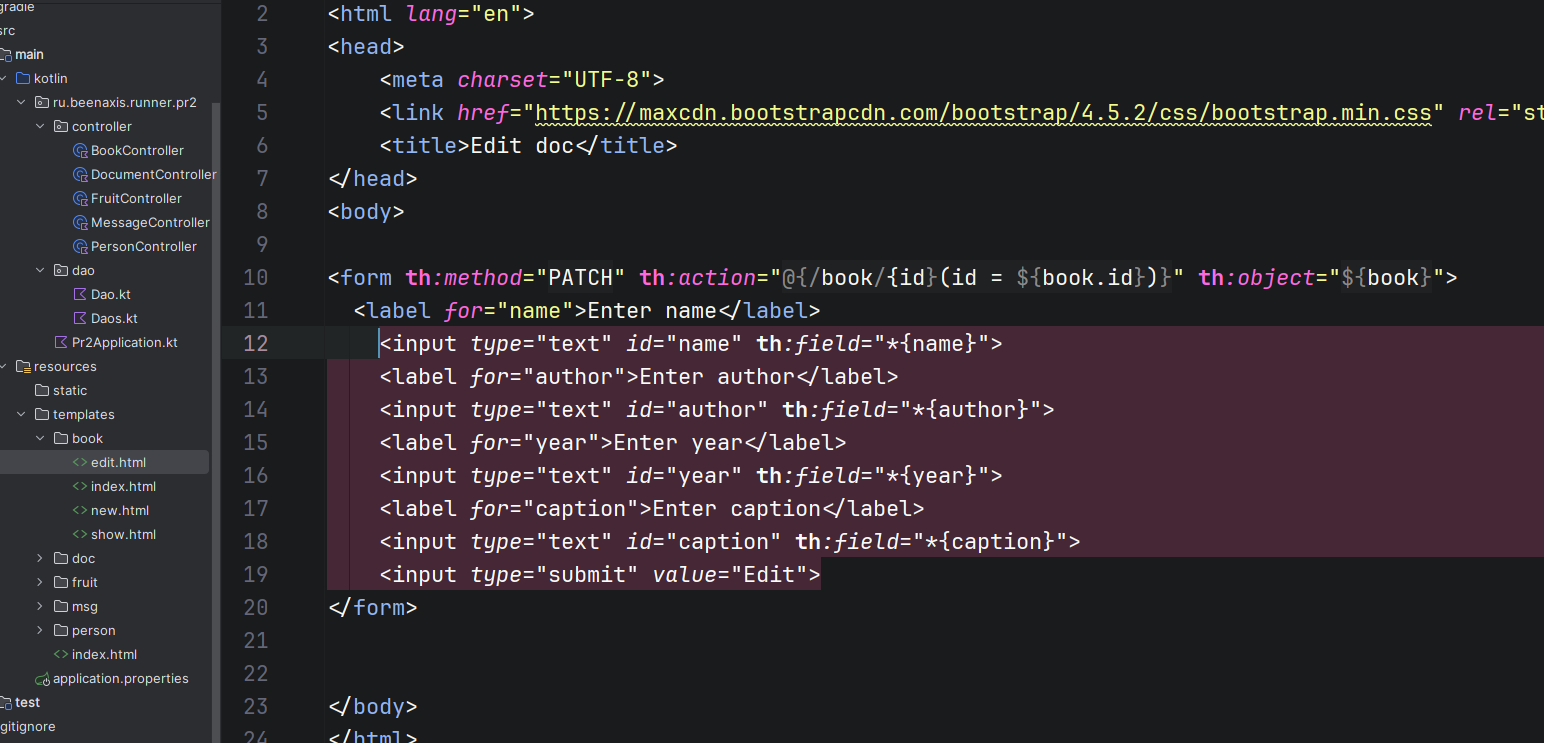


Рисунок 18 – Html страница Edit

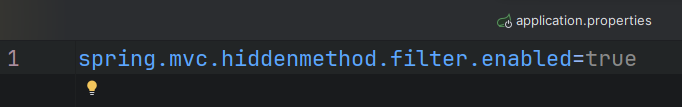


Рисунок 19 – Если это не добавить, не будет ничего работать в MVC.

Вывод: В ходе практической был разработан паттерн DAO, созданы модели, контроллеры, и шаблоны страниц для загрузки из Endpoints.

# ПРАКТИЧЕСКАЯ РАБОТА №3

## Работа с JPA и Validator

Цель работы: Освоение принципов создания и управления сущностями в рамках Java Persistence API (JPA) и Validator, а также разработка контроллеров для взаимодействия с базой данных и реализация функциональности простого поиска в приложении.

1. Конфигурация подключения к БД Postgres.

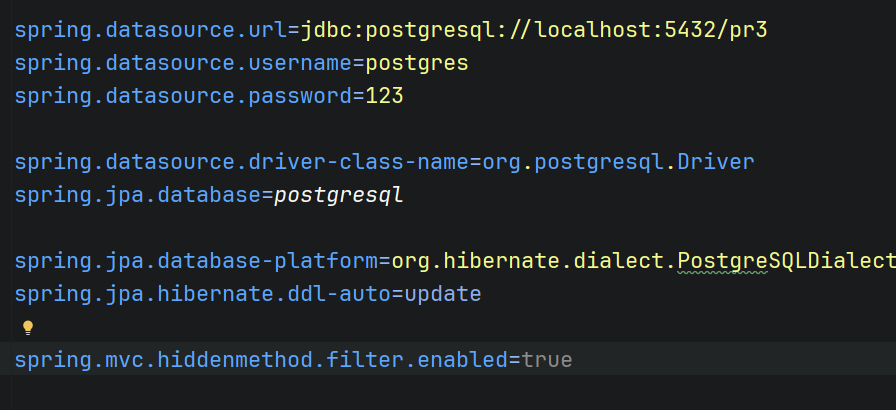


Рисунок 20 – Конфиг спринга

1. Подключение зависимостей и драйвера постгрес.

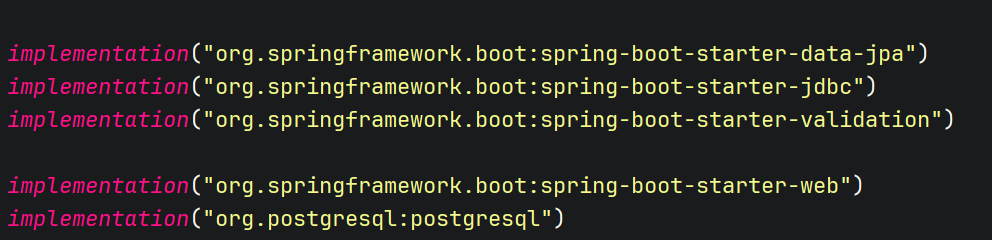


Рисунок 21 – Подключение зависимостей

1. Использование аннотаций Id – поле с айди, @Entity – обозначает что этот класс является сущностью, @Table – определяет название таблицы. @Column – параметры столбца. @Size – размеры строки.

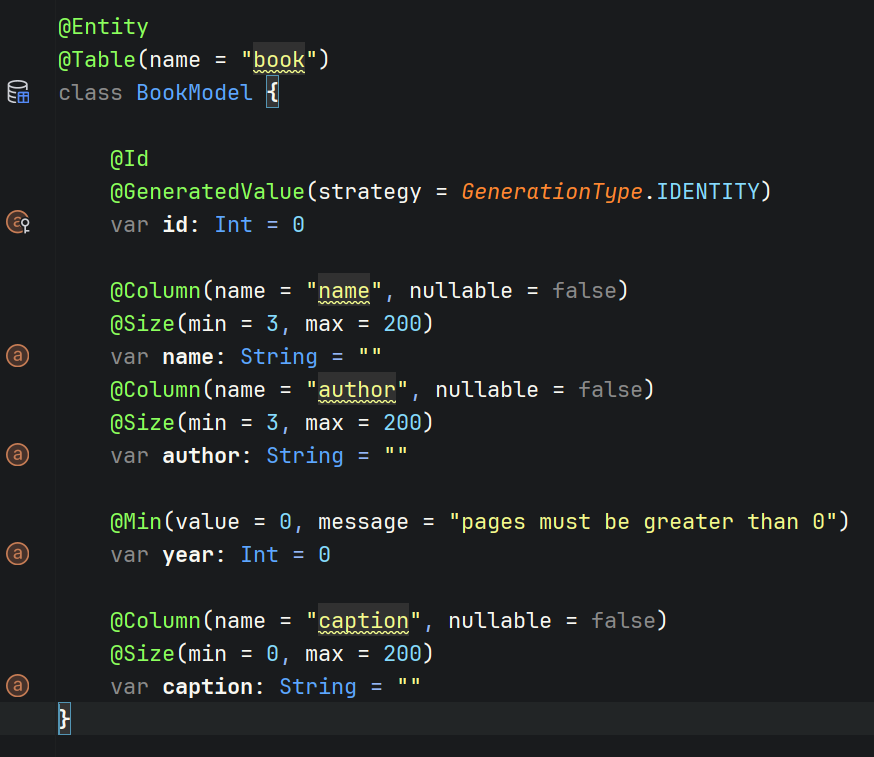


Рисунок 22 – Аннотации JPA и Validator

1. MinMax, означает минимальное и максимально возможное значение поля.

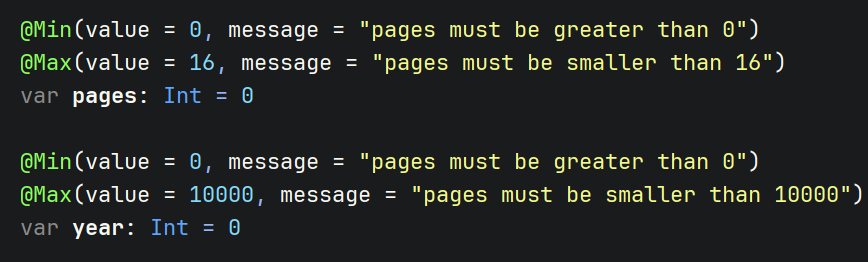


Рисунок 23 – MinMax Аннотации

1. Аннотации Positive означает что в этом поле всегда будет положительное число

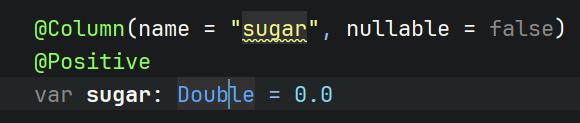


Рисунок 24 - @Positive

1. Создание прокси репозитория, добавление прокси функции, которую сгенерирует Spring.

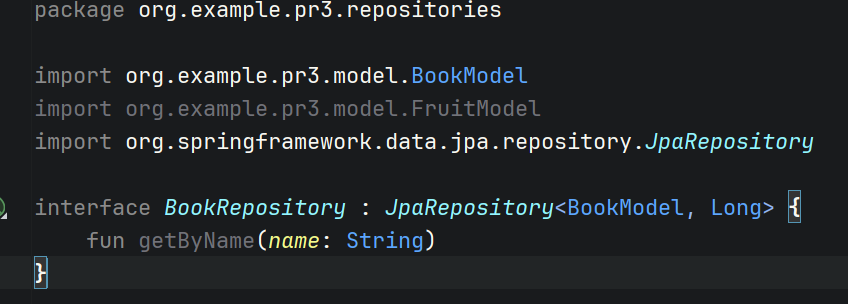


Рисунок 25 – BookRepository

1. Страница для поиска, через форму отправляется запрос.

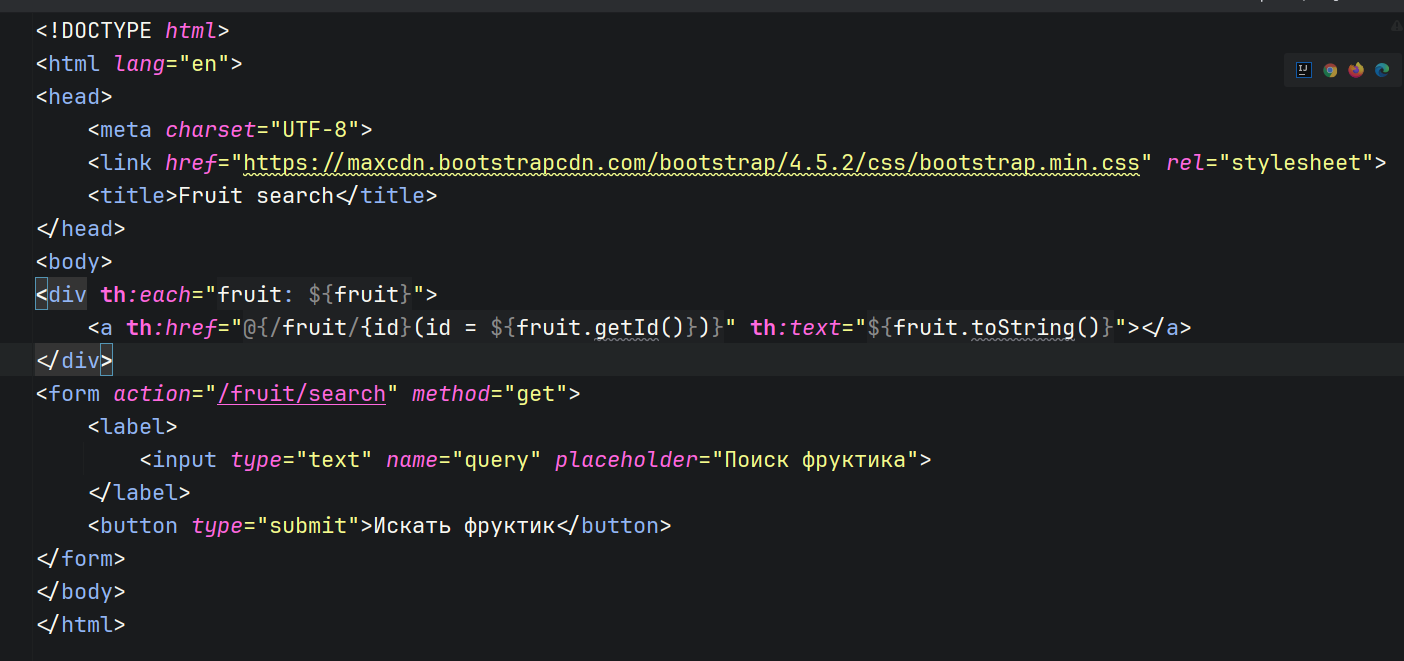


Рисунок 26 – Поиск

1. Поиск с вызовом getByName, который вернёт по имени сущность.



Рисунок 27 – Реализация поиска

Вывод: В ходе практической работы был разработаны сущности через JPA & Validator, созданы контроллеры для модификации их в БД, подключена база данных, и разработан простейший поиск.

# ПРАКТИЧЕСКАЯ РАБОТА №4

## Связи

Цель работы: добавление и настройка различных типов связей между сущностями в модели данных, с целью определения и организации полной структуры базы данных.

1. OneToOne – один к одному, @ManyToOne – многие к одному, @ManyToMany – МКМ. @JoinColumn значит что по айди сущности будет происходить заполнени этого поля. Каскад означает какие поля будут добавляться, изменятся удаляться.



Рисунок 28 – Аннотации связей

1. Для связи МКМ, joinColumns – Указывает колонку в промежуточной таблице, которая будет хранить идентификаторы сущности. inverseJoinColumns - Указывает колонку в промежуточной таблице, которая будет хранить идентификаторы сущности

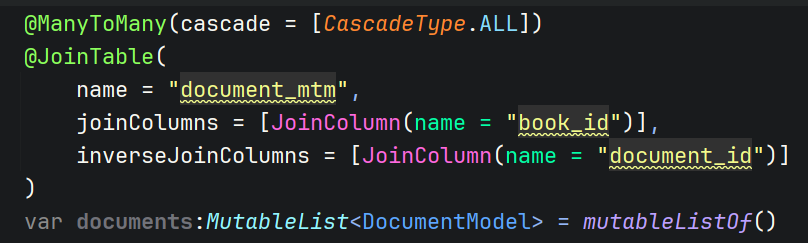


Рисунок 29 – МКМ

1. В книгах ManyToOne writer будет иметь множество книг в сущности PersonModel, где это определяется в виде списка, а в книге в виде одной сущности.

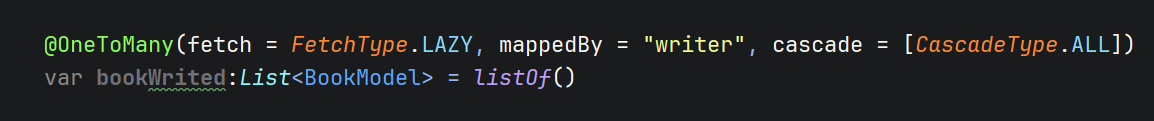


Рисунок 30 – OneToMany и ManyToOne

1. В DataGrip построим диаграмму для визуализации связей.

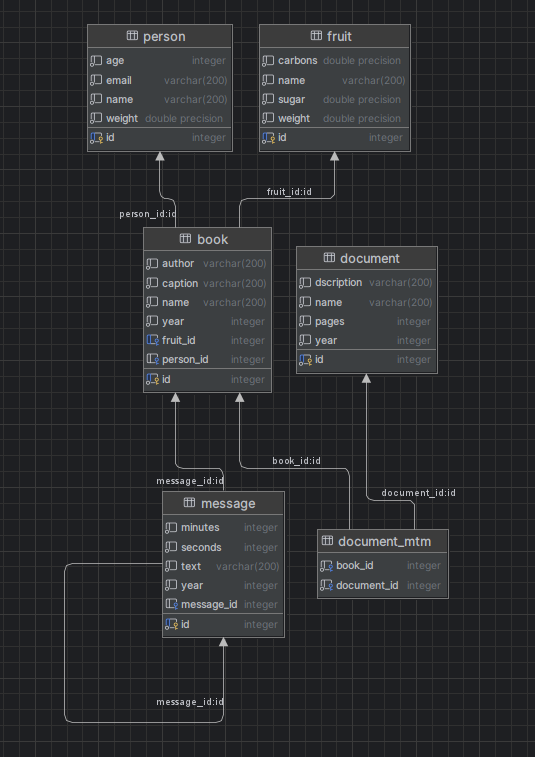


Рисунок 31 – Результат работы

Вывод: В ходе практической работы в модели сущностей были добавлены различные связи, для определения структуры всей БД.

# ПРАКТИЧЕСКАЯ РАБОТА №5

## Авторизация и регистрация

Цель работы: Добавить регистрацию и авторизацию в свой проект.

1. Подключение спринг security, по умолчанию он заблокирует все пути к эндпоинтам.



Рисунок 32 – Подключение Spring Security

1. Создаём модель юзера, и наследуем от UserDetails интерфейса чтобы указать что это авторизируемый класс.



Рисунок 33 – Модель юзера

1. Конфигурация доступов ко всем путям



Рисунок 34 Пути

1. Добавляем репозиторий User

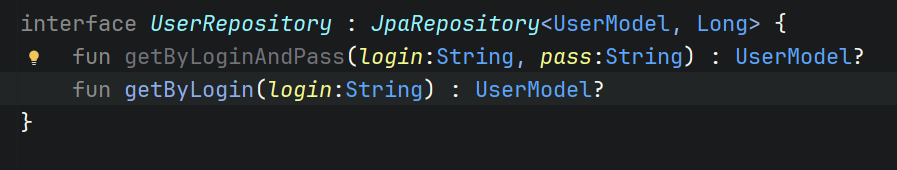


Рисунок 35 – UserRepository

1. Создание контроллера для регистрации

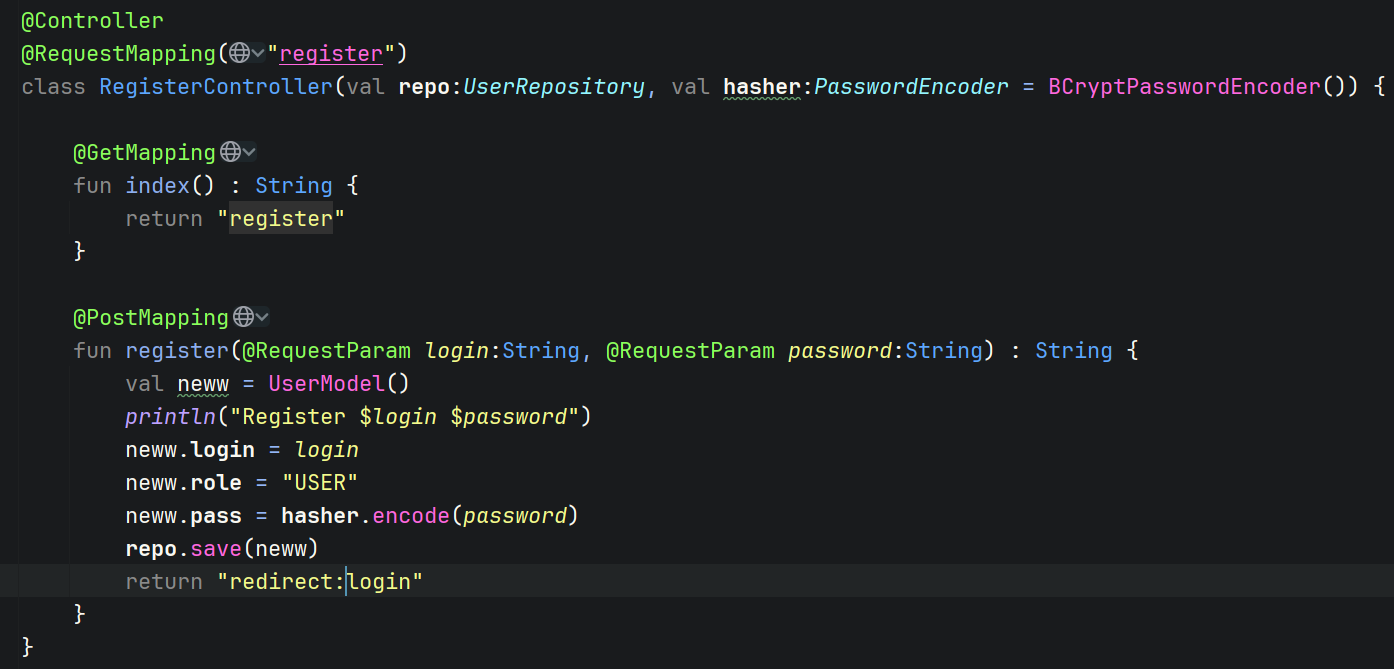


Рисунок 36 – Регистрация

1. Определение шаблона страницы регистрации

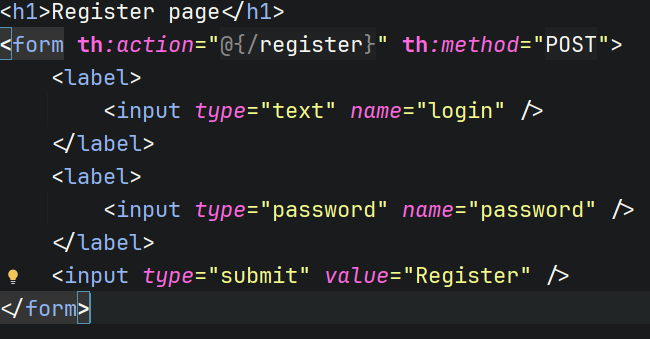


Рисунок 37 – Шаблон регистрации

1. Добавление кнопки logout



Рисунок 38 – Logout

1. Регистрируемся через /register



Рисунок 39 – Register Page

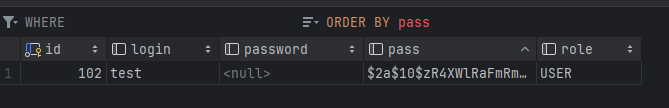


Рисунок 40 – User в БД

1. После успешной регистрации, мы попадаем уже на авторизованную страницу. Если заходить сюда без авторизации то перекинет на страницу логина.

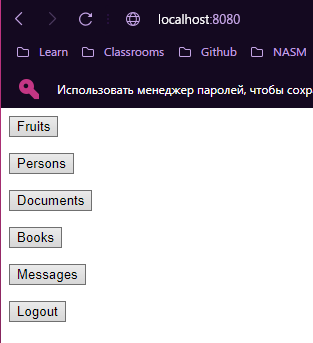


Рисунок 41 – Главная

1. Тем самым у нас теперь есть доступ ко всему почти.

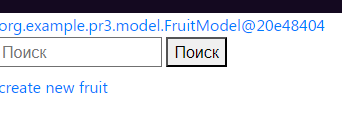


Рисунок 42 – Авторизованный

1. Если зайти в msg, доступ запрещён

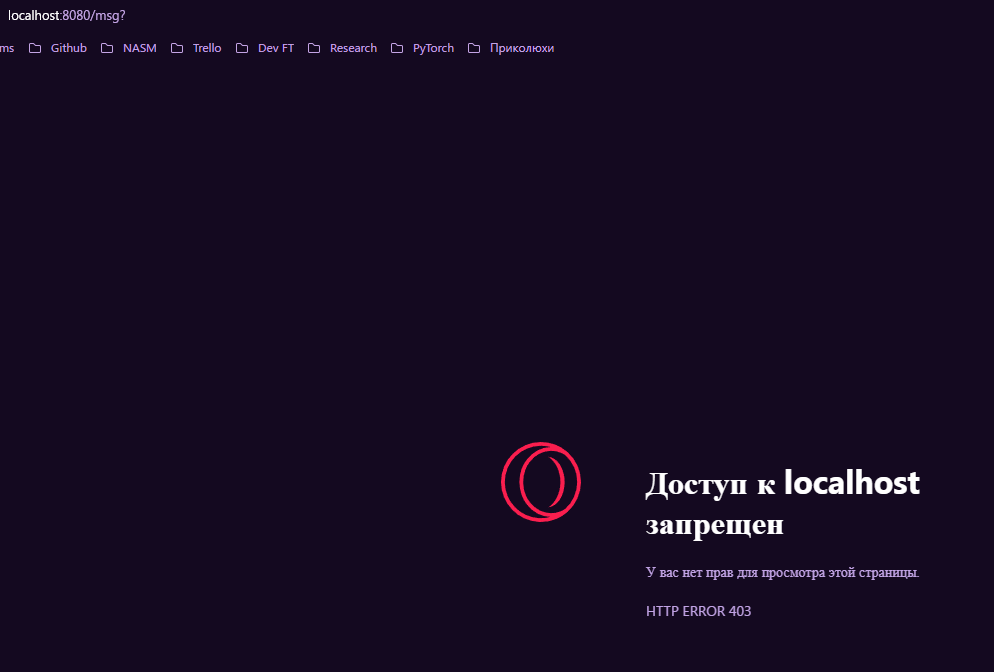


Рисунок 43 - /msg

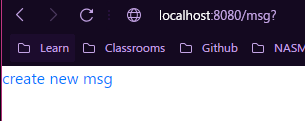


Рисунок 44 - /msg с ролью ADMIN

1. Изменяем роль на юзера, и у нас снова нет доступа к ADMIN.



Вывод: В ходе практической работы была разработана авторизация и регистрация, а так же вход выход, и роли.

# ПРАКТИЧЕСКАЯ РАБОТА №6

## Разделение прав доступа

Цель работы: Реализовать механизм шифрование пароля пользователя. Добавить разграничение прав доступа для пользователей.

1. Механизм шифрования, везде где будет @Autowired с PasswordEncoder – будет браться этот объект в методе. BCryptPasswordEncoder – хэширует пароли.

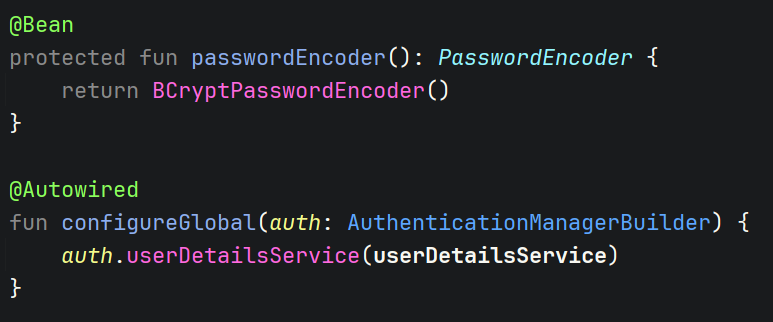


Рисунок 45 – Bean Шифрования

1. Реализация юзер дитейлс, для того чтобы определить юзеров в авторизации. Добавляем новую роль ADMIN



Рисунок 46 – Авторизация

1. Шифрование на пароль в юзере.

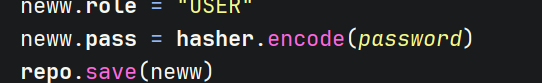
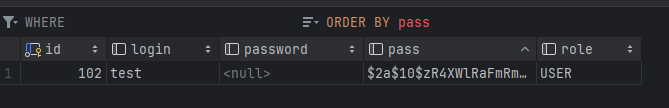


Рисунок 47 – Регистрация с шифрованием

1. В БД строка с паролём зашифрована.



1. Задействуем роль в путях.

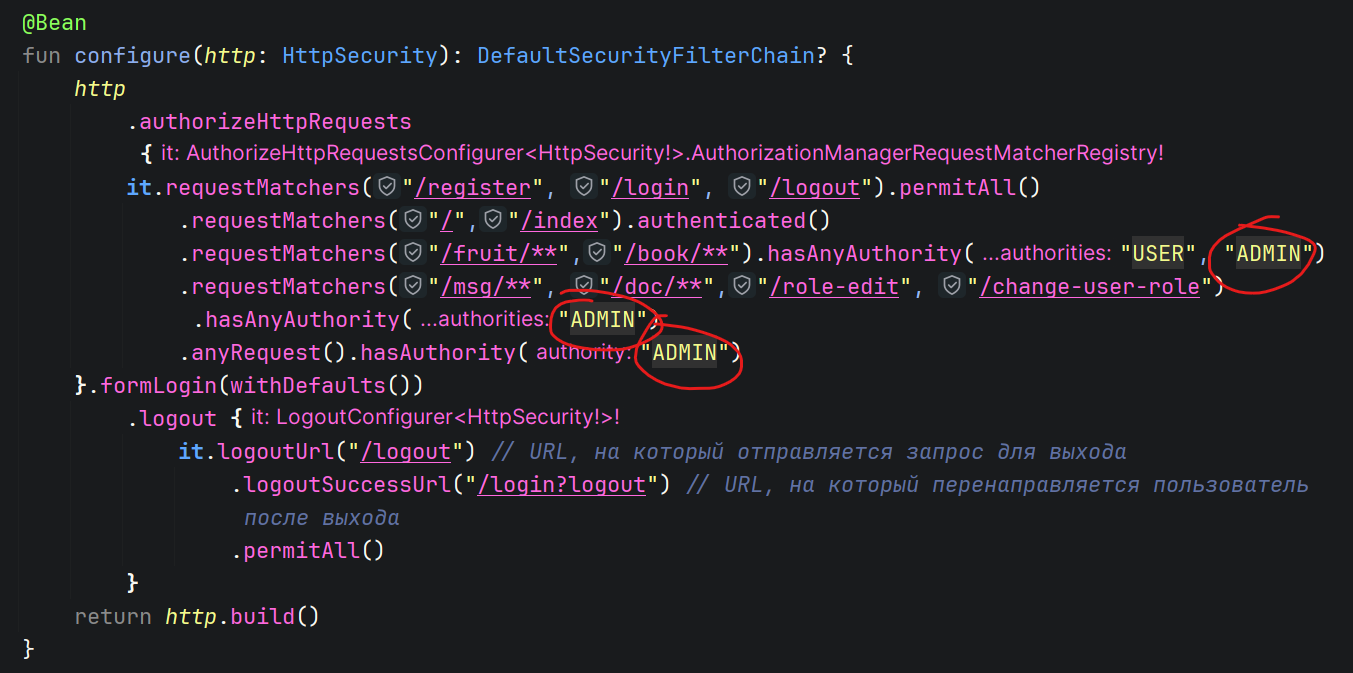


Рисунок 48 – Новая роль и права

1. Создаём контроллер, где будет гет и пост для изменений.



Рисунок 49 – Get

1. Смена роли, получаем юзер из БД, меняем роль, и сохраняем.

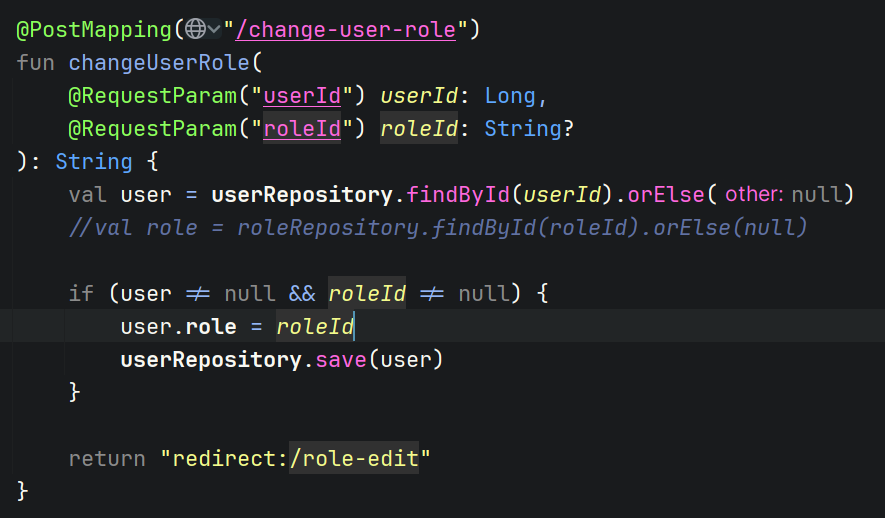


Рисунок 50 – Смена роли

1. При входе с USER на /msg, выдает запрет доступа.

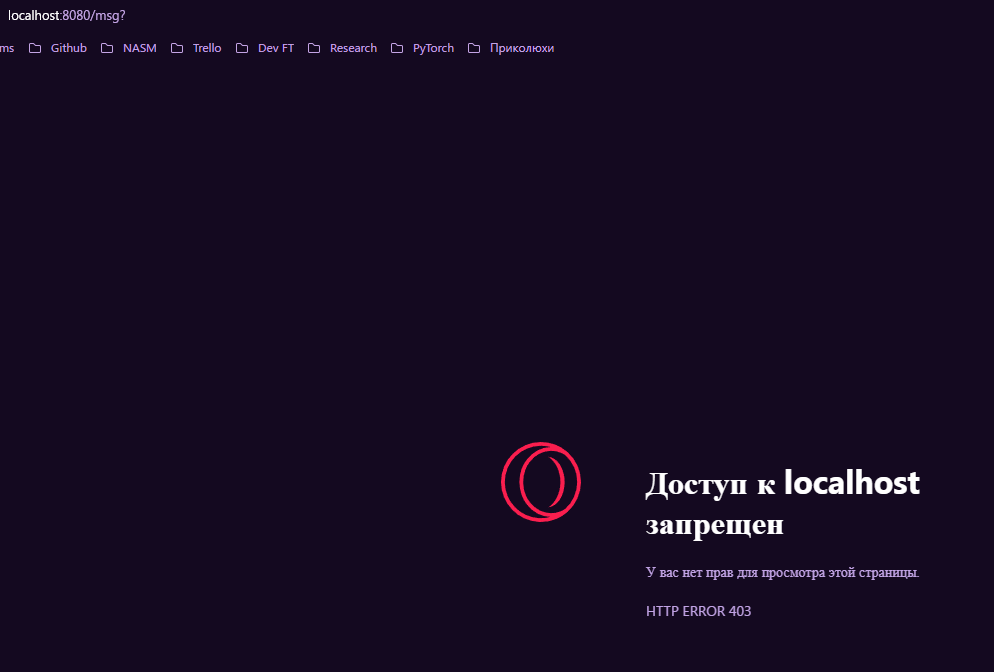


Рисунок 51 – Доступ запрещён (USER)

1. При входе с ADMIN (изменения из БД), доступ есть.

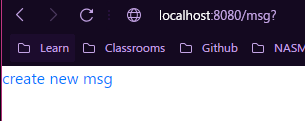


Рисунок 52 – Доступ разрешён (ADMIN)

1. При изменении роли снова на USER, доступ снова запретиться.



Рисунок 53 – Страница изменения роли

Вывод: В ходе практической работы был разработана новая роль, шифрование пароля, и разделение доступа по ролям.