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
2. ТЕМА 4
3. ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5
4. СХЕМА БАЗЫ ДАННЫХ 6
5. СЛОВАРЬ ДАННЫХ 8
6. СКРИПТ БАЗЫ ДАННЫХ 9
7. КОД ПРОГРАММЫ 11
8. ДЕМОНСТРАЦИЯ ПРОГРАММЫ 36
9. ВЫВОД 42

1. СОДЕРЖАНИЕ

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

2. ТЕМА

Индивидуальные проект написан на тему «Форум».

3. 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.

4. СХЕМА БАЗЫ ДАННЫХ

4.1. Логическая схема базы данных

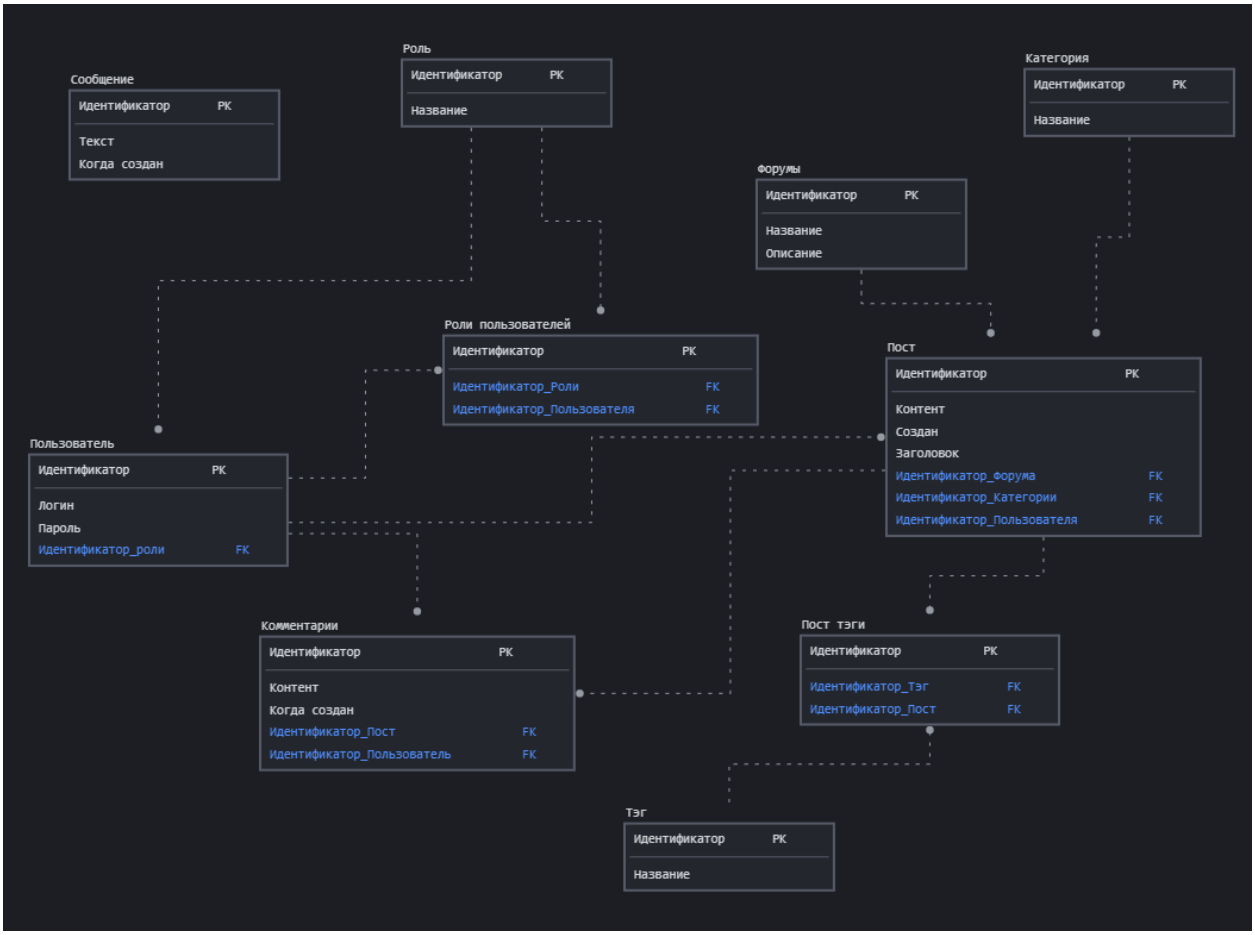


Рисунок 1 – Логическая схема данных

4.2. Физическая схема базы данных

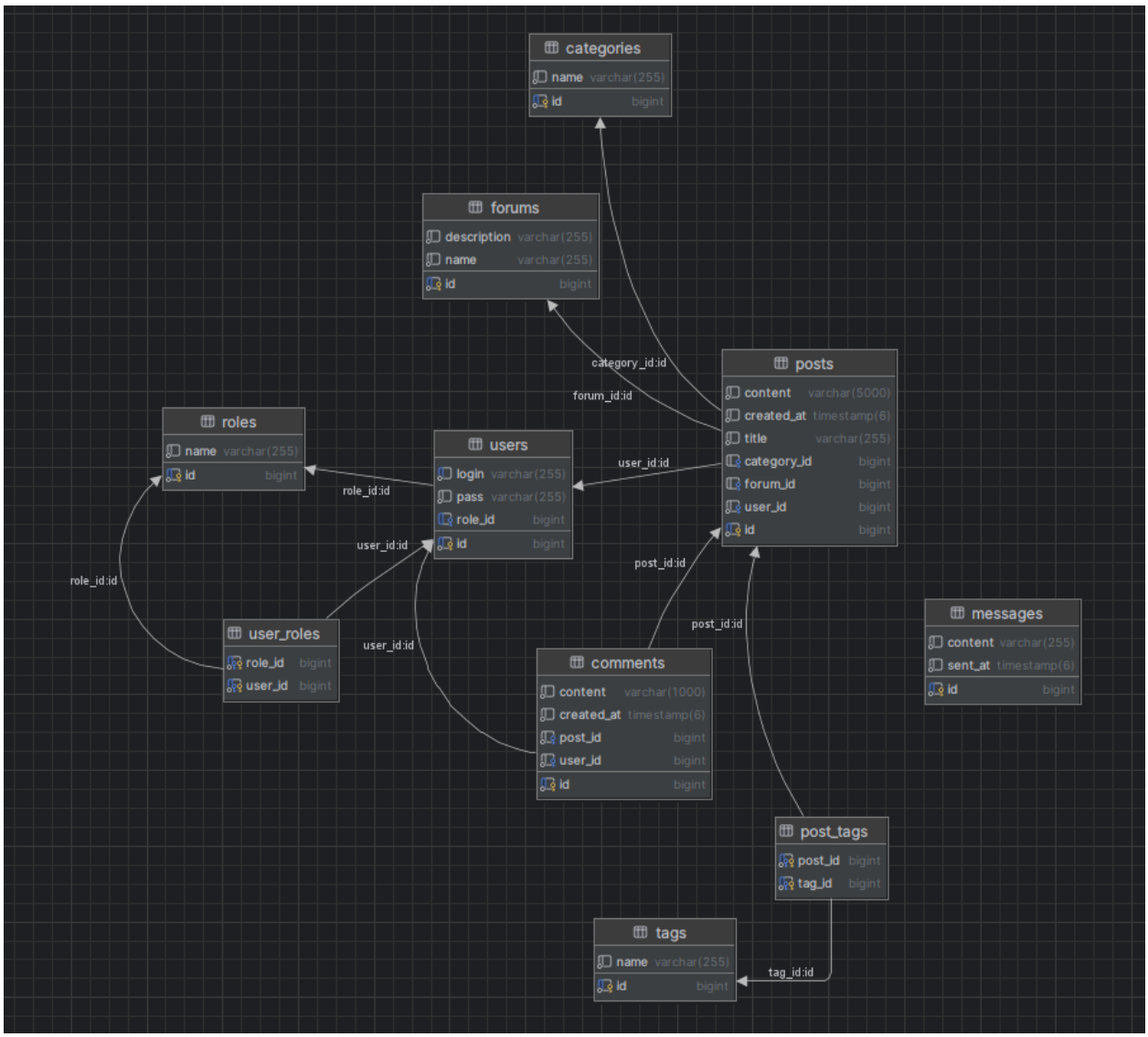


Рисунок 2 – Физическая схема данных

5. СЛОВАРЬ ДАННЫХ

Таблица 1 – 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

6. 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;
```

7. 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() {
    }

}
```

8. ДЕМОНСТРАЦИЯ ПРОГРАММЫ

Рисунок 3 – Страница авторизации

Рисунок 4 – Страница регистрации

Edit User Role

test | Select Role | Change Role
Select Role
Admin
User
Create New Role
Admin | Delete | Edit
User | Delete | Edit

Рисунок 5 – Страница управления ролями

Forums

Minecraft
Dota 2

Create new forum

Create forum

Рисунок 6 – Страница всех форумов, Создание форума

Форум Minecraft

Описание: Обсуждение по майну

dawfawf

test(Admin) new comment1
test(Admin)
test(Admin) dawfawf
Andrey(User) Privet
test(Admin) fwafwa

Comment New comment

New Post

test(Admin) Привет
test(Admin) Как дела?
Andrey(User) Ну типо классна
Andrey(User)
Andrey(User)
Andrey(User) :)
test(Admin) rvrvrv

Comment New comment

afawf

test(Admin) dwaawf

Рисунок 7 – Страница форума, добавление поста, добавление комментарий

Category edit

Create New Role

math Delete Edit

journey Delete Edit

dawfawf Delete Edit

wdawdawfawf Delete Edit

Рисунок 8 – Страница редактирования категорий

Tags edit

Create New Role

cake Delete Edit

potato Delete Edit

some tags Delete Edit

Рисунок 9 – Страница редактирования тэгов

Forum

Форумы

Глобальный чат

Роли

Тэги

Категории

Рисунок 10 – Главная страница

9. 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	Проверил
группы П50-6-20	преподаватель
Баранов Андрей Викторович	_____ Д.В. Серяк
	«___» _____ 2023 года

Москва 2023

ОГЛАВЛЕНИЕ

ПРАКТИЧЕСКАЯ РАБОТА №1 46

Калькулятор и конвертер валют 46

ПРАКТИЧЕСКАЯ РАБОТА №2 53

Работа с паттерном DAO 53

ПРАКТИЧЕСКАЯ РАБОТА №3 58

Работа с JPA и Validator 58

ПРАКТИЧЕСКАЯ РАБОТА №4 62

Связи 62

ПРАКТИЧЕСКАЯ РАБОТА №5 65

Авторизация и регистрация 65

ПРАКТИЧЕСКАЯ РАБОТА №6 70

Разделение прав доступа 70

ПРАКТИЧЕСКАЯ РАБОТА №1

Калькулятор и конвертер валют

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

1) Создание проекта

```
implementation("org.springframework.boot:spring-boot-starter-thymeleaf")
implementation("org.springframework.boot:spring-boot-starter-web")
implementation("com.fasterxml.jackson.module:jackson-module-kotlin")
implementation("org.jetbrains.kotlin:kotlin-reflect")
developmentOnly("org.springframework.boot:spring-boot-devtools")
testImplementation("org.springframework.boot:spring-boot-starter-test")
```

Рисунок 11 – Подключение зависимостей

```
1
2  spring.thymeleaf.prefix=classpath:/templates/
3  spring.thymeleaf.suffix=.html
4  spring.thymeleaf.mode=HTML
5  spring.thymeleaf.encoding=UTF-8
```

Рисунок 12 – Конфигурация Thymeleaf

2) Главный сайт

```
1  <html>
2  <head>
3      <title>DOA0wf</title>
4  </head>
5  <body>
6  <h1>Moy site</h1>
7  <a href="/calculator">Калькулятор</a>
8  <a href="/currency-converter">Перевод валюты</a>
9  </body>
10 </html>
```

Рисунок 13 – Index.html

3) Конвертер, через form идёт запрос http в Spring.

```html
<html>
<head>
    <title>Calculator</title>
</head>
<body>
    <a href="/">Назад</a>
    <form action="/calculate" method="post">
        <input type="number" name="operand1" required>
        <select name="operation">
            <option value="add">+</option>
            <option value="subtract">-</option>
            <option value="multiply">*</option>
            <option value="divide">/</option>
        </select>
        <input type="number" name="operand2" required>
        <button type="submit">Вычислить</button>
    </form>
</body>
</html>
```

Рисунок 14 – Конвертер валюты

4) Конвертер контроллер, получение страницы, и вычисление конвертации.

```
@Controller
class CurrencyConverterController {

    @GetMapping("/currency-converter")
    fun converterForm(): String {
        return "converter"
    }

    @PostMapping("/convert")
    fun convert(@RequestParam("amount") amount: Double,
                @RequestParam("fromCurrency") fromCurrency: String,
                @RequestParam("toCurrency") toCurrency: String,
                model: Model): String {
        // Пример реализации конвертаци
        // whu
        val result = when(fromCurrency){
            "USD"->{
                when(toCurrency){

                    "RUB"->{
                        amount * 100
                    }
                    "JPY"->{
                        amount * 1.59
                    }

                    else->throw IllegalStateException("Unk")
                }

            }
            "EUR"->{
                when(toCurrency){

                    "RUB"->{
                        amount * 98
                    }
                    "JPY"->{
                        amount * 155.69
                    }

                    else->throw IllegalStateException("Unk")
                }

            }

            else->throw IllegalStateException("Unk")
        }
        //val convertedAmount = amount * 1.1 // Замените это на реальный механизм конвертации
        model.addAttribute( attributeName: "convertedAmount", result)
        return "converted"
    }
}
```

Рисунок 15 – Контроллер для валюты

5) Калькулятор он прибавляет складывает, минусует, значения и так далее.

```html
<html>
<head>
    <title>Calculator</title>
</head>
<body>
    <a href="/">Назад</a>
    <form action="/calculate" method="post">
        <input type="number" name="operand1" required>
        <select name="operation">
            <option value="add">+</option>
            <option value="subtract">-</option>
            <option value="multiply">*</option>
            <option value="divide">/</option>
        </select>
        <input type="number" name="operand2" required>
        <button type="submit">Вычислить</button>
    </form>
</body>
</html>
```

Рисунок 16 – Страница калькулятора

```
8
9   @Controller
10  class CalculatorController {
11
12      @GetMapping("/calculator")
13      fun calculatorForm(): String {
14          return "calculator"
15      }
16
17      @PostMapping("/calculate")
18      fun calculate(@RequestParam("operand1") operand1: Double,
19                    @RequestParam("operand2") operand2: Double,
20                    @RequestParam("operation") operation: String,
21                    model: Model
22      ): String {
23          val result = when (operation) {
24              "add" -> operand1 + operand2
25              "subtract" -> operand1 - operand2
26              "multiply" -> operand1 * operand2
27              "divide" -> operand1 / operand2
28              else -> 0.0
29          }
30          model.addAttribute( attributeName: "result", result)
31          return "result"
32      }
33  }
```

Рисунок 17 – Контроллер калькулятора

6) Результат работы

Moy site

Калькулятор Перевод валюты

Рисунок 18 – Главная страница

Назад

12 + 14 Вычислить

Рисунок 19 – Калькулятор

Результат

Результат: 26.0

Назад к калькулятору

Рисунок 20 – Результат

Конвертер валют

12 USD RUB Конвертировать

Рисунок 21 – Конвертер валют

Конвертированная сумма

Конвертированная сумма: 1200.0

Назад к конвертеру

Рисунок 22 – 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.

```
interface Entity<T> {
    var id: Int
    fun update(other: T)
}
```

Рисунок 23 - Интерфейс

2) Импровизация бд с дженериком по интерфейсу.

```
138 open class EntityDao<T : Entity<T>> {
139         private val fruits = mutableListOf<T>()
140         var count = 0
141
142         fun index(): List<T> {
143             return fruits
144         }
145
146         fun show(id: Int): T? {
147             return fruits.find { it.id == id }
148         }
149
150         fun save(fruit: T) {
151             fruit.id = fruits.size
152             fruits.add(fruit)
153             count++
154         }
155
156         fun update(id: Int, personModel: T) {
157             val existingPerson = show(id) ?: return
158             existingPerson.update(personModel)
159         }
160
161         fun delete(id: Int) {
162             fruits.removeIf { it.id == id }
163         }
164     }
```

Рисунок 24 – GenericDAO

3) Создание моделей

```
class MessageModel :
    Entity<MessageModel> {

    override var id: Int = 0
    var text: String = ""
    var year: Int = 0
    var minutes: Int = 0
    var seconds: Int = 0

    constructor()
    constructor(id: Int, text: String, year: Int, minutes: Int, seconds: Int) {
        this.id = id
        this.text = text
        this.year = year
        this.minutes = minutes
        this.seconds = seconds
    }


    override fun update(other: MessageModel) {
        text = other.text
        year = other.year
        minutes = other.minutes
        seconds = other.seconds
    }
}

class BookModel :
    Entity<BookModel> {

    override var id: Int = 0
    var name: String = ""
    var author: String = ""
    var year: Int = 0
    var caption: String = ""
    constructor()
    constructor(id: Int, name: String, author: String, year: Int, caption: String) {
        this.id = id
        this.name = name
        this.author = author
        this.year = year
        this.caption = caption
    }


    override fun update(other: BookModel) {
        name = other.name
        author = other.author
```

Рисунок 25 - Модели

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

```
@Component
class DaoBook : EntityDao<BookModel>() {

    init {
        save(BookModel( id: 0,  name: "dawf",  author: "kto to",  year: 16,  caption: "DOKAwfoawokfko"))
        save(BookModel( id: 1,  name: "awfawf",  author: "kto to 2",  year: 26,  caption: "DOKAwfoawokfko"))
        save(BookModel( id: 2,  name: "awfw",  author: "kto to 3",  year: 216,  caption: "DOKAwfoawokfko"))
        save(BookModel( id: 3,  name: "aggwa",  author: "kto to 4",  year: 3316,  caption: "DOKAwfoawokfko"))
    }
}
```

Рисунок 26 – BookDAO

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

```kotlin
@Controller
@RequestMapping("book")
class BookController {

    @Autowired
    lateinit var dao:DaoBook

    @GetMapping
    fun index(model: Model) : String {
        model.addAttribute( attributeName: "book", dao.index())
        return "book/index"
    }

    @GetMapping("/{id}")
    fun show(@PathVariable("id") id: Int, model: Model): String
        val book = dao.show(id) ?: throw NullPointerException()
        model.addAttribute( attributeName: "book", book)
        return "book/show"
    }

    @GetMapping("/new")
    fun new(model: Model): String {
```

Рисунок 27 – Контроллер Book

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

```
main
kotlin
ru.beenaxis.runner.pr2
controller
BookController
DocumentController
FruitController
MessageController
PersonController
dao
Dao.kt
Daos.kt
Pr2Application.kt
resources
static
templates
book
edit.html
index.html
new.html
show.html
doc
fruit
msg
person
index.html
application.properties
test
gitignore
```

```html
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="st
    <title>Edit doc</title>
</head>
<body>

<form th:method="PATCH" th:action="@{/book/{id}(id = ${book.id})}" th:object="${book}">
  <label for="name">Enter name</label>
    <input type="text" id="name" th:field="*{name}">
    <label for="author">Enter author</label>
    <input type="text" id="author" th:field="*{author}">
    <label for="year">Enter year</label>
    <input type="text" id="year" th:field="*{year}">
    <label for="caption">Enter caption</label>
    <input type="text" id="caption" th:field="*{caption}">
    <input type="submit" value="Edit">
</form>


</body>
</html>
```

Рисунок 28 – Html страница Edit

```
application.properties
1   spring.mvc.hiddenmethod.filter.enabled=true
```

Рисунок 29 – 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.

```
spring.datasource.url=jdbc:postgresql://localhost:5432/pr3
spring.datasource.username=postgres
spring.datasource.password=123

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

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update

spring.mvc.hiddenmethod.filter.enabled=true
```

Рисунок 30 – Конфиг спринга

2) Подключение зависимостей и драйвера постгрес.

```
implementation("org.springframework.boot:spring-boot-starter-data-jpa")
implementation("org.springframework.boot:spring-boot-starter-jdbc")
implementation("org.springframework.boot:spring-boot-starter-validation")

implementation("org.springframework.boot:spring-boot-starter-web")
implementation("org.postgresql:postgresql")
```

Рисунок 31 – Подключение зависимостей

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

```
@Entity
@Table(name = "book")
class BookModel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Int = 0

    @Column(name = "name", nullable = false)
    @Size(min = 3, max = 200)
    var name: String = ""
    @Column(name = "author", nullable = false)
    @Size(min = 3, max = 200)
    var author: String = ""

    @Min(value = 0, message = "pages must be greater than 0")
    var year: Int = 0

    @Column(name = "caption", nullable = false)
    @Size(min = 0, max = 200)
    var caption: String = ""
}
```

Рисунок 32 – Аннотации JPA и Validator

4) MinMax, означает минимальное и максимально возможное значение поля.

```
@Min(value = 0, message = "pages must be greater than 0")
@Max(value = 16, message = "pages must be smaller than 16")
var pages: Int = 0

@Min(value = 0, message = "pages must be greater than 0")
@Max(value = 10000, message = "pages must be smaller than 10000")
var year: Int = 0
```

Рисунок 33 – MinMax Аннотации

5) Аннотации Positive означает что в этом поле всегда будет положительное число

```
@Column(name = "sugar", nullable = false)
@Positive
var sugar: Double = 0.0
```

Рисунок 34 - @Positive

6) Создание прокси репозитория, добавление прокси функции, которую сгенерирует Spring.

```
package org.example.pr3.repositories

import org.example.pr3.model.BookModel
import org.example.pr3.model.FruitModel
import org.springframework.data.jpa.repository.JpaRepository

interface BookRepository : JpaRepository<BookModel, Long> {
    fun getByName(name: String)
}
```

Рисунок 35 – BookRepository

7) Страница для поиска, через форму отправляется запрос.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css" rel="stylesheet">
    <title>Fruit search</title>
</head>
<body>
<div th:each="fruit: ${fruit}">
    <a th:href="@{/fruit/{id}(id = ${fruit.getId()})}" th:text="${fruit.toString()}"></a>
</div>
<form action="/fruit/search" method="get">
    <label>
        <input type="text" name="query" placeholder="Поиск фруктика">
    </label>
    <button type="submit">Искать фруктик</button>
</form>
</body>
</html>
```

Рисунок 36 – Поиск

7) Поиск с вызовом getByName, который вернёт по имени сущность.

```
@GetMapping("/search")
fun search(model: Model, @RequestParam("query") query: String): String {
    model.addAttribute( attributeName: "query", query)
    model.addAttribute( attributeName: "fruit", repo.getByName(query))
    return "/fruit/search"
}
```

Рисунок 37 – 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.

```
@ManyToOne(cascade = [CascadeType.ALL])
@JoinColumn(name = "person_id")
var writer:PersonModel? = null

@OneToOne(cascade = [CascadeType.ALL])
@JoinColumn(name = "fruit_id")
var fruit:FruitModel? = null

@OneToMany(cascade = [CascadeType.ALL])
@JoinColumn(name = "message_id")
var messages:MutableList<MessageModel> = mutableListOf()

@ManyToMany(cascade = [CascadeType.ALL])
@JoinTable(
    name = "document_mtm",
    joinColumns = [JoinColumn(name = "book_id")],
    inverseJoinColumns = [JoinColumn(name = "document_id")]
)
var documents:MutableList<DocumentModel> = mutableListOf()
```

Рисунок 38 – Аннотации связей

2) 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

```
@ManyToMany(cascade = [CascadeType.ALL])
@JoinTable(
    name = "document_mtm",
    joinColumns = [JoinColumn(name = "book_id")],
    inverseJoinColumns = [JoinColumn(name = "document_id")]
)
var documents:MutableList<DocumentModel> = mutableListOf()
```

Рисунок 39 – МКМ

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

```
@OneToMany(fetch = FetchType.LAZY, mappedBy = "writer", cascade = [CascadeType.ALL])
var bookWrited:List<BookModel> = listOf()
```

Рисунок 40 – OneToMany и ManyToOne

4) В DataGrip построим диаграмму для визуализации связей.

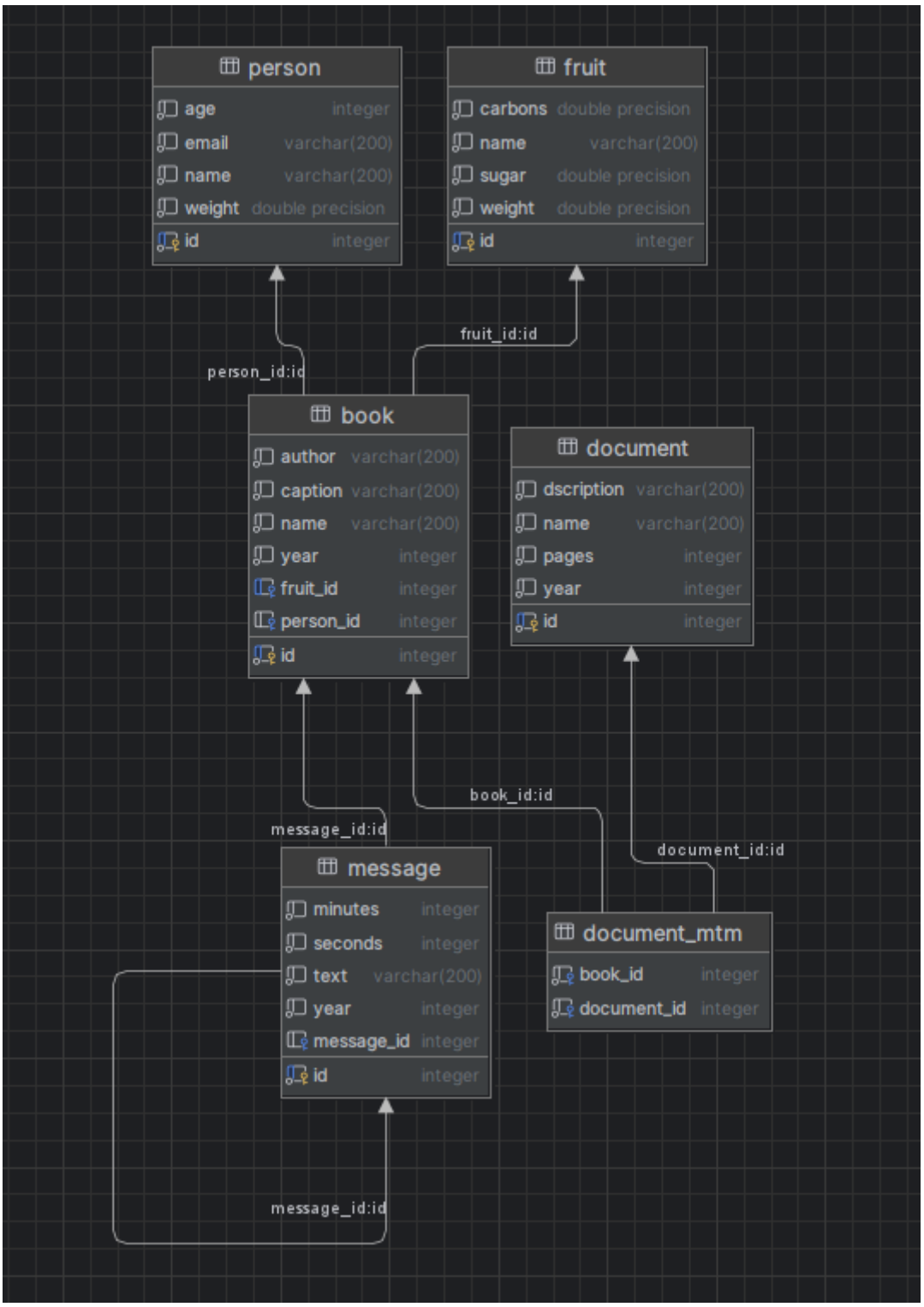


Рисунок 41 – Результат работы

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.

```
implementation("org.springframework.boot:spring-boot-starter-security")
```

Рисунок 42 – Подключение Spring Security

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

```
@Entity
@Table(name = "users")
class UserModel : UserDetails{
    @Id
    @GeneratedValue
    val id:Long = 0L

    var login:String = ""
    var pass:String = ""
    var role:String = ""

    constructor()

    constructor(login:String, password:String, role:String){
        this.login = login
        this.pass = password
        this.role = role
    }

    override fun getAuthorities(): MutableCollection<out GrantedAuthority> {
        return mutableListOf(SimpleGrantedAuthority(role))
    }

    override fun getPassword(): String {
        return pass
    }
```

Рисунок 43 – Модель юзера

3) Конфигурация доступов ко всем путям

```
@Bean
fun configure(http: HttpSecurity): DefaultSecurityFilterChain? {
    http
        .authorizeHttpRequests
         { it: AuthorizeHttpRequestsConfigurer<HttpSecurity!>.AuthorizationManagerRequestMatcherRegistry!
        it.requestMatchers("/register", "/login", "/logout").permitAll()
            .requestMatchers("/", "/index").authenticated()
            .requestMatchers("/fruit/**", "/book/**").hasAnyAuthority( ...authorities: "USER", "ADMIN")
            .requestMatchers("/msg/**", "/doc/**", "role_edit/change_user_role").hasAnyAuthority
             ( ...authorities: "ADMIN")
            .anyRequest().hasAuthority( authority: "ADMIN")
    }.formLogin(withDefaults())
        .logout { it: LogoutConfigurer<HttpSecurity!>!
            it.logoutUrl("/logout") // URL, на который отправляется запрос для выхода
                .logoutSuccessUrl("/login?logout") // URL, на который перенаправляется пользователь
                 после выхода
                .permitAll()
        }
    return http.build()
}
```

Рисунок 44 Пути

4) Добавляем репозиторий User

```
interface UserRepository : JpaRepository<UserModel, Long> {
    fun getByLoginAndPass(login:String, pass:String) : UserModel?
    fun getByLogin(login:String) : UserModel?
}
```

Рисунок 45 – UserRepository

5) Создание контроллера для регистрации

```
@Controller
@RequestMapping("register")
class RegisterController(val repo:UserRepository, val hasher:PasswordEncoder = BCryptPasswordEncoder()) {

    @GetMapping
    fun index() : String {
        return "register"
    }

    @PostMapping
    fun register(@RequestParam login:String, @RequestParam password:String) : String {
        val neww = UserModel()
        println("Register $login $password")
        neww.login = login
        neww.role = "USER"
        neww.pass = hasher.encode(password)
        repo.save(neww)
        return "redirect:login"
    }
}
```

Рисунок 46 – Регистрация

6) Определение шаблона страницы регистрации

```
<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>
```

Рисунок 47 – Шаблон регистрации

7) Добавление кнопки logout

```
<form action="/logout" method="post">
    <input type="hidden" name="_csrf" th:value="${_csrf.token}" />
    <button type="submit">Logout</button>
</form>
</body>
```

Рисунок 48 – Logout

8) Регистрируемся через /register

Рисунок 49 – Register Page

WHERE ORDER BY pass

	id	login	password	pass	role
1	102	test	<null>	$2a$10$zR4XWlRaFmRm…	USER

Рисунок 50 – User в БД

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

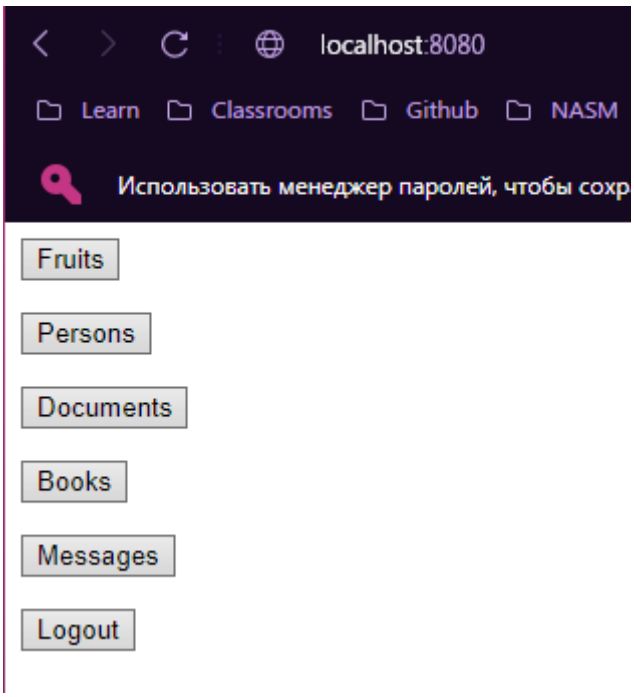


Рисунок 51 – Главная

10) Тем самым у нас теперь есть доступ ко всему почти.

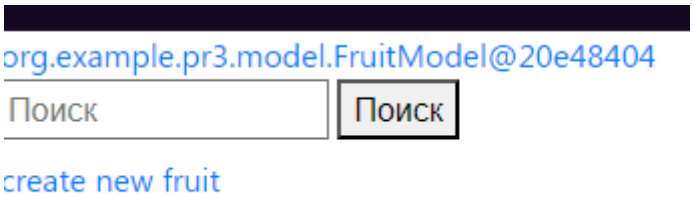


Рисунок 52 – Авторизованный

11) Если зайти в msg, доступ запрещён

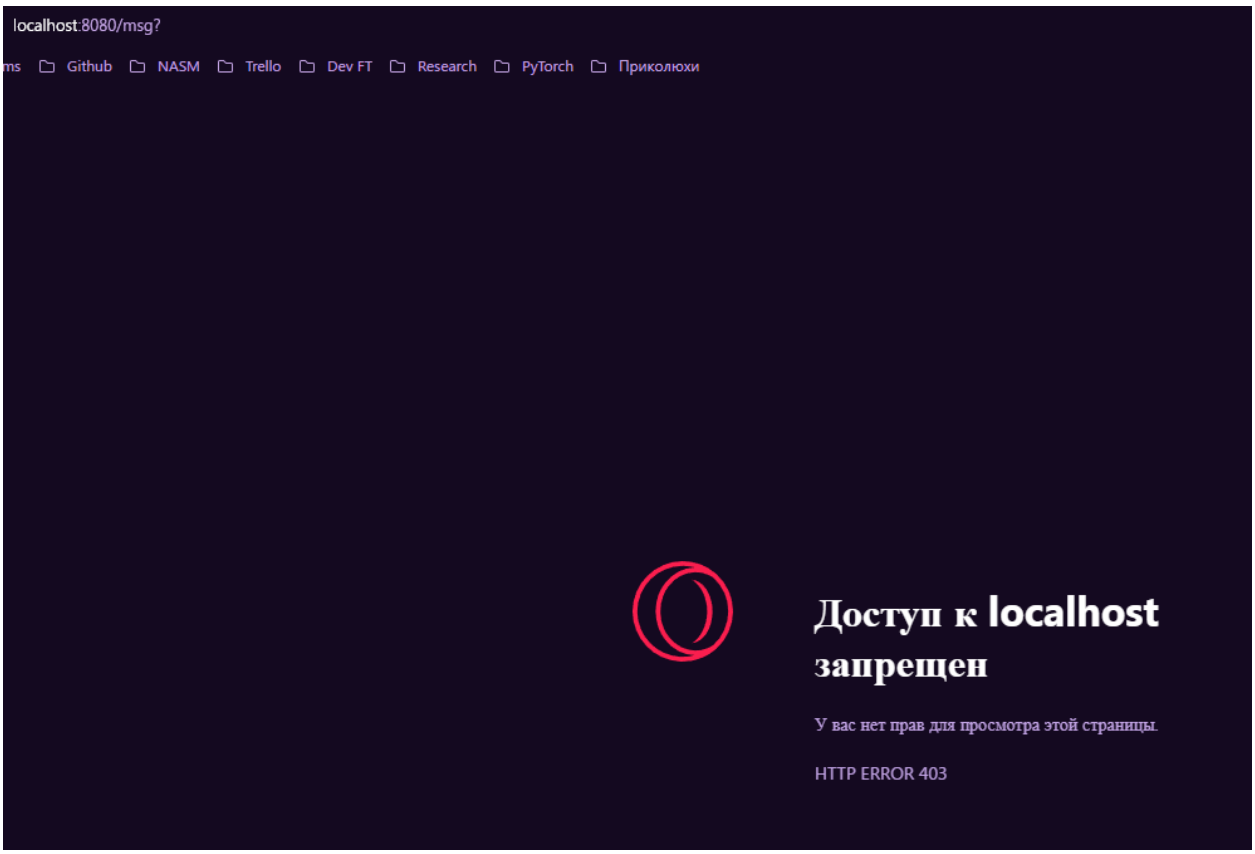


Рисунок 53 - /msg

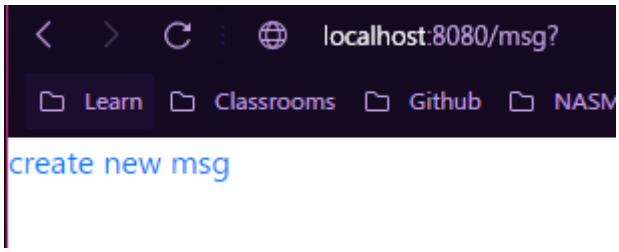
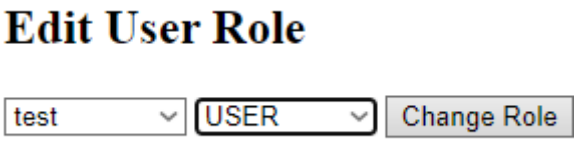


Рисунок 54 - /msg с ролью ADMIN

12) 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.

```
@Bean
protected fun passwordEncoder(): PasswordEncoder {
    return BCryptPasswordEncoder()
}

@Autowired
fun configureGlobal(auth: AuthenticationManagerBuilder) {
    auth.userDetailsService(userDetailsService)
}
```

Рисунок 55 – Bean Шифрования

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

```
@Service
class PersonDetailsService(
    private val userRepository: UserRepository
) : UserDetailsService {
    override fun loadUserByUsername(username: String): UserDetails {
        if(username == "root")
            return UserModel(
                login: "root",
                BCryptPasswordEncoder().encode( rawPassword: "root"),
                role: "ADMIN"
            )
        val user = userRepository.getByLogin(username)
        if(user≠null){
            return UserModel(
                user.login,
                user.password,
                user.role
            )
        }else{
            return UserModel(
                login: "guest",
                BCryptPasswordEncoder().encode( rawPassword: "guest"),
                role: "GUEST"
            )
        }
    }
}
```

Рисунок 56 – Авторизация

3) Шифрование на пароль в юзере.

```
neww.role = "USER"
neww.pass = hasher.encode(password)
repo.save(neww)
```

Рисунок 57 – Регистрация с шифрованием

4) В БД строка с паролём зашифрована.

WHERE ORDER BY pass

	id	login	password	pass	role
1	102	test	<null>	$2a$10$zR4XWlRaFmRm…	USER

5) Задействуем роль в путях.

```
@Bean
fun configure(http: HttpSecurity): DefaultSecurityFilterChain? {
    http
        .authorizeHttpRequests
         { it: AuthorizeHttpRequestsConfigurer<HttpSecurity!>.AuthorizationManagerRequestMatcherRegistry!
        it.requestMatchers("/register", "/login", "/logout").permitAll()
            .requestMatchers("/", "/index").authenticated()
            .requestMatchers("/fruit/**", "/book/**").hasAnyAuthority( ...authorities: "USER", "ADMIN")
            .requestMatchers("/msg/**", "/doc/**", "/role-edit", "/change-user-role")
             .hasAnyAuthority( ...authorities: "ADMIN")
            .anyRequest().hasAuthority( authority: "ADMIN")
    }.formLogin(withDefaults())
        .logout { it: LogoutConfigurer<HttpSecurity!>!
            it.logoutUrl("/logout") // URL, на который отправляется запрос для выхода
                .logoutSuccessUrl("/login?logout") // URL, на который перенаправляется пользователь
                 после выхода
                .permitAll()
        }
    return http.build()
}
```

Рисунок 58 – Новая роль и права

6) Создаём контроллер, где будет гет и пост для изменений.

```
@GetMapping("/role-edit")
fun showRoleEditForm(): ModelAndView {
    val users = userRepository.findAll()
    val modelAndView = ModelAndView( viewName: "role-edit")
    modelAndView.addObject( attributeName: "users", users)
    modelAndView.addObject( attributeName: "roles", mutableListOf("USER", "ADMIN"))
    return modelAndView
}
```

Рисунок 59 – Get

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

```
@PostMapping("/change-user-role")
fun changeUserRole(
    @RequestParam("userId") userId: Long,
    @RequestParam("roleId") roleId: String?
): String {
    val user = userRepository.findById(userId).orElse( other: null)
    //val role = roleRepository.findById(roleId).orElse(null)

    if (user != null && roleId != null) {
        user.role = roleId
        userRepository.save(user)
    }

    return "redirect:/role-edit"
}
```

Рисунок 60 – Смена роли

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

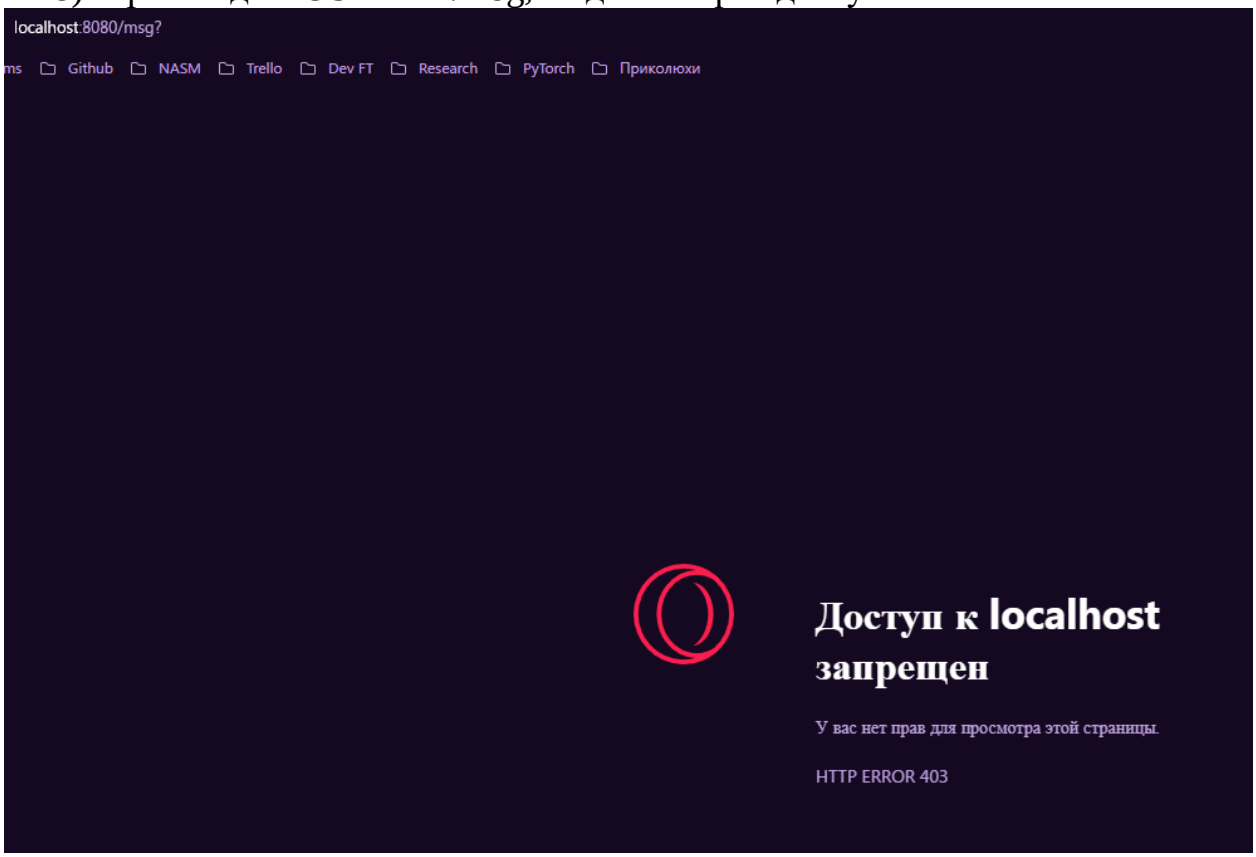


Рисунок 61 – Доступ запрещён (USER)

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

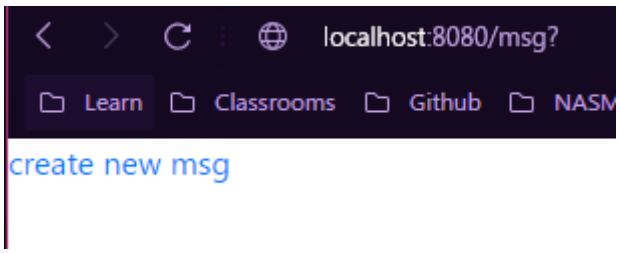


Рисунок 62 – Доступ разрешён (ADMIN)

10) При изменении роли снова на USER, доступ снова запретиться.

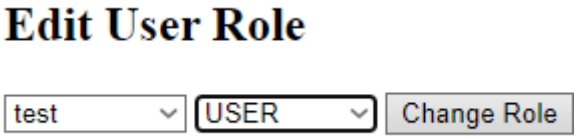


Рисунок 63 – Страница изменения роли

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