**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет

По курсовой работе

Дисциплина «JAVA-разработка»

Тема «Туристическое агентство»

Выполнил: студент группы ИКПИ-91 Березинец Д.Е./

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

Приняла: Белая Т.И./

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /

Санкт-Петербург

2021г.

**Оглавление**

1. Постановка……………………………………………………………..3
2. Приложение состоит…………………………………………………..3
3. BACK-END…………………………………………………………….3
4. FRONT-END…………………………………………………………...3
5. Основные пункты разработки ………………………………………..4
6. Безопасность…………………………………………………………...4
7. Разделение на роли…………………………………………………….5
8. Поисковая система……………………………………………………..6
9. Динамичность…………………………………………………….……6
10. Spring MVC…………………………………………………….………7
11. Работа BD…………………………………………………….………...8
12. Вид BD…………………………………………………….……………8
13. Реализация ошибок………………………………….………………....9
14. Личный кабинет ……………………………………………………...10
15. Внешний вид сайта …………………………………………………..10
16. Работа сайта…………………………………………………….……..11
17. Заключение…………………………………………………….……...11
18. Реализация кода……………………………………………………….12

**Постановка задачи**

Нужно разработать клиент-серверное приложение «Туристическое агентство», с возможностью пользователям подобрать путевку по их желанию.

**Приложение состоит**

1. Back-end - программно-аппаратная часть сервиса.
2. Front-end - клиентская сторона пользовательского интерфейса к программно-аппаратной части сервиса.

**Back-end**

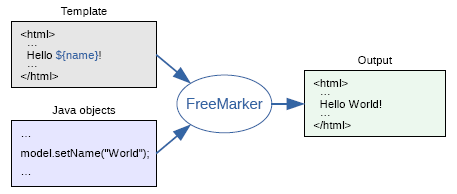
**Spring Framework**, или просто **Spring** — один из самых популярных фреймворков для создания веб-приложений на Java. Фреймворк — это что-то похожее на библиотеку (возможно этот термин вам более знаком), но есть один момент. Грубо говоря, используя библиотеку, вы просто создаете объекты классов, которые в ней есть, вызываете нужные вам методы, и таким образом получаете нужный вам результат. То есть, тут более императивный подход: вы четко указываете в своей программе в какой конкретный момент надо создать какой объект, в какой момент вызвать конкретный метод, и тд.

С фреймворками дела обстоят слегка иначе. Вы просто пишете какие-то свои классы, прописываете там какую-то часть логики, а создает объекты ваших классов и вызывает методы за вас уже сам фреймворк. Чаще всего, ваши классы имплементируют какие-то интерфейсы из фреймворка или наследуют какие-то классы из него, таким образом получая часть уже написанной за вас функциональности. Но не обязательно именно так. В спринге например стараются по максимуму отойти от такой жесткой связности (когда ваши классы напрямую зависят от каких-то классов/интерфейсов из этого фреймворка), и используют для этой цели аннотации.

**Front-end**

Apache FreeMarker™ - это движок шаблонов: библиотека Java для генерации текстового вывода (веб-страницы HTML, электронные письма, конфигурационные файлы, исходный код и т. Д.) На основе шаблонов и изменяющихся данных. Шаблоны написаны на языке шаблонов FreeMarker (FTL), который является простым, специализированным языком (а не полноценным языком программирования, таким как PHP). Обычно для подготовки данных используется язык программирования общего назначения (например, Java) (выдача запросов к базе данных, выполнение бизнес-вычислений). Затем Apache FreeMarker отображает подготовленные данные с помощью шаблонов.





**Основные пункты разработки**

1. Безопасность.
2. Разделение на роли.
3. Поисковая система.
4. Динамичность.

**Безопасность**

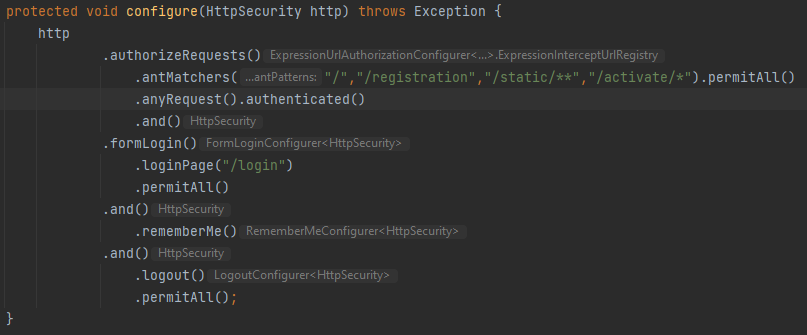
**Spring Security**

JSON Web Token (JWT) — это открытый стандарт для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях. Токены создаются сервером, подписываются секретным ключом и передаются клиенту, который в дальнейшем использует данный токен для подтверждения своей личности.

Позволяет отправлять на сервер только те запросы, которые имеют токен.



Безопасность HTTP в Spring, чтобы предоставить конфигурацию в методе configure(HttpSecurity http):



Все пароли мы хешируем:



Чтобы заставить эту случайную генерацию соли работать, BCrypt будет хранить соль внутри самого хэш-значения. Например, в следующем хэш-значении:

Есть три поля, разделенные $:

"2a" представляет версию алгоритма BCrypt

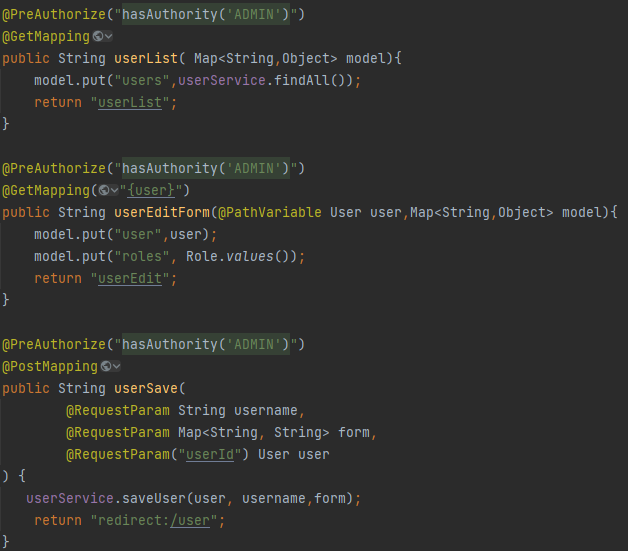
«10» представляет силу алгоритма

Объект "ZLhnHxdpHETcxmtEStgpI". часть на самом деле является случайно сгенерированной солью. В основном, первые 22 символа — это соль. Оставшаяся часть последнего поля является фактической хэшированной версией обычного текста

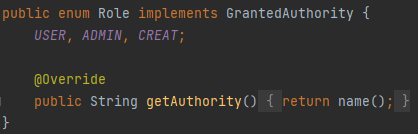
Кроме того ,алгоритм BCrypt генерирует строку длиной 60.

**Разделение на роли**

Для полной безопасности на сайте должен быть только один администратор, который уже и будет выдавать роли.



Создаем какие будут роли на сайте



Существует система выдачи роли (смотреть приложение 13 и 14), которая позволяет обеспечить безопасность, чтобы пользователь не смог добавить или изменить новости или туры. Для этих целей есть Администратор и Модератор, Администратор — полный функционал инструментов.

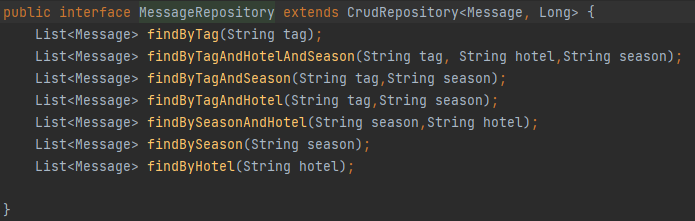
Модератор — добавлять и изменять новости и туры (смотреть приложение 11 и 16).

Если пользователь не зарегистрирован, он не сможет перейти на другие страницы, может видеть только основную и страницу авторизации и регистрации. Также каждая страница защищена собственными токенами(метками), чтобы избегать сторонние запросы на странице. При регистрации пароли кешируются и заносятся в базу данных. Также на странице регистрации присутствует ReCAPTCHA для защиты сайта от интернет-ботов.

(смотреть приложение 5)

**Поисковая система**

Мы создаем Repository, чтобы обращаться к классу базы данных, и осуществлять поиск по тем параметрам, которые мы указали.



**Динамичность**

Динамический сайт — сайт, состоящий из динамичных страниц — шаблонов, контента, скриптов и прочего, в большинстве случаев хранящихся на сервере как отдельные ресурсы (файлы, данные в базах данных и на сторонних серверах). При запросе страницы клиентом (браузером или иным приложением) она может формироваться двумя способами или их комбинацией: а) на стороне сервера из страницы-шаблона и отдельно хранимого содержимого (информации, скриптов и др.). в приложении клиента, с подгрузкой ресурсов.

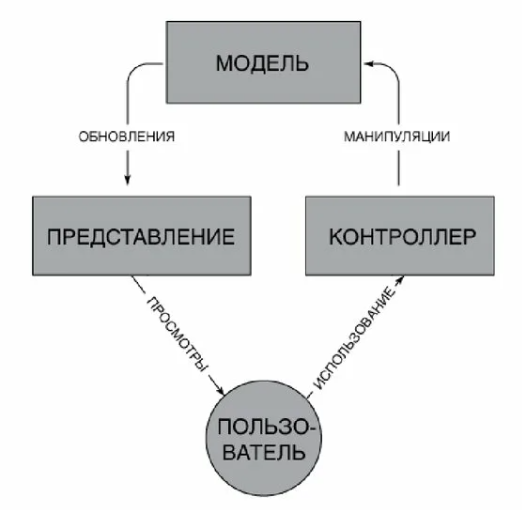
Редактирование собственно содержимого (как и страницы-шаблона) может производиться как средствами самого сайта, так и с применением стороннего ПО. Возможность править все страницы предоставляется только определённой категории пользователей (например администраторам, или же зарегистрированным пользователям). В некоторых случаях к правке определённого веб-контента допускаются анонимные пользователи, что бывает реже (например на форумах — добавление сообщений).

Добавление / Редактирование и Удаления можно выполнять сразу на сайте и обновляется сразу в BD.

* Добавление — (смотреть приложение 11).
* Редактирование — (смотреть приложение 23).
* Удаление - (смотреть приложение 18).

**Spring MVC**

Фреймворк Spring MVC обеспечивает архитектуру паттерна Model — View — Controller (Модель — Отображение (далее — Вид) — Контроллер) при помощи слабо связанных готовых компонентов. Паттерн MVC разделяет аспекты приложения (логику ввода, бизнес-логику и логику UI), обеспечивая при этом свободную связь между ними.



* Model (Модель) инкапсулирует (объединяет) данные приложения, в целом они будут состоять из POJO («Старых добрых Java-объектов», или бинов).
* View (Отображение, Вид) отвечает за отображение данных Модели, — как правило, генерируя HTML, которые мы видим в своём браузере.
* Controller (Контроллер) обрабатывает запрос пользователя, создаёт соответствующую Модель и передаёт её для отображения в Вид.

Ниже приведена последовательность событий, соответствующая входящему HTTP-запросу:

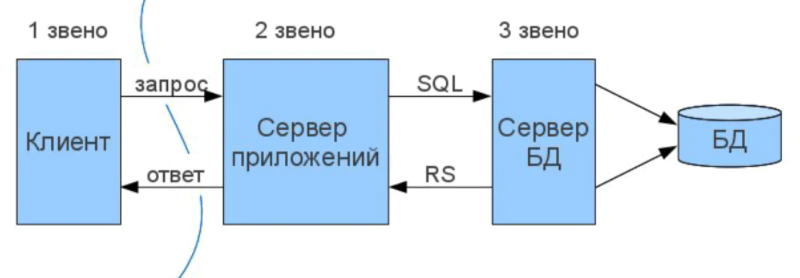
* После получения HTTP-запроса DispatcherServlet обращается к интерфейсу HandlerMapping, который определяет, какой Контроллер должен быть вызван, после чего, отправляет запрос в нужный Контроллер.
* Контроллер принимает запрос и вызывает соответствующий служебный метод, основанный на GET или POST. Вызванный метод определяет данные Модели, основанные на определённой бизнес-логике и возвращает в DispatcherServlet имя Вида (View).
* При помощи интерфейса ViewResolver DispatcherServlet определяет, какой Вид нужно использовать на основании полученного имени.
* После того, как Вид (View) создан, DispatcherServlet отправляет данные Модели в виде атрибутов в Вид, который в конечном итоге отображается в браузере.

Все вышеупомянутые компоненты, а именно, HandlerMapping, Controller и ViewResolver, являются частями интерфейса WebApplicationContext extends ApplicationContext, с некоторыми дополнительными особенностями, необходимыми для создания web-приложений.

**Работа с BD**

Вид СУБД (Система управления базой данных) - Реляционные. Данные этого вида СУБД организованы в таблицах. Таблицы могут быть связаны друг с другом, информация в них структурирована.

Система управления базами данных — это ПО, которое обеспечивает взаимодействие разных внешних программ с данными и дополнительные службы.

Для нашего сайта используем MySQL.

Классы по созданию таблиц:



**Вид BD**

Таблица пользователей

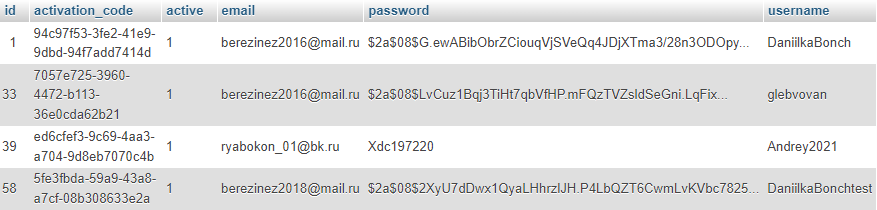


Таблица туров



Таблица новостей

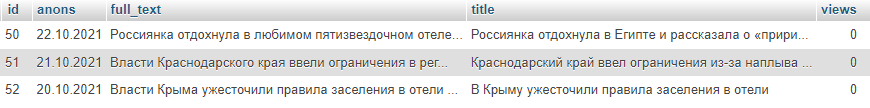


Таблица сеансов

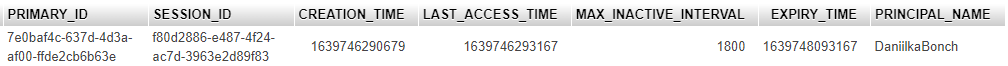
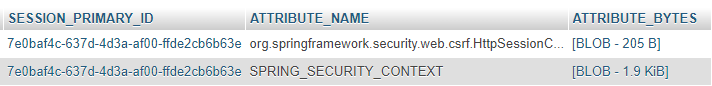
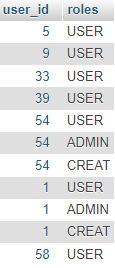
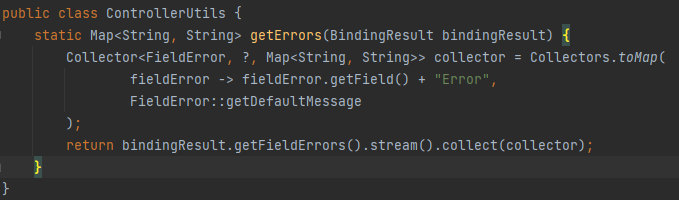


Таблица ролей



**Реализация ошибок**

Для отображения всех ошибок используем класс:



Вариант ошибки:



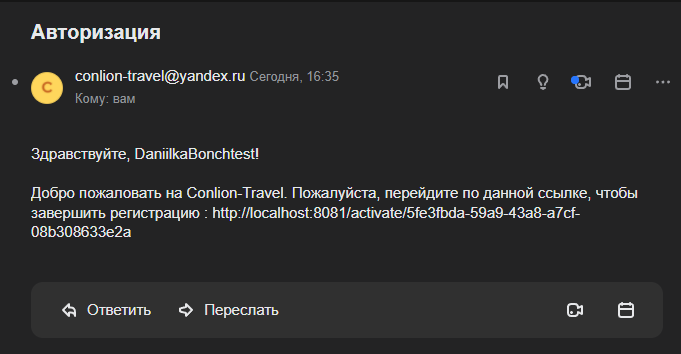
Показать пользователю:



**Личный кабинет**

При регистрации (смотреть приложение 5) пользователю на указанную почту приходит письмо для завершения регистрации. При переходе по ссылке происходит проверка авторизации, при успешной проверке пользователь заносится в базу данных и сохраняется в ней. При не успешной появляется ошибка (смотреть приложение 7).

Пример письма



**Внешний вид сайта**

Страницы:

* Основная страница для пользователя и незарегистрированных (смотреть приложение 1,2,3,19).
* Основная страница для администратора (смотреть приложение 15).
* Авторизация (смотреть приложение 4).
* Регистрация (смотреть приложение 5).
* Страница подбора тура для пользователя (смотреть приложение 21,9).
* Страница подбора тура для администратора (смотреть приложение 8,9).
* Страница профиля (смотреть приложение 12).
* Страница пользователя только для администратора (смотреть приложение 13 и 14).
* Страница добавления новостей для администратора (смотреть приложение 16).
* Страница узнать тур подробнее (смотреть приложение 18).

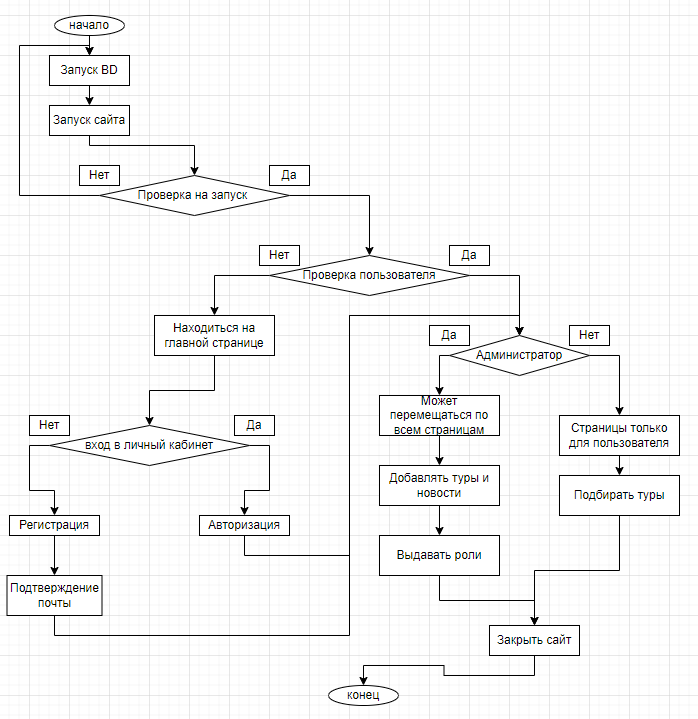
Поисковая система:

* Весь список (смотреть приложение 17).
* С параметрами (смотреть приложение 21).
* Если ничего не нашлось (смотреть приложение 22).

Реализация ошибок:

* При добавления тура (смотреть приложение 10).
* При авторизации (смотреть приложение 7).
* При регистрации (смотреть приложение 6).

**Работа сайта**



**Заключение**

При разработке туристического сайта Conlion-Travel были усвоены и изучены такие методы, как работа с Spring, MySQL и MAMP. Сайт бы успешно протестирован на локальном сервере. Все ошибки успешно отрабатывают свои задачи. Код был написан в среде разработки INTELLIJ IDEA.

**Реализация кода**

**BlogApplication**

package com.Tyrism.blog;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BlogApplication {

public static void main(String[] args) {

SpringApplication.run(BlogApplication.class, args);

}

}

**UserService**

package com.Tyrism.blog.service;

import com.Tyrism.blog.models.Role;

import com.Tyrism.blog.models.User;

import com.Tyrism.blog.repo.UserRepository;

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

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

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

import org.springframework.security.core.userdetails.UsernameNotFoundException;

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

import org.springframework.stereotype.Service;

import org.springframework.util.StringUtils;

import java.util.\*;

import java.util.stream.Collectors;

@Service

public class UserService implements UserDetailsService {

@Autowired

private UserRepository userRepository;

@Autowired

private MailSender mailSender;

@Autowired

private PasswordEncoder passwordEncoder;

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepository.findByUsername(username);

if (user == null) {

throw new UsernameNotFoundException("Пользователь не найден");

}

return user;

}

public boolean addUser(User user){

User userFromDb = userRepository.findByUsername(user.getUsername());

if (userFromDb != null) {

return false;

}

user.setActive(true);

user.setRoles(Collections.singleton(Role.USER));

user.setActivationCode(UUID.randomUUID().toString());

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

userRepository.save(user);

sendMessage(user);

return true;

}

private void sendMessage(User user) {

if (!StringUtils.isEmpty(user.getEmail())) {

String message = String.format(

"Здравствуйте, %s! \n\n" +

"Добро пожаловать на Conlion-Travel. Пожалуйста, перейдите по данной ссылке, чтобы завершить регистрацию : http://localhost:8081/activate/%s",

user.getUsername(),

user.getActivationCode()

);

mailSender.send(user.getEmail(), "Авторизация", message);

}

}

public boolean activateUser(String code) {

User user = userRepository.findByActivationCode(code);

if(user == null){

return false;

}

user.setActivationCode(null);

userRepository.save(user);

return true;

}

public List<User> findAll() {

return (List<User>) userRepository.findAll();

}

public void saveUser(User user, String username, Map<String, String> form) {

user.setUsername(username);

Set<String> roles = Arrays.stream(Role.values())

.map(Role::name)

.collect(Collectors.toSet());

user.getRoles().clear();

for (String key : form.keySet()) {

if (roles.contains(key)) {

user.getRoles().add(Role.valueOf(key));

}

}

userRepository.save(user);

}

public void updateProfile(User user, String password, String email) {

String userEmail = user.getEmail();

boolean isEmailChanged = (email != null && !email.equals(userEmail)) ||

(userEmail != null && !userEmail.equals(email));

if (isEmailChanged) {

user.setEmail(email);

if (!StringUtils.isEmpty(email)) {

user.setActivationCode(UUID.randomUUID().toString());

}

}

if (!StringUtils.isEmpty(password)) {

user.setPassword(password);

}

userRepository.save(user);

if (isEmailChanged) {

sendMessage(user);

}

}

}

**MailSender**

package com.Tyrism.blog.service;

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

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

import org.springframework.mail.SimpleMailMessage;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.stereotype.Service;

@Service

public class MailSender {

@Autowired

private JavaMailSender mailSender;

@Value("${spring.mail.username}")

public String username;

public void send(String emailTo,String subject, String message) {

SimpleMailMessage mailMessage = new SimpleMailMessage();

mailMessage.setFrom(username);

mailMessage.setTo(emailTo);

mailMessage.setSubject(subject);

mailMessage.setText(message);

mailSender.send(mailMessage);

}

}

**MessageRepository**

package com.Tyrism.blog.repo;

import com.Tyrism.blog.models.Message;

import org.springframework.data.repository.CrudRepository;

import java.util.List;

public interface MessageRepository extends CrudRepository<Message, Long> {

List<Message> findByTag(String tag);

List<Message> findByTagAndHotelAndSeason(String tag, String hotel,String season);

List<Message> findByTagAndSeason(String tag,String season);

List<Message> findByTagAndHotel(String tag,String season);

List<Message> findBySeasonAndHotel(String season,String hotel);

List<Message> findBySeason(String season);

List<Message> findByHotel(String hotel);

}

**PostRepository**

package com.Tyrism.blog.repo;

import com.Tyrism.blog.models.Post;

import org.springframework.data.repository.CrudRepository;

public interface PostRepository extends CrudRepository<Post,Long> {

}

**UserRepository**

package com.Tyrism.blog.repo;

import com.Tyrism.blog.models.User;

import org.springframework.data.repository.CrudRepository;

public interface UserRepository extends CrudRepository<User,Long>{

User findByUsername(String username);

User findByActivationCode(String code);

}

**User**

package com.Tyrism.blog.models;

import org.hibernate.validator.constraints.Length;

import org.springframework.security.core.GrantedAuthority;

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

import javax.persistence.\*;

import javax.validation.constraints.Email;

import javax.validation.constraints.NotBlank;

import java.util.Collection;

import java.util.Set;

@Entity

@Table(name = "usr")

public class User implements UserDetails {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

@Length(min=8,message = "Ник должен иметь больше 7 символов!")

private String username;

@Length(min=8,message = "Пароль должен иметь больше 7 символов!")

private String password;

private boolean active;

@Email(message = "Не правильный ввод Email")

@NotBlank(message = "Заполните строку Email")

private String email;

private String activationCode;

@ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)

@CollectionTable(name = "user\_role",joinColumns = @JoinColumn(name = "user\_id"))

@Enumerated(EnumType.STRING)

private Set<Role> roles;

public boolean isAdmin(){

return roles.contains(Role.ADMIN);

}

public Long getId(){ return id; }

public void setId(Long id){ this.id = id; }

public String getUsername(){ return username; }

@Override

public boolean isAccountNonExpired() {

return true;

}

@Override

public boolean isAccountNonLocked() {

return true;

}

@Override

public boolean isCredentialsNonExpired() {

return true;

}

@Override

public boolean isEnabled() {

return isActive();

}

public void setUsername(String username){

this.username = username;

}

@Override

public Collection<? extends GrantedAuthority> getAuthorities() {

return getRoles();

}

public String getPassword(){

return password;

}

public void setPassword(String password){

this.password = password;

}

public boolean isActive(){

return active;

}

public void setActive(boolean active){

this.active = active;

}

public Set<Role> getRoles(){

return roles;

}

public void setRoles(Set<Role> roles){

this.roles = roles;

}

public String getEmail() {

return email;

}

public String getActivationCode() {

return activationCode;

}

public void setActivationCode(String activationCode) {

this.activationCode = activationCode;

}

public User(){

}

public User(String username, String password){

this.username = username;

this.password = password;

}

//что-то могло поменяться

public void setEmail(String email) {

}

}

**Role**

package com.Tyrism.blog.models;

import org.springframework.security.core.GrantedAuthority;

public enum Role implements GrantedAuthority {

USER, ADMIN, CREAT;

@Override

public String getAuthority() {

return name();

}

}

**Post**

package com.Tyrism.blog.models;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Post {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long id;

private String title, anons;

private String full\_text;

private int views;

public Long getId(){ return id; }

public void setId(Long id){ this.id = id; }

public String getTitle(){ return title; }

public void setTitle(String title){

this.title = title;

}

public String getAnons(){

return anons;

}

public void setAnons(String anons){

this.anons = anons;

}

public String getFull\_text(){

return full\_text;

}

public void setFull\_text(String full\_text){

this.full\_text = full\_text;

}

public int getViews(){ return views; }

public void setViews(int views){

this.views = views;

}

public Post(){

}

public Post(String title,String anons,String full\_text){

this.title = title;

this.anons = anons;

this.full\_text = full\_text;

}

}

**Message**

package com.Tyrism.blog.models;

import org.hibernate.validator.constraints.Length;

import javax.persistence.\*;

import javax.validation.constraints.NotBlank;

@Entity // This tells Hibernate to make a table out of this class

public class Message {

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

private Long id;

@NotBlank(message = "Пожалуйста введите текст")

@Length(max=2048, message = "Текст не должен превышать 2048 символа")

private String text;

@NotBlank(message = "Пожалуйста введите локацию")

@Length(max=255, message = "Тэг не должен превышать 255 символа")

private String tag;

@NotBlank(message = "Пожалуйста введите проживание")

private String hotel;

@NotBlank(message = "Пожалуйста введите сезон")

private String season;

public String getHotel() {return hotel;}

public void setHotel(String hotel) {this.hotel = hotel;}

public String getSeason() {return season;}

public void setSeason(String season) {this.season = season;}

@ManyToOne(fetch = FetchType.EAGER)

@JoinColumn(name = "user\_id")

private User author;

private String filename;

public String getFilename() {return filename;}

public void setFilename(String filename) {this.filename = filename;}

public User getAuthor() {

return author;

}

public void setAuthor(User author) {

this.author = author;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getText() {

return text;

}

public void setText(String text) {

this.text = text;

}

public String getTag() {

return tag;

}

public void setTag(String tag) {

this.tag = tag;

}

public Message(){

}

public Message(String text, String tag, User user) {

this.author = user;

this.text = text;

this.tag = tag;

}

public String getAuthorName(){

return author != null ? author.getUsername() : "<none>";

}

}

**CaptchaResponseDto**

package com.Tyrism.blog.models.dto;

import com.fasterxml.jackson.annotation.JsonAlias;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import java.util.Set;

@JsonIgnoreProperties(ignoreUnknown = true)

public class CaptchaResponseDto {

private boolean success;

@JsonAlias("error-codes")

private Set<String> errorCodes;

public boolean isSuccess() {

return success;

}

public void setSuccess(boolean success) {

this.success = success;

}

public Set<String> getErrorCodes() {

return errorCodes;

}

public void setErrorCodes(Set<String> errorCodes) {

this.errorCodes = errorCodes;

}

}

**BlogController**

package com.Tyrism.blog.controllers;

import com.Tyrism.blog.models.Post;

import com.Tyrism.blog.repo.PostRepository;

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

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.RequestParam;

import java.util.ArrayList;

import java.util.Map;

import java.util.Optional;

@Controller

public class BlogController {

@Autowired

private PostRepository postRepository;

@GetMapping("/blog")

public String blogMain(Map<String,Object> model){

Iterable<Post> posts = postRepository.findAll();

model.put("posts",posts);

return "blog";

}

@PostMapping ("/blog/{id}/edit")

public String blogPostUpdate(@PathVariable(value = "id") long post\_id,@RequestParam String title, @RequestParam String anons, @RequestParam String full\_text, Model model){

Post post = postRepository.findById(post\_id).orElseThrow();

post.setTitle(title);

post.setAnons(anons);

post.setFull\_text(full\_text);

postRepository.save(post);

return "redirect:/blog";

}

@PostMapping ("/blog/{id}/remove")

public String blogPostUpdate(@PathVariable(value = "id") long post\_id, Model model){

Post post = postRepository.findById(post\_id).orElseThrow();

postRepository.delete(post);

return "redirect:/blog";

}

}

**ControllerUtils**

package com.Tyrism.blog.controllers;

import org.springframework.validation.BindingResult;

import org.springframework.validation.FieldError;

import java.util.Map;

import java.util.stream.Collector;

import java.util.stream.Collectors;

public class ControllerUtils {

static Map<String, String> getErrors(BindingResult bindingResult) {

Collector<FieldError, ?, Map<String, String>> collector = Collectors.toMap(

fieldError -> fieldError.getField() + "Error",

FieldError::getDefaultMessage

);

return bindingResult.getFieldErrors().stream().collect(collector);

}

}

**MainController**

package com.Tyrism.blog.controllers;

import com.Tyrism.blog.models.Message;

import com.Tyrism.blog.models.Post;

import com.Tyrism.blog.models.User;

import com.Tyrism.blog.repo.MessageRepository;

import com.Tyrism.blog.repo.PostRepository;

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

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

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

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.RequestParam;

import org.springframework.web.multipart.MultipartFile;

import javax.validation.Valid;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Map;

import java.util.Optional;

import java.util.UUID;

@Controller

public class MainController {

@Autowired

private MessageRepository messageRepository;

@Value("${upload.path}")

private String uploadPath;

@Autowired

private PostRepository postRepository;

@GetMapping("/")

public String home(Map<String,Object> model){

Iterable<Post> posts = postRepository.findAll();

model.put("posts",posts);

return "greeting";

}

@GetMapping("/greeting/add")

public String home(Model model){

return "greeting-add";

}

@PostMapping ("/greeting/add")

public String blogPostAdd(@RequestParam String title, @RequestParam String anons, @RequestParam String full\_text, Model model){

Post post = new Post(title,anons,full\_text);

postRepository.save(post);

return "redirect:/";

}

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

public String blogDetails(@PathVariable(value = "id") long message\_id, Model model){

if(!messageRepository.existsById(message\_id)){

return "redirect:/main";

}

Optional<Message> message = messageRepository.findById(message\_id);

ArrayList<Message> res = new ArrayList<>();

message.ifPresent(res::add);

model.addAttribute("message",res);

return "main-edit";

}

@GetMapping("/main/{id}/delail")

public String blogDetail(@PathVariable(value = "id") long message\_id, Model model){

if(!messageRepository.existsById(message\_id)){

return "redirect:/main";

}

Optional<Message> message = messageRepository.findById(message\_id);

ArrayList<Message> res = new ArrayList<>();

message.ifPresent(res::add);

model.addAttribute("message",res);

return "main-delail";

}

@PostMapping ("/main/{id}/delail")

public String blogPostUpdate(@PathVariable(value = "id") long post\_id,@RequestParam String text, @RequestParam String tag, @RequestParam String hotel, @RequestParam String season, Model model){

Message message = messageRepository.findById(post\_id).orElseThrow();

message.setText(text);

message.setTag(tag);

message.setHotel(hotel);

message.setSeason(season);

messageRepository.save(message);

return "redirect:/main";

}

@PostMapping ("/main/{id}/remove")

public String blogPostUpdate(@PathVariable(value = "id") long message\_id, Model model){

Message message = messageRepository.findById(message\_id).orElseThrow();

messageRepository.delete(message);

return "redirect:/main";

}

@PostMapping ("/greeting/{id}/remove")

public String blogPostDelete(@PathVariable(value = "id") long message\_id, Model model){

Post post = postRepository.findById(message\_id).orElseThrow();

postRepository.delete(post);

return "redirect:/";

}

@GetMapping("/main")

public String main(@RequestParam(required = false, defaultValue = "") String filter,String filter1,String filter2, Model model) {

Iterable<Message> messages = messageRepository.findAll();

Iterable<Post> posts = postRepository.findAll();

if(filter != null && !filter.isEmpty() && filter1 != null && !filter1.isEmpty()&& filter2 != null && !filter2.isEmpty()){

messages = messageRepository.findByTagAndHotelAndSeason(filter,filter2,filter1);

}else if (filter != null && !filter.isEmpty() && filter1 != null && !filter1.isEmpty()){

messages = messageRepository.findByTagAndSeason(filter,filter1);

}else if (filter != null && !filter.isEmpty() && filter2 != null && !filter2.isEmpty()) {

messages = messageRepository.findByTagAndHotel(filter, filter2);

}else if (filter1 != null && !filter1.isEmpty() && filter2 != null && !filter2.isEmpty()) {

messages = messageRepository.findBySeasonAndHotel(filter1, filter2);

}else if(filter2 != null && !filter2.isEmpty()){

messages = messageRepository.findByHotel(filter2);

} else if(filter != null && !filter.isEmpty()){

messages = messageRepository.findByTag(filter);

}else if(filter1 != null && !filter1.isEmpty()){

messages = messageRepository.findBySeason(filter1);

}else {

messages = messageRepository.findAll();

}

model.addAttribute("posts",posts);

model.addAttribute("messages", messages);

model.addAttribute("filter",filter);

model.addAttribute("filter1",filter1);

model.addAttribute("filter2",filter2);

return "main";

}

@PostMapping("/main")

public String add(

@AuthenticationPrincipal User user,

@Valid Message message,

BindingResult bindingResult,

Model model,

@RequestParam("file") MultipartFile file

) throws IOException {

message.setAuthor(user);

if (bindingResult.hasErrors()) {

Map<String, String> errorsMap = ControllerUtils.getErrors(bindingResult);

model.mergeAttributes(errorsMap);

model.addAttribute("message", message);

} else {

if (file != null && !file.getOriginalFilename().isEmpty()) {

File uploadDir = new File(uploadPath);

if (!uploadDir.exists()) {

uploadDir.mkdir();

}

String uuidFile = UUID.randomUUID().toString();

String resultFilename = uuidFile + "." + file.getOriginalFilename();

file.transferTo(new File(uploadPath + "/" + resultFilename));

message.setFilename(resultFilename);

}

model.addAttribute("message", null);

messageRepository.save(message);

}

Iterable<Message> messages = messageRepository.findAll();

model.addAttribute("messages", messages);

return "main";

}

}

**RegistrationController**

package com.Tyrism.blog.controllers;

import com.Tyrism.blog.models.User;

import com.Tyrism.blog.models.dto.CaptchaResponseDto;

import com.Tyrism.blog.repo.UserRepository;

import com.Tyrism.blog.service.UserService;

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

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.util.StringUtils;

import org.springframework.validation.BindingResult;

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.RequestParam;

import org.springframework.web.client.RestTemplate;

import javax.validation.Valid;

import java.util.Collections;

import java.util.Map;

@Controller

public class RegistrationController {

private final static String CAPTCHA\_URL = "https://www.google.com/recaptcha/api/siteverify?secret=%s&response=%s";

@Autowired

private UserRepository userRepository;

@Value("${recaptcha.secret}")

private String secret;

@Autowired

private RestTemplate restTemplate;

@Autowired

private UserService userService;

@GetMapping("/registration")

public String registration(){

return "registration";

}

@PostMapping("/registration")

public String addUser(

@RequestParam("password2") String passwordConfirm,

@RequestParam("g-recaptcha-response") String captchaResponce,

@Valid User user,

BindingResult bindingResult,

Model model) {

String url = String.format(CAPTCHA\_URL,secret,captchaResponce);

CaptchaResponseDto response = restTemplate.postForObject(url, Collections.emptyList(), CaptchaResponseDto.class);

if (!response.isSuccess()) {

model.addAttribute("captchaError","Проверка была провалена");

}

boolean isConfirmEmpty = StringUtils.isEmpty(passwordConfirm);

if(isConfirmEmpty){

model.addAttribute("password2Error","Введите пароль ");

}

if (user.getPassword() != null && !user.getPassword().equals(passwordConfirm)) {

model.addAttribute("passwordError", "Пароль не совподает!");

}

if (isConfirmEmpty || bindingResult.hasErrors() || !response.isSuccess()) {

Map<String, String> errors = ControllerUtils.getErrors(bindingResult);

model.mergeAttributes(errors);

return "registration";

}

if (!userService.addUser(user)) {

model.addAttribute("usernameError", "Такой пользователь существует!");

return "registration";

}

return "redirect:/login";

}

@GetMapping("/activate/{code}")

public String activate(Model model, @PathVariable String code){

boolean isActivated = userService.activateUser(code);

if (isActivated){

model.addAttribute("messageType","success");

model.addAttribute("message","Пользователь успешно зарегистрирован");

}else{

model.addAttribute("messageType","danger");

model.addAttribute("message","Код активации не найден");

}

return "login";

}

}

**UserController**

package com.Tyrism.blog.controllers;

import com.Tyrism.blog.models.Role;

import com.Tyrism.blog.models.User;

import com.Tyrism.blog.service.UserService;

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

import org.springframework.security.access.prepost.PreAuthorize;

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

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

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

import java.util.Map;

@Controller

@RequestMapping("/user")

public class UserController {

@Autowired

private UserService userService;

@PreAuthorize("hasAuthority('ADMIN')")

@GetMapping

public String userList( Map<String,Object> model){

model.put("users",userService.findAll());

return "userList";

}

@PreAuthorize("hasAuthority('ADMIN')")

@GetMapping("{user}")

public String userEditForm(@PathVariable User user,Map<String,Object> model){

model.put("user",user);

model.put("roles", Role.values());

return "userEdit";

}

@PreAuthorize("hasAuthority('ADMIN')")

@PostMapping

public String userSave(

@RequestParam String username,

@RequestParam Map<String, String> form,

@RequestParam("userId") User user

) {

userService.saveUser(user, username,form);

return "redirect:/user";

}

@GetMapping("profile")

public String getProfile(Model model, @AuthenticationPrincipal User user) {

model.addAttribute("username",user.getUsername());

model.addAttribute("email",user.getEmail());

return "profile";

}

@PostMapping("profile")

public String updateProfile(

@AuthenticationPrincipal User user,

@RequestParam String password,

@RequestParam String email) {

userService.updateProfile(user, password, email);

return "redirect:/user/profile";

}

}

**WebSecurityConfig**

package com.Tyrism.blog.Config;

import com.Tyrism.blog.service.UserService;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;

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

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

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

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

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity(prePostEnabled = true)

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserService userService;

@Autowired

private PasswordEncoder passwordEncoder;

@Bean

public PasswordEncoder getPasswordEncoder() {

return new BCryptPasswordEncoder(8);

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.antMatchers("/","/registration","/static/\*\*","/activate/\*").permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll()

.and()

.rememberMe()

.and()

.logout()

.permitAll();

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception{

auth.userDetailsService(userService)

.passwordEncoder(passwordEncoder);

}

}

**MvcConfig**

package com.Tyrism.blog.Config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.client.RestTemplate;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.ViewControllerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration

public class MvcConfig implements WebMvcConfigurer {

@Value("${upload.path}")

private String uploadPath;

@Bean

public RestTemplate getRestTemplate(){

return new RestTemplate();

}

public void addViewControllers(ViewControllerRegistry registry) {

registry.addViewController("/login").setViewName("login");

}

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/img/\*\*")

.addResourceLocations("file://" + uploadPath + "/");

registry.addResourceHandler("/static/\*\*")

.addResourceLocations("classpath:/static/");

}

}

**MailConfig**

package com.Tyrism.blog.Config;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.mail.javamail.JavaMailSenderImpl;

import java.util.Properties;

@Configuration

public class MailConfig {

@Value("${spring.mail.host}")

public String host;

@Value("${spring.mail.username}")

public String username;

@Value("${spring.mail.password}")

public String password;

@Value("${spring.mail.port}")

public int port;

@Value("${spring.mail.protocol}")

public String protocol;

@Value("${mail.debug}")

public String debug;

@Bean

public JavaMailSender getMailSender(){

JavaMailSenderImpl mailSender = new JavaMailSenderImpl();

mailSender.setHost(host);

mailSender.setUsername(username);

mailSender.setPassword(password);

mailSender.setPort(port);

Properties properties = mailSender.getJavaMailProperties();

properties.setProperty("mail.transport.protocol",protocol);

properties.setProperty("mail.debug",debug);

return mailSender;

}

}

**Application.properties**

server.port = 8081

spring.datasource.url=jdbc:mysql://${MYSQL\_HOST:localhost}:3306/spring\_web\_blog

spring.datasource.username=root

spring.datasource.password=root

spring.jpa.generate-ddl=false

spring.jpa.show-sql=false

spring.jpa.hibernate.ddl-auto=update

spring.freemarker.expose-request-attributes=true

upload.path=/D:/upload

spring.jpa.properties.hibernate.jdbc.lob.non\_contextual\_creation=true

spring.mail.host=smtp.yandex.ru

spring.mail.username=XXXX

spring.mail.password=XXXX

spring.mail.port=465

spring.mail.protocol=smtps

mail.debug=true

recaptcha.secret=6Le7UdwcAAAAAMOJsaCu0Vm1rVb4uFYViS0r9CoK

spring.session.jdbc.initialize-schema=always

spring.session.jdbc.table-name=SPRING\_SESSION