Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

Курсовая работа

Реализация приложения «Патентное Бюро» с помощью «Spring Framework»

по дисциплине «ПО распределённых вычислительных систем»

Работу выполнил Филиппов Д.В.
студент группы 15341/3

Работу принял Стручков И.В.

«___»_____ 2018г.

Санкт-Петербург

2018

Оглавление

Анализ задания

1. Формулировка индивидуального задания

Реализовать приложение «Патентное Бюро» с помощью Spring Framework

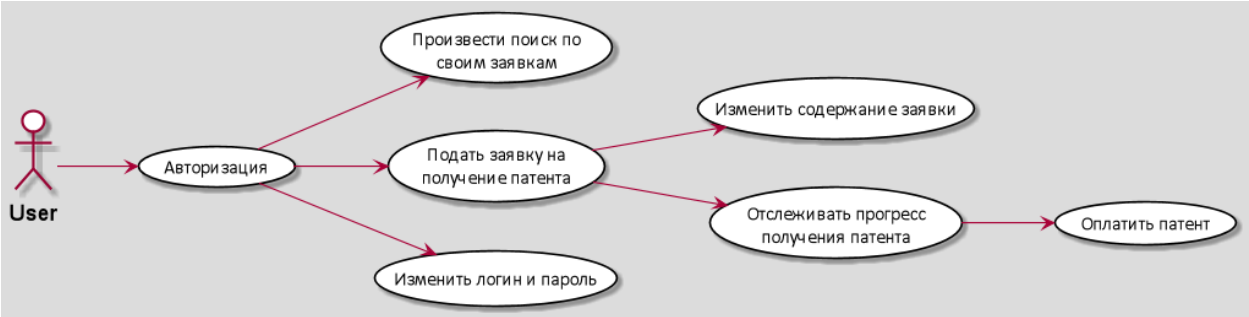
2. Требования, которым должно удовлетворять приложение

Приложение должно строиться, используя следующие принципы и инструменты:

- Spring Fraemwork
- Rest
- Паттерны

3. Основные варианты использования

Клиент



Подать заявку на получение патента

1. Клиент подает заявку на получение патента.
2. Клиент предоставляет краткое описание изделия, тип изделия и документ, полностью описывающий изделие.
3. Клиент получает от системы оповещение об успешном прохождении проверки заявки.

Альтернатива: *Введены некорректные описание или тип изделия*: На шаге 3 устанавливается, что введенные описание и тип изделия некорректны. Клиенту предоставляется возможность повторного ввода описания и типа изделия.

Получение патента

1. Клиент получает от системы оповещение об успешном прохождении верификации.
2. Клиент получает от системы номер счета, на который следует произвести оплату, и цену, сколько это будет стоить.
3. Система подтверждает оплату и выдает Клиенту ссылку на патент его изделия.

Альтернатива: *Оплата не прошла*: На шаге 3 оказывается, что Клиент не может оплатить услуги Патентного бюро по той или иной причине. Клиенту предлагается поменять способ оплаты, в противном случае патент он не получит.

Отказ от патента

1. Клиент получает от системы оповещение о непрохождении верификации и список причин, повлиявших на этот результат.
2. Клиент делает выбор не вносить изменения в свое изделие, тем самым отказывается от патента.

Альтернатива: *Клиент исправляет заявку*: На шаге 2 Клиент решил исправить свое изделие, тем самым отправив заявку на повторную верификацию.

Админ

Прием заявки на получение патента

1. Админ получает от Клиента заявку на получение патента его изделия.
2. Админ проверяет заявку Клиента на корректность: если она корректна, помечает ее и передает верификатору.
3. Если заявка некорректна, Админ уведомляет Клиента об этом и указывает, что необходимо исправить
4. После подтверждения Верификатором, что заявка успешно прошла верификацию, Админ предоставляет Клиенту номер счета, на который следует произвести оплату, и цену, сколько это будет стоить.
5. После подтверждения оплаты, Админ выдает Клиенту ссылку на патент его изделия.

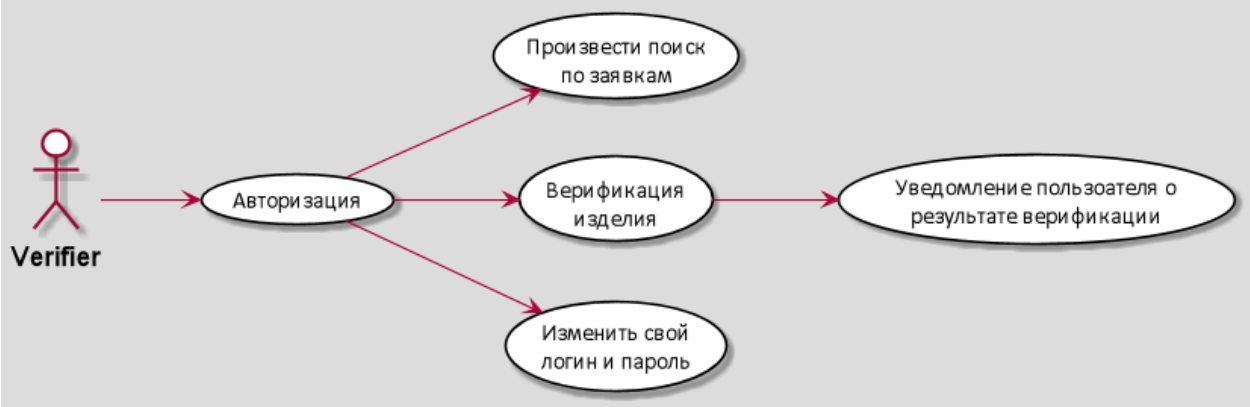
Альтернатива: *Заявка не прошла верификацию*: на шаге 4 заявка не прошла верификацию, поэтому клиенту следует полностью ее исправить. Админ заново проверяет исправленную заявку.

Прекращение работы с клиентом

1. Админ видит, что Клиент отказался от исправления заявки, поэтому он удаляет Клиента из базы.

Альтернатива: *Логин Клиента содержит мат*: Если логин клиента содержит мат, то Админ исправляет его на стандартный. Если Клиент снова поменяет свой Логин на неподобающий, Админ удаляет его из базы.

Верификатор

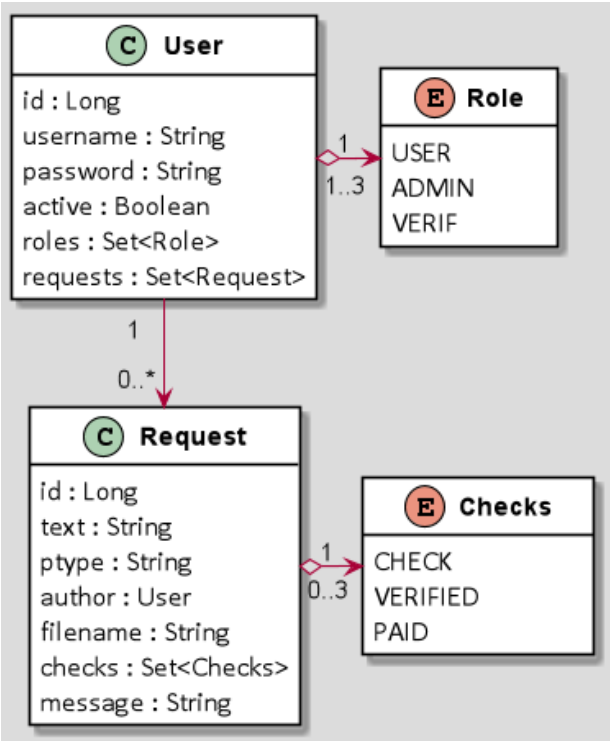


Верификация изделия

1. Верификатор авторизуется в системе.
2. Верификатор обращается к системе с целью получения заявок от Клиентов.
3. Верификатору предоставляется список всех заявок.
4. Верификатор выбирает одну из заявок Клиентов, которую он хочет верифицировать.
5. Верификатор производит верификацию изделия и сообщает о ее результатах Клиенту.

Альтернатива: *Непрохождение верификации*: На шаге 5 оказывается, что изделие не прошло верификацию. Верификатор сообщает Клиенту о причинах, почему это произошло.

4. Описание модели предметной области



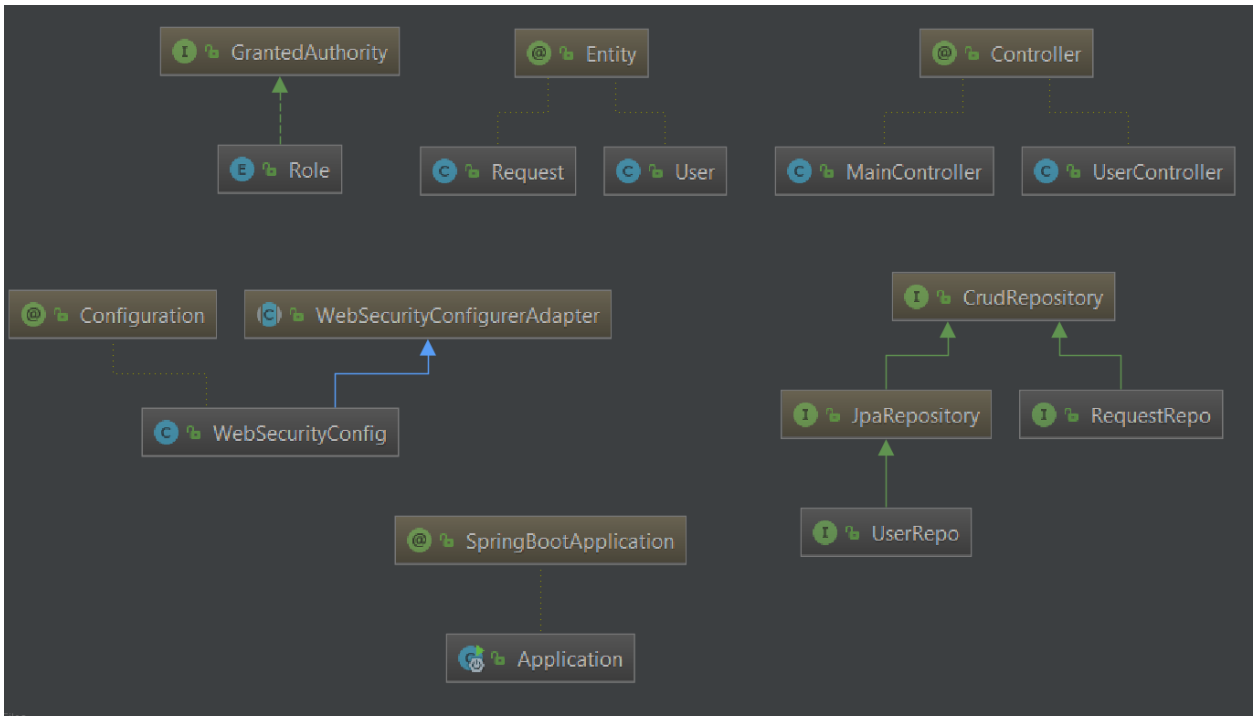
Вовлеченные Сущности

В результате разработки вариантов использования были выделены следующие вовлеченные сущности:

- клиент; логин и пароль клиента; авторизация клиента; заявка клиента
- изделие; тип изделия; краткое описание изделия; документ с полным описанием изделия; список патентов, закрепленных за клиентом;
- верификация; результаты верификации.

Реализация задания с помощью «Spring Framework»

1. Объектно-ориентированное проектирование с учётом особенностей технологии



@SpringBootApplication – сканирует все контроллеры, сервисы и прочие составляющие спринга, чтобы все это вместе запустить

Application – класс запуска программы

@Controller – принимает запросы пользователя и возвращает какие-то данные

MainController – контроллер добавления, регистрации и создания заявки

UserController – контроллер управления пользователем и заявкой

@Entity – дает знать спрингу, что это часть кода – сущность, которую мы сохраняем в БД

Request – класс заявки

User – класс пользователя

GrantedAuthority – полномочия, которые предоставляются пользователю. Используются для авторизации.

Role – enum-класс с ролями

@Configuration – при старте приложения конфигурирует веб-секюрити

WebSecurityConfigurerAdapter – обеспечивает безопасность на основе веб:

- Управление доступом к URL с учетом аутентификации
- Управление созданием пользователя (шифровка паролей)
- Сохранение сессий

WebSecurityConfig – доступ к URL с учетом аутентификации

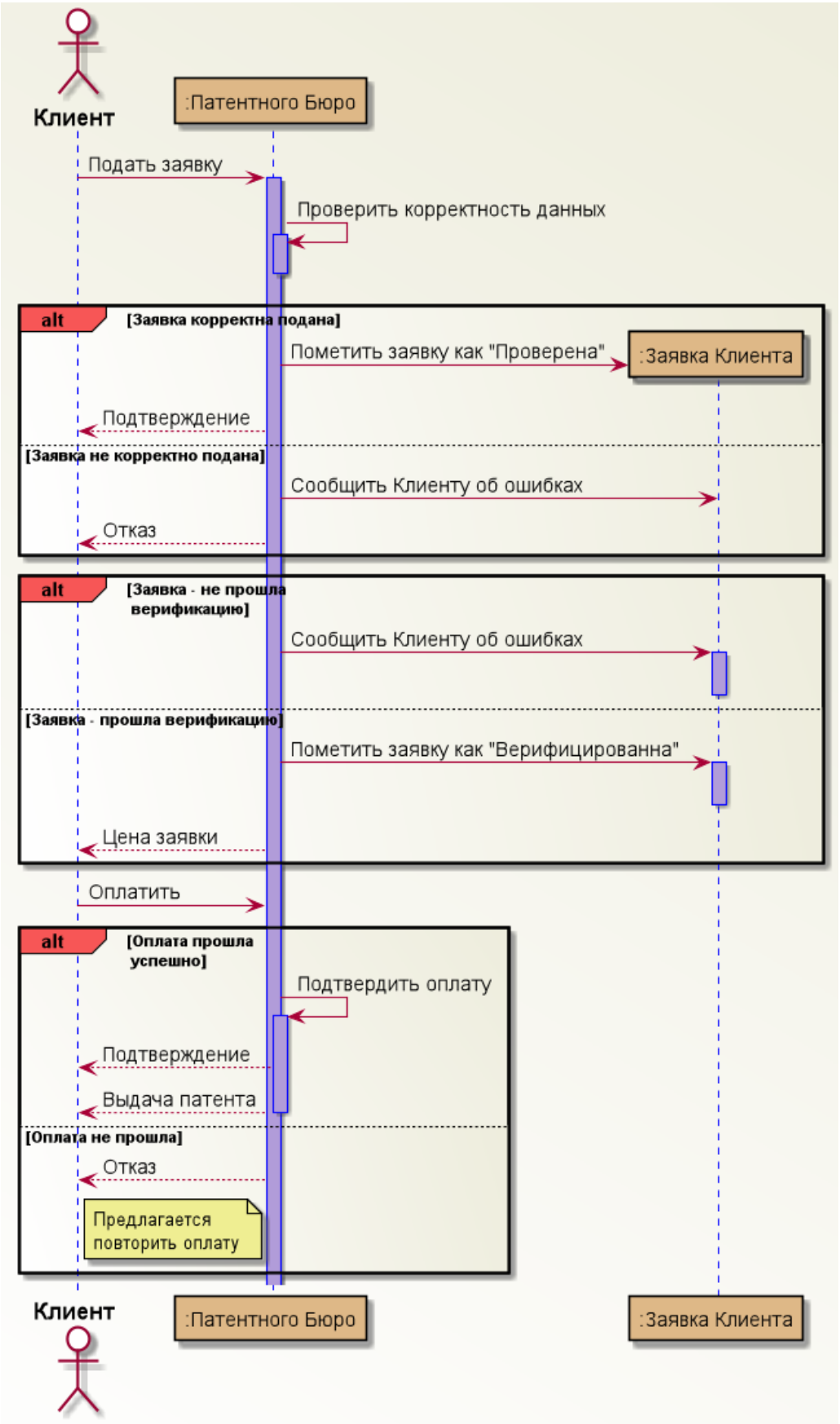
CrudRepository – обеспечивает сложные функциональные возможности CRUD для класса сущностей

JpaRepository – промежуточный интерфейс для объявления общего поведения

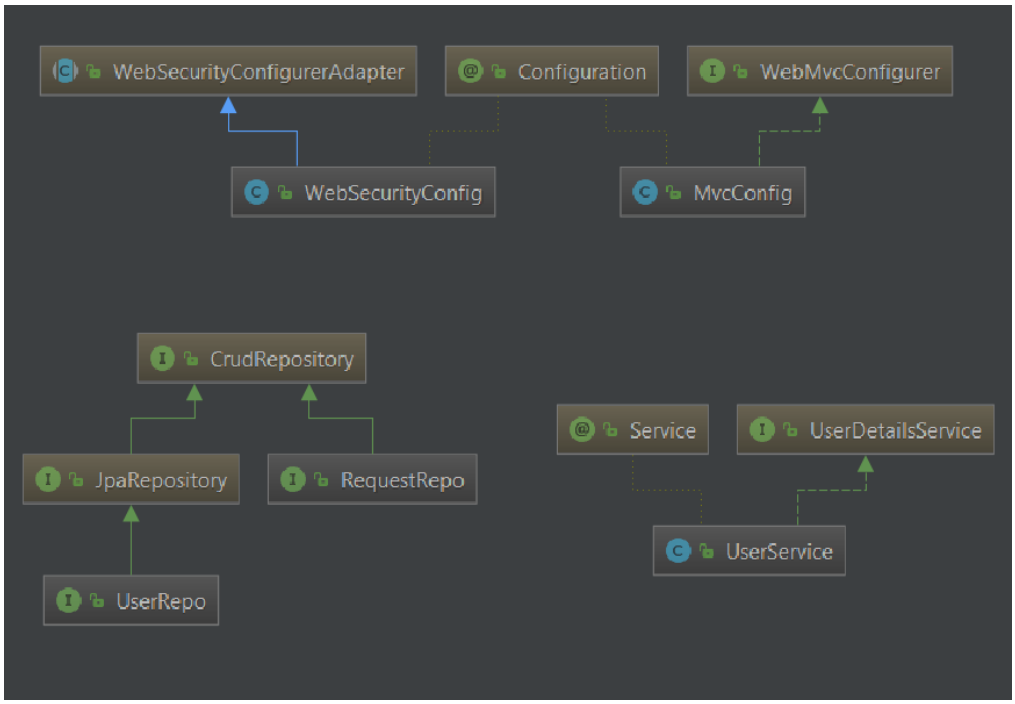
RequestRepo – запросы к БД для заявки

UserRepo – запросы к БД для пользователя

Диаграмма последовательности



2. Описание программы



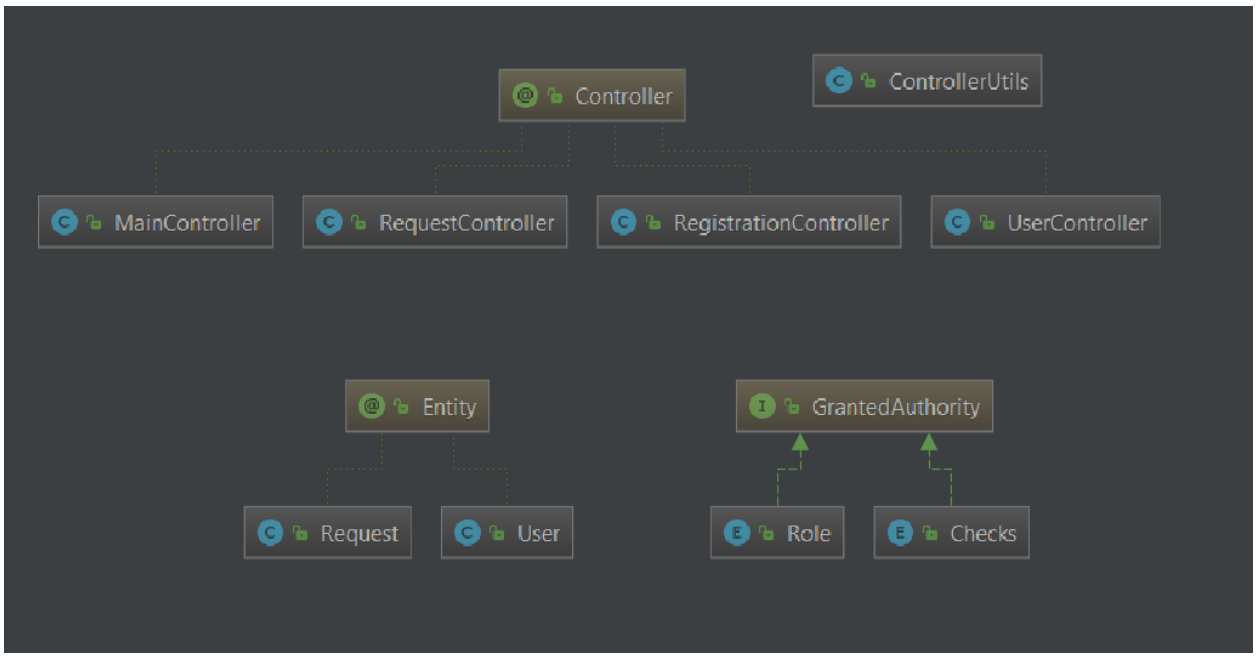
WebMvcConfigurer – базовая поддержка MVC (регистрация контроллеров и отображений, преобразователи типов, поддержка проверки, преобразователи сообщений и обработка исключений)

MvcConfig – содержит конфигурацию веб-слоя (автоматически-создаваемые контроллеры, ресурсы)

@Service – пометка сервисного слоя

UserDetailsService – основной интерфейс, который загружает пользовательские данные

UserService – реализация запросов к БД для пользователя



Checks – enum-класс проверок заявки

RegistrationController – контроллер регистрации

RequestController – контроллер обработки заявки

ControllerUtils – реализация функционала для контроллеров (сохранение документа, поиск ошибок)

3. Полный текст программы

См. Приложение.

4. Методика и результаты тестирования

Вариант тестирования	Ожидаемый результат	Фактический результат
Регистрация нового пользователя	Добавлен новый пользователь	Совпадает с ожидаемым
	Нельзя использовать уже занятый логин	Совпадает с ожидаемым
	Все поля формы регистрации должны пройти валидацию	Совпадает с ожидаемым
Авторизация	Уведомить, если логин или пароль не прошли валидацию	Совпадает с ожидаемым
	Не авторизированным пользователям видна только страница приветствия	Совпадает с ожидаемым
	Авторизованный пользователь не может изменить заявку на странице меню	Совпадает с ожидаемым
	Авторизованный пользователь может менять логин и пароль	Совпадает с ожидаемым
Роль Админа	Админ не может создать заявку	Совпадает с ожидаемым
	admin не может изменить свой логин	Совпадает с ожидаемым
	Админ может удалять пользователей	Совпадает с ожидаемым
	Админ может оставлять сообщения для пользователей	Совпадает с ожидаемым
	Админ может просматривать заявки всех пользователей	Совпадает с ожидаемым
	Админ может отмечать заявки пользователей как проверенные и оплаченные	Совпадает с ожидаемым
	Админ может удалять пользователя	Совпадает с ожидаемым

Роль Верификатора	Верификатор не может создать заявку	Совпадает с ожидаемым
	Верификатор может оставлять сообщения для пользователей	Совпадает с ожидаемым
	Верификатор может просматривать заявки всех пользователей	Совпадает с ожидаемым
	Верификатор может отмечать заявки пользователей верифицированные	Совпадает с ожидаемым
Роль Клиента	Клиент может создать заявку	Совпадает с ожидаемым
	Клиент может изменить содержание заявки	Совпадает с ожидаемым
	Клиент может просматривать только свои заявки	Совпадает с ожидаемым
	Клиент может получать сообщения от системы	Совпадает с ожидаемым

5. Инструкция системному администратору по развёртыванию приложения

1. Создать в PostgreSQL базу данных с именем RisePatent
2. Скачать архив BuroRisePatent
3. Запустить файл run.bat
4. Войти в систему (логин: admin, пароль: 1)

6. Инструкция пользователю по запуску приложения

1. Зарегистрироваться в системе
2. Нажать «Создать заявку»
3. Заполнить форму
4. Периодически проверять состояние заявки

Вывод

В данной работе было спроектировано и разработано приложение «Патентное Бюро». В процессе проектирования были закреплены на практике знания о паттернах и внутреннем устройстве Spring Framework.

На этапе разработки, были получены навыки по написанию REST приложений, а также освоен фреймворк Spring Boot, на котором и писалось приложение.

Степень выполнения поставленного задания составляет 90%, поскольку на этапе разработки было принято решение отказаться от сторонних API, таких как регистрация и уведомление через почту, платежная система и шифрование. Причиной тому стала возросшая сложность тестирования проделанных модификаций, т.к. требовалось каждый раз взаимодействовать с почтой и банковской системой оплаты. И поскольку целью данного курса было получение знаний о создании приложений с распределёнными вычислительными системами, то сторонние API здесь играло чисто дополнительный характер.

Оценивая приложение с точки зрения распределённых систем, можно сказать, что приложение является:

- Прозрачным – за счет понятной структуры программы
- Открытым – за счет свободы воли при проектировании и разработке приложения
- Масштабируемым – за счет отделения логики от реализации и использования паттернов

К дополнительным критериям, можно отнести:

- Производительность
  - запуск програмы составляет 3.73 секунды
  - запуск JVM составляет 4.443 секунды
- Удобство использования – интуитивно понятный интерфейс

В целом курс очень понравился, т.к. благодаря нему былы изучены такие важные технологии как REST и Spring.

Литература

1. Буч Г., Рамбо Д., Якобсон А. Язык UML. Руководство пользователя. ДМК, 2006.

2. Фаулер М. UML. Основы, 3-е издание. Символ-Плюс, 2006.

3. Гранд М. Шаблоны проектирования в Java. BHV-СПб, 2004.

4. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования, 3-е издание. Вильямс, 2007.

5. Гайд по разработке приложений на Spring - https://spring.io/guides

Приложение

```
package com.kspt.buro;

@SpringBootApplication
public class Application {
   public static void main(String[] args) {  SpringApplication.run(Application.class, args); }
}
```

```
package com.kspt.buro.service;

@Service
public class UserService implements UserDetailsService {

   @Autowired
   private RequestRepo requestRepo;

   @Autowired
   private UserRepo userRepo;

   @Override
   public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
      User user = userRepo.findByUsername(username);
      if (user == null) throw new UsernameNotFoundException("Пользователя не
существует");
      return user;
   }

   public List<User> findAll() {  return userRepo.findAll(); }

   public boolean addUser(User user) {
      User userFromDb = userRepo.findByUsername(user.getUsername());

      if (userFromDb != null)   return false;

      user.setActive(true);
      user.setRoles(Collections.singleton(Role.USER));
      userRepo.save(user);
      return true;
   }

   public String updateUsername(User user, String username, String url) {
      if (username.equals("admin"))   return "redirect:/user/" + url;
      if (user.getUsername().equals(username))   return "ok";

      for (User u : findAll())
         if (u.getUsername().equals(username))   return "redirect:/user/" + url;

      user.setUsername(username);
      return "ok";
   }
```

```
    public void updateRoles(User user, Map<String, String> form) {
        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));
    }

    public void deleteUser(String username) {
        Iterable<Request> requests = requestRepo.findAll();
        for (Request request : requests) {
            if
(request.getAuthorName().equals(userRepo.findByUsername(username).getUsername()))
                requestRepo.delete(request);
        }

        userRepo.delete(userRepo.findByUsername(username));
    }

    public void updatePassword(User user, String password) {
        if (!password.trim().isEmpty())
            user.setPassword(password.trim());
    }
}
```

```
package com.kspt.buro.repos;

public interface RequestRepo extends CrudRepository<Request, Long> {

    List<Request> findByPtype(String ptype);
    List<Request> findByAuthor(User user);
}
```

```
package com.kspt.buro.repos;

public interface UserRepo extends JpaRepository<User, Long> {
    User findByUsername (String username);
}
```

```
package com.kspt.buro.domain;

public enum Checks implements GrantedAuthority {
    CHECK, VERIFIED, PAID;

    @Override
    public String getAuthority() { return name(); }
}
```

```
package com.kspt.buro.domain;

public enum Role implements GrantedAuthority {
    USER, ADMIN, VERIF;

    @Override
    public String getAuthority() { return name(); }
}
```

```
package com.kspt.buro.domain;

@Entity
public class Request {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @NotBlank(message = "Поле должно быть заполнено")
    @Length(max = 2048,message = "Описание слишком большое")
    private String text;

    @NotBlank(message = "Поле должно быть заполнено")
    @Length(max = 255,message = "Описание слишком большое")
    private String ptype;

    @ManyToOne(fetch = FetchType.EAGER)
    @JoinColumn(name = "user_id")
    private User author;
    private String filename;

    @ElementCollection(targetClass = Checks.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "request_checks", joinColumns = @JoinColumn(name =
"request_id"))
    @Enumerated(EnumType.STRING)
    private Set<Checks> checks;

    private String message;

    public Request() { }
    public Request(String text, String ptype, User user) {
        this.author = user;
        this.text = text;
        this.ptype = ptype;
    }

    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 getPtype() { return ptype; }
    public void setPtype(String ptype) { this.ptype = ptype; }
```

```
    public User getAuthor() { return author; }
    public void setAuthor(User author) { this.author = author; }
    public String getFilename() { return filename; }
    public void setFilename(String filename) { this.filename = filename; }

    public Set<Checks> getChecks() { return checks; }
    public void setChecks(Set<Checks> checks) { this.checks = checks; }
    public String getMessage() { return message; }
    public void setMessage(String message) { this.message = message; }

    public String getAuthorName(){  return  author != null ? author.getUsername() : "<none>";  }
}
```

```
package com.kspt.buro.domain;

@Entity
@Table(name = "usr")
public class User implements UserDetails{

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @NotBlank(message = "Поле должно быть заполнено")
    private String username;
    @NotBlank(message = "Поле должно быть заполнено")
    private String password;
    private boolean active;

    @ElementCollection(targetClass = Role.class, fetch = FetchType.EAGER)
    @CollectionTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"))
    @Enumerated(EnumType.STRING)
    private Set<Role> roles;

    @OneToMany(mappedBy = "author", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private Set<Request> requests;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    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 Set<Request> getRequests() { return requests; }
    public void setRequests(Set<Request> requests) { this.requests = requests; }

    public String getRole() { return roles.iterator().next().toString(); }
```

```
    public Boolean isAdmin(){ return roles.contains(Role.ADMIN); }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id);
    }
    @Override
    public int hashCode() { return Objects.hash(id); }


    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() { return getRoles(); }
    @Override
    public boolean isAccountNonExpired() {  return true; }
    @Override
    public boolean isAccountNonLocked() { return true; }
    @Override
    public boolean isCredentialsNonExpired() { return true; }
    @Override
    public boolean isEnabled() { return isActive(); }
}
```

```
package com.kspt.buro.controller;

@Controller
public class MainController {

    @Value("${upload.path}")
    private String uploadPath;

    @Autowired
    private RequestRepo requestRepo;

    @GetMapping("/")
    public String greeting(Model model) {
        String auth = SecurityContextHolder.getContext().getAuthentication().getName();

        if (auth.equals("anonymousUser")) model.addAttribute("user", "гость");
        else model.addAttribute("user", auth);

        return "greeting";
    }

    @GetMapping("/main")
    public String main(
            @AuthenticationPrincipal User user,
            @RequestParam(required = false, defaultValue = "") String filter,
            Model model) {
        Iterable<Request> requests;
```

```
        if (user.getRole().equals("USER")) {
            requests = filter != null && !filter.isEmpty() ?
                    requestRepo.findByPtype(filter) : requestRepo.findByAuthor(user);
        } else requests = filter != null && !filter.isEmpty() ?
                requestRepo.findByPtype(filter) : requestRepo.findAll();
        model.addAttribute("requests", requests);
        model.addAttribute("checks", Checks.values());
        model.addAttribute("filter", filter);
        model.addAttribute("user", user);
        return "main";
    }

    @PostMapping("/main")
    public String add(
            @AuthenticationPrincipal User user,
            @Valid Request request,
            BindingResult bindingResult,
            Model model,
            @RequestParam("file") MultipartFile file) throws IOException {

        request.setAuthor(user);
        if (bindingResult.hasErrors()) {
            Map<String, String> errorsMap = ControllerUtils.getErrors(bindingResult);
            model.mergeAttributes(errorsMap);
            model.addAttribute("request", request);
        } else if (file == null || file.getOriginalFilename().isEmpty())
            model.addAttribute("fileError", "Файл нужно выбрать");
        else {
            ControllerUtils.saveFile(uploadPath, request, file);
            model.addAttribute("request", null);
            requestRepo.save(request);
        }
        model.addAttribute("user", user);
        model.addAttribute("checks", Checks.values());
        model.addAttribute("requests", requestRepo.findByAuthor(user));
        return "main";
    }

    @GetMapping(value = "/main", params = "delete")
    public String delete(
            @RequestParam(required = false, defaultValue = "") String filter,
            Model model) {

        model.addAttribute("requests", filter != null && !filter.isEmpty() ?
                requestRepo.findByPtype(filter) : requestRepo.findAll());
        model.addAttribute("filter", filter);
        model.addAttribute("user",
SecurityContextHolder.getContext().getAuthentication().getName());
        return "main";
    }
}
```

```
package com.kspt.buro.controller;

@Controller
public class RegistrationController {

    @Autowired
    private UserService userService;

    @GetMapping("/registration")
    public String registration() {
        return "registration";
    }

    @PostMapping("/registration")
    public String addUser(
            @RequestParam String password2,
            @Valid User user,
            BindingResult bindingResult,
            Model model) {

        if (user.getPassword() != null && !user.getPassword().equals(password2)) {
            model.addAttribute("passwordError", "Пароли не совпадают");
            model.addAttribute("password2Error", "Пароли не совпадают");
            return "registration";
        }

        if (bindingResult.hasErrors()) {
            Map<String, String> errors = ControllerUtils.getErrors(bindingResult);
            model.mergeAttributes(errors);
            return "registration";
        }

        if (!userService.addUser(user)) {
            model.addAttribute("usernameError", "Этот логин занят");
            return "registration";
        }

        return "redirect:/login";
    }
}
```

```
package com.kspt.buro.controller;

@Controller
public class RequestController {

    @Value("${upload.path}")
    private String uploadPath;

    @Autowired
    private RequestRepo requestRepo;
```

```
@GetMapping("/user-requests/{user}")
public String userRequests(
        @AuthenticationPrincipal User currentUser,
        @PathVariable User user,
        Model model,
        @RequestParam(required = false) Request request) {

    model.addAttribute("user", user);
    model.addAttribute("requests", Collections.singleton(request));
    model.addAttribute("checks", Checks.values());
    model.addAttribute("request", request);
    model.addAttribute("isCurrentUser", currentUser.equals(user));
    return "userRequests";
}

@PostMapping("/user-requests/{user}")
public String updateRequest(
        @AuthenticationPrincipal User currentUser,
        @PathVariable Long user,
        @RequestParam("id") Request request,
        @RequestParam("text") String text,
        @RequestParam("ptype") String ptype,
        @RequestParam("file") MultipartFile file) throws IOException {

    assert request != null;
    if (request.getAuthor().equals(currentUser)) {
        if (!StringUtils.isEmpty(text)) request.setText(text);
        if (!StringUtils.isEmpty(ptype)) request.setPtype(ptype);
        ControllerUtils.saveFile(uploadPath, request, file);
        requestRepo.save(request);
    }
    return "redirect:/main";
}

@GetMapping("/check")
public String checkRequestEdit(
        @AuthenticationPrincipal User user,
        Model model,
        @RequestParam(required = false) Request request) {

    model.addAttribute("user", user);
    model.addAttribute("request", request);
    model.addAttribute("checks", Checks.values());
    return "checkRequest";
}

@PostMapping("/check")
public String checkRequestSave(
        @AuthenticationPrincipal User user,
        @RequestParam("requestId") Request request,
        @RequestParam String message,
        @RequestParam Map<String, String> form) {
```

```
        Set<String> checks = Arrays.stream(Checks.values())
                .map(Checks::name)
                .collect(Collectors.toSet());
        request.getChecks().clear();
        for (String key : form.keySet())
           if (checks.contains(key)) request.getChecks().add(Checks.valueOf(key));
        request.setMessage(message+"\n");
        requestRepo.save(request);
        return "redirect:/main";
    }
}
```

```
package com.kspt.buro.controller;

@Controller
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserRepo userRepo;

    @Autowired
    private RequestRepo requestRepo;

    @Autowired
    private UserService userService;

    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping
    public String userList(Model model) {
        model.addAttribute("users", userService.findAll());
        return "userList";
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping("{user}")
    public String userEditForm(@PathVariable User user, Model model) {
        model.addAttribute("user", user);
        model.addAttribute("roles", Role.values());
        return "userEdits";
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @PostMapping(params = "save")
    public String userSave(
            @RequestParam String username,
            @RequestParam Map<String, String> form,
            @RequestParam("userId") User user) {

        String newUsername = userService.updateUsername(user, username.trim(),
user.getId().toString());
```

```
        if (!newUsername.equals("ok")) return newUsername;

        userService.updateRoles(user, form);

        userRepo.save(user);

        return "redirect:/user";
    }

    @PreAuthorize("hasAuthority('ADMIN')")
    @PostMapping(params = "delete")
    public String delete(@RequestParam String username) {

        userService.deleteUser(username);

        return "redirect:/user";
    }

    @GetMapping("profile")
    public String getProfile(Model model) {

        model.addAttribute("user",
userService.loadUserByUsername(SecurityContextHolder.getContext().getAuthentication().getN
ame()));

        return "profile";
    }

    @PostMapping("profile")
    public String updateProfile(
            @RequestParam(defaultValue = "") String username,
            @RequestParam String password,
            @RequestParam("userId") User user) {

        if (!user.getUsername().equals("admin")) {
            String newUsername = userService.updateUsername(user, username.trim(), "profile");
            if (!newUsername.equals("ok")) return newUsername;
        }

        userService.updatePassword(user, password);

        userRepo.save(user);

        return "redirect:/user/profile";
    }
}
```

```
package com.kspt.buro.controller;

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);
    }

    static void saveFile(String uploadPath, Request request, MultipartFile file) throws
IOException {

        if (file != null
                && !file.getOriginalFilename().isEmpty()) {
            File uploadDir = new File(uploadPath);

            if (!uploadDir.exists()) uploadDir.mkdir();

            String uuidFile = UUID.randomUUID().toString().substring(0, 5);
            String resultFilename = uuidFile + "." + file.getOriginalFilename();

            file.transferTo(new File(uploadPath + "/" + resultFilename));
            request.setFilename(resultFilename);
        }
    }
}
```

```
package com.kspt.buro.config;

@Configuration
public class MvcConfig implements WebMvcConfigurer {

    @Value("${upload.path}")
    private String uploadPath;

    public void addViewControllers(ViewControllerRegistry registry) {
        registry.addViewController("/login").setViewName("login");
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/file/**")
                .addResourceLocations("file://"+uploadPath+"/");

        registry.addResourceHandler("/static/**")
                .addResourceLocations("classpath:/static/");
    }
}
```

```
package com.kspt.buro.config;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

   @Autowired
   private UserService userService;

   @Override
   protected void configure(HttpSecurity http) throws Exception {
      http
            .authorizeRequests()
               .antMatchers("/","/registration", "/static/**")
                  .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(NoOpPasswordEncoder.getInstance());
   }
}
```