Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Национальный исследовательский университет

«Высшая школа экономики»

Факультет информатики, математики и компьютерных наук

Образовательная программа «Программная инженерия»

**Курсовая работа**

**По дисциплине:**

**«Программирование»**

**На тему:**

**Реализация игрового сервера для подсчёта и отображения рейтинга игроков**

Выполнил:

студент 1 курса группы 19ПИ-1

направления подготовки «Программная инженерия»

Стоянов С. С.

Научный руководитель: доцент

Шаров Г. В.

Нижний Новгород 2020

**Оглавление**

[1. Введение 3](#_Toc41741531)

[2. Техническое задание 4](#_Toc41741532)

[2.1 Описание задачи 4](#_Toc41741533)

[2.2 Формат результата 5](#_Toc41741534)

[3. Обзор существующих решений 5](#_Toc41741535)

[3.1 Описание проблемы существующих решений 6](#_Toc41741536)

[4. Описание решения 6](#_Toc41741537)

[4.1 Логика приложения 8](#_Toc41741538)

[4.2 Структура проекта 15](#_Toc41741539)

[5. Описание приложения 16](#_Toc41741540)

[6. Заключение 18](#_Toc41741541)

[7. Список используемой литературы, документации и источников 19](#_Toc41741542)

# 1. Введение

Настоящая курсовая работа посвящена разработке сервера для его последующего использования в практических целях.

Актуальность исследования связана с продолжающимся развитием интернета, а именно игровой индустрии, которая переходит из оффлайна в онлайн сектор и нуждается в серверных решениях для вычисления, подсчёта, хранения и отображения рейтинга игроков.

**Проблематика** состоит в написании сервера для подсчёта рейтинга.

**Цель** настоящей работы состоит в создании сервера для подсчёта, хранения и отображения рейтинга игроков, а также изучение паттернов разработки, фреймворков и библиотек.

**Новизна** исследования заключается в реализации подсчёта и хранения рейтинга игроков.

Таким образом, **объектом** настоящей курсовой работы является игровой сервер, тогда как **предметом** исследования является разработка веб-сервиса.

Следовательно, выделяются **задачи**, реализация которых необходима для выполнения работы:

1. Написание сервера
   1. Реализация функций регистрации и авторизации
   2. Реализация алгоритмов для работы с базой данных
   3. Создание REST протокола
2. Реализация личного кабинета для отображения рейтинга
3. Имплементация интерфейса для загрузки достижений

# 2. Техническое задание

## 2.1 Описание задачи

Сервер, в данном случае, рассматривается как программный компонент, выполняющий сервисные функции по запросу клиента, предоставляя ему доступ к определённым ресурсам или услугам.

Серверное ПО будет состоять из двух основных модулей.

1. Реализует backend решение для регистрации, авторизации и отображения рейтинга в личном кабинете игрока
2. Реализует интерфейс для работы пользователя с базой данных

Большинство веб-ресурсов создаются с помощью языка программирования PHP. Он прост в освоении и позволяет писать код прямо в html-странице, что импонирует многим разработчикам. Но такой подход многократно осложняет поддержку и модернизацию программного продукта. Чтобы избежать этих недостатков, для создания настоящего клиент-серверного приложения был выбран подход паттерна MVC. Согласно мировому опыту в разработке серверного ПО, лучшая практика – придерживаться общепринятых принципов программирования, таких как SOLID который повышает возможность повторного использования кода, декларируют принцип единственной ответственности, инверсии зависимостей, открытости\закрытости и собственно **MVC** – паттерн разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо

Наиболее эффективно применение концепции MVC в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах или с различных точек зрения – что в полной мере описывает принцип использования игрового сервера. Поэтому в процессе разработки я буду придерживаться паттерна MVC, так это оставляет пространство для добавления новых функций без существенного изменения кода.

Философия паттерна MVC представляет собой совокупность моделей, контроллеров и представлений. При этом модель никаким образом не связана с представлением, она занимается лишь хранением данным и не визуализирует их. Представление, наоборот, визуализирует какие-либо данные пользователю. Представление не обрабатывает введённые данные пользователя. Контроллер является связующим звеном между моделью и представлением. Он принимает данные из модели в представление и отправляет данные из модели в представление. Контроллер — это связующее звено между моделью и представлением для реализации необходимого действия.

## 2.2 Формат результата

Результатом должен быть готовый для развёртывания на любой операционной системе веб-сервер,

А также исходя из **целей**:

1. Реализована система авторизации и регистрации
2. Реализован сервис для взаимодействия приложения с базовой данных
3. Реализован личный кабинет для отображения рейтинга
4. Реализован REST протокол
5. Имплементирован интерфейс для загрузки рейтинга

# 3. Обзор существующих решений

Рассмотрим два примера решения для веб-серверов

1. REST сервер написанный на Python (URL: <https://github.com/Mregussek/rest-api-server> )

Это REST сервер, возвращающий JSON объект и используемый для получения записей о книгах – названии, редакции и авторе.

В нём реализованы 4 функции:

* GET – выполняет поиск запрошенного идентификатора в списке книг и возвращает данные, если они найдены вместе с кодом ответа HTTP 200. В противном случае HTTP 404 – не найден
* POST – вставляет новые данные книги в список книг и возвращает вставленные данные с созданным кодом ответа HTTP 201. Если запись уже существует, она возвращает код ошибки HTTP 400.
* PUT – перезаписывает запись и возвращает данные вместе с кодом ответа HTTP 200. Если запись не существует, он создаёт данные и возвращает их с созданным кодом ответа HTTP 201.
* DELETE – удаляет запись, если она существует, и возвращает данные с кодом ответа HTTP 200. В противном случае HTTP 404.

1. REST сервер написанный на C++ (URL: <https://github.com/bracoe/REST_server> )

Это REST сервер, возвращающий JSON объект и используемый для получения записей о температуре и влажности, и реализует те же функции:

* GET – выполняет поиск запрошенного идентификатора в списке книг и возвращает данные, если они найдены вместе с кодом ответа HTTP 200. В противном случае HTTP 404 – не найден
* POST – вставляет новые данные книги в список книг и возвращает вставленные данные с созданным кодом ответа HTTP 201. Если запись уже существует, она возвращает код ошибки HTTP 400.
* PUT – перезаписывает запись и возвращает данные вместе с кодом ответа HTTP 200. Если запись не существует, он создаёт данные и возвращает их с созданным кодом ответа HTTP 201.
* DELETE – удаляет запись, если она существует, и возвращает данные с кодом ответа HTTP 200. В противном случае HTTP 404.

## 3.1 Описание проблемы существующих решений

Прежде всего, эти решения сложно поддерживать, изменять и дополнять. Особенно это касается языка Python. Во-первых, он динамически типизирован, а значит и риск ошибки выше, чем в коде, который статически типизирован, а во-вторых – его стиль написания предполагает процедурный стиль, что уменьшает читаемость кода – и соответственно ухудшает поддержку и расширение. Python на уровне ядра не поддерживает многопоточность, поэтому необходимо запускать столько экземпляров приложения, сколько ядер у процессора. Решения на С++ слишком низкоуровневые, и на их написание затрачивается много времени. Сервера на C++ зависят от платформы, на которой они работают, поэтому необходимо знание библиотек под конкретную платформу разработки.

Во-вторых, отсутствует интерфейс для пользователя для работы с результатами – просмотр и добавление.

# 4. Описание решения

Основным языком разработки была выбрана Java 11 версии вследствие того, что Java предоставляет платформу для удобной платформо-независимой разработки с использованием парадигмы ООП, с помощью которой легко реализовывать подход MVC. Системой сборки, компиляции и управлениями зависимостями был выбран Gradle. Разработка ведётся в продуктах компании JetBrains – бэкенд решения IntelliJ IDEA Ultimate, фронтенд решения – в WebStrom, для работы с базой данных – DataGrip. Тестирование REST API производится с помощью Postman.

Сервер будет строиться с использованием фреймворка Spring, который предоставляет множество библиотек для построения микро-сервисной архитектуры. Будут использованы модули Boot, Security и Framework.

Spring Boot это среда, используемая для создания микросервисов, предоставляющая единую точку входа в приложение и возможности для удобной конфигурации.

1. package edu.ssstoyanov.webserver;
3. import org.springframework.boot.SpringApplication;
4. import org.springframework.boot.autoconfigure.SpringBootApplication;
6. @SpringBootApplication
7. public class Application {
8. public static void main(String[] args) {
9. SpringApplication.run(Application.class, args);
10. }
11. }

Данный фрагмент кода позволяет без дополнительных конфигураций создать сервер с уже готовой страницей приветствия.

Spring Security это инструмент для аутентификации и контроля доступа и де-факто стандарт для защиты приложений на базе Spring Boot. Он требует только настройку разграничения доступа и берет на себя функцию шифрования паролей, сессии и защиту от несанкционированного доступа.

1. @Override
2. protected void configure(HttpSecurity http) throws Exception {
3. http
4. .authorizeRequests()
5. .antMatchers("/", "/main", "/signin", "/signup", "/error", "/rest/api/\*\*").permitAll()
6. .antMatchers("/homepage/\*\*").hasAnyAuthority("USER", "ADMIN")
7. .antMatchers("/admin/\*\*").hasAuthority("ADMIN")
8. .anyRequest().authenticated()
9. .and()
10. .formLogin().loginPage("/signin").failureUrl("/signin?error=true")
11. .defaultSuccessUrl("/homepage", true)
12. .usernameParameter("username").passwordParameter("password")
13. .and()
14. .logout()
15. .logoutUrl("/logout")
16. .logoutSuccessUrl("/signin?logout=true")
17. .invalidateHttpSession(true)
18. .permitAll()
19. .and()
20. .exceptionHandling()
21. .and().csrf().disable();
22. }

Данный метод в классе WebSecurityConfiguration отвечает за разграничение доступа.

Spring Framework – платформа для веб-приложений, с помощью которой реализуется front-end части приложения, тестирование и работа с базой данных.

Для упрощения работы с базой данных используется встроенная в Spring Framework библиотека Hibernate реализующая принципы JPA, и заменяющая низкоуровневый JDBC[[1]](#footnote-1). Таким образом я исключил ошибки при работе с SQL-запросами, делегировав эту задачу библиотеке. Так же я использовал библиотеку Lombok, которая уменьшает количество шаблонного кода и улучшает читаемость сущностей. В качестве базы данных была выбрана PostgreSQL, как одна из самых популярных и удобных для использования среди прочих баз данных.

Для написания фронтенда используются язык разметки HTML5, CSS и JavaScript, а также библиотеки Bootstrap и JQuery. Thymeleaf используется как нативный для Spring шаблонизатор, который работает c HTML5 и подходит для обслуживания на уровне представления (View) веб-приложений на основе MVC и Security.

## 4.1 Логика приложения

Схема таблиц в базе данных, связанной с сервером, выглядит следующим образом.

Изображение выглядит как снимок экрана

Автоматически созданное описание

Для реализации авторизации и регистрации были созданы сущности User и Role, которые отвечают за данные пользователя и его статус в системе. Эти сущности связаны с объектами в базе данных и являются их представлением в коде. За связь и взаимодействие сущностей с базой данных отвечают репозитории UserRepository и RoleRepository. Сервисы UserService и UserHandlingService используются для нужд контроллеров предоставляя им функции поиска пользователя по имени, шифрование паролей, установку ролей и получение списка разрешённых действий.

Сущность User представляет собой отображение пользователя в базе данных, а сущность Role – роль пользователя в системе.

1. @Data
2. @Builder
3. @AllArgsConstructor
4. @NoArgsConstructor
5. @Entity
6. @Table(name = "users")
7. public class User {
8. @Id
9. @GeneratedValue(strategy = GenerationType.AUTO)
10. @Column(name = "user\_id")
11. private Integer id;
12. @Column(name = "user\_name")
13. @Length(min = 5, max = 32, message = "Имя пользователя должно быть диапазоне от 5 до 32 символов")
14. @NotEmpty(message = "Пожалуйста, введите имя пользователя")
15. private String username;
16. @Column(name = "password")
17. @Length(min = 5, message = "Пароль должен состоять хотя бы 5 символов")
18. @NotEmpty(message = "Пожалуйста, введите пароль")
19. private String password;
20. @ManyToMany(cascade = CascadeType.MERGE)
21. @JoinTable(name = "user\_role",
22. joinColumns = @JoinColumn(name = "user\_id"),
23. inverseJoinColumns = @JoinColumn(name = "role\_id"))
24. private Set<Role> roles;
25. @OneToMany(mappedBy = "user", fetch = FetchType.EAGER)
26. private List<Record> record;
27. }
28. @Data
29. @Builder
30. @AllArgsConstructor
31. @NoArgsConstructor
32. @Entity
33. @Table(name = "roles")
34. public class Role {
35. @Id
36. @GeneratedValue(strategy = GenerationType.AUTO)
37. @Column(name = "role\_id")
38. private int id;
39. @Column(name = "role")
40. private String role;
41. }

Таблицы users и roles связаны между собой таблицей user\_role через JoinTable по полям user\_id и role\_id.

Сервисы UserService и UserHandlingService реализуют работу со Spring Security и предоставляют методы для работы с сущностями.

1. @Service
2. public class UserService {
4. @Autowired
5. private UserRepository userRepository;
7. @Autowired
8. private RoleRepository roleRepository;
10. @Autowired
11. private BCryptPasswordEncoder bCryptPasswordEncoder;
13. public User findUserByUserName(String userName) {
14. return userRepository.findByUsername(userName);
15. }
17. public void saveUser(User user) {
18. user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
19. user.setRoles(new HashSet<>(Collections.singletonList(roleRepository.findByRole("USER"))));
20. userRepository.save(user);
21. }
23. }
24. @Service
25. public class UserHandlingService implements UserDetailsService {
27. @Autowired
28. private UserService userService;
30. @Override
31. @Transactional
32. public UserDetails loadUserByUsername(String userName) {
33. User user = userService.findUserByUserName(userName);
34. List<GrantedAuthority> authorities = getUserAuthority(user.getRoles());
35. return buildUserForAuthentication(user, authorities);
36. }
38. private List<GrantedAuthority> getUserAuthority(Set<Role> userRoles) {
39. Set<GrantedAuthority> roles = new HashSet<>();
40. for (Role role : userRoles) {
41. roles.add(new SimpleGrantedAuthority(role.getRole()));
42. }
43. return new ArrayList<>(roles);
44. }
46. private UserDetails buildUserForAuthentication(User user, List<GrantedAuthority> authorities) {
47. return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(),
48. true, true, true, true, authorities);
49. }
50. }

Для работы с записями достижений пользователей создана сущность Record, с репозитарием RecordRepository и сервисом RecordService который предоставляет загрузку записей по имени пользователя и сохранения записи в базе данных.

1. @Data
2. @Builder
3. @AllArgsConstructor
4. @NoArgsConstructor
5. @Entity
6. @Table(name = "records")
7. public class Record {
8. @Id
9. @GeneratedValue(strategy = GenerationType.AUTO)
10. @Column(name = "record\_id")
11. private Integer id;
12. @Column(name = "game")
13. private String game;
14. @Column(name = "score")
15. private Long score;
16. @ManyToOne(optional = false, cascade = CascadeType.ALL)
17. @JoinColumn(name = "user\_id")
18. private User user;
19. }

Класс RecordService выполняет транзакционную работу с базой данных используя методы из RecordRepository.

1. @Service
2. public class RecordService {
4. @Autowired
5. private RecordRepository recordRepository;
7. @Autowired
8. private UserService userService;
10. @Transactional
11. public void saveRecordFromGamesPost(Record record) {
12. User user = userService.findUserByUserName(SecurityContextHolder.getContext().getAuthentication().getName());
13. record.setUser(user);
14. record.setId(null);
15. recordRepository.saveAndFlush(record);
16. }
18. @Transactional
19. public Record saveRecordFromPost(Record record) {
20. return recordRepository.saveAndFlush(record);
21. }

24. public HashMap<String, Long> getRecordsByUser(User user) {
25. HashMap<String, Long> data = new HashMap<>();
26. List<Record> recordList = recordRepository.getAllByUser(user);
27. for (Record record : recordList) {
28. data.putIfAbsent(record.getGame(), record.getScore());
29. }
30. return data;
31. }
33. public List<Record> getAllRecords() {
34. return recordRepository.findAll();
35. }
37. public List<Record> getAllRecordsByUsername(String username) {
38. return recordRepository.getByUser(userService.findUserByUserName(username));
39. }
41. }

Классы GamesController, HomepageController, RecordsController, SigninController и SignupController реализуют сущность контроллеров и работают промежуточным звеном между моделью и отображением с использованием вышеупомянутых сервисов. В них нет существенных технических особенностей.

С помощью Thymeleaf были написаны страницы для регистрации, авторизации, личный кабинет, главная страница, домашняя страница, страницы отображения и загрузки результатов, а также заглушка для админ-панели.

Контроль страницы входа происходит без участия определенного контроллера и прописан в WebSecuriyConfiguration.

1. .formLogin().loginPage("/signin").failureUrl("/signin?error=true")
2. .defaultSuccessUrl("/homepage", true)
3. .usernameParameter("username").passwordParameter("password")

 Который связан со следующим HTML5 файлом signin.html

1. <div class="limiter">
2. <div class="container-login100">
3. <div class="wrap-login100">
4. <form class="login100-form validate-form p-l-55 p-r-55 p-t-178"
5. th:action="@{/signin}" method="post">
6. <span class="login100-form-title">Вход</span>
7. <div class="wrap-input100 validate-input m-b-16" data-validate="Введите имя пользователя">
8. <input class="input100" type="text" id="username" name="username" placeholder="Имя пользователя">
9. <span class="focus-input100"></span>
10. </div>
11. <div class="wrap-input100 validate-input" data-validate="Введите пароль">
12. <input class="input100" type="password" id="password" name="password" th:placeholder="Пароль">
13. <span class="focus-input100"></span>
14. </div>
15. <div align="center" th:if="${param.error}">
16. <p style="font-size: 20; color: #FF1C19;">Имя пользователя или пароль не верны, повторите попытку</p>
17. </div>
18. <div align="center" th:if="${param.logout}">
19. Вы вышли из аккаунта
20. </div>
21. <div class="text-right p-t-13 p-b-23"></div>
22. <div class="container-login100-form-btn">
23. <button class="login100-form-btn"
24. name="Submit" value="Login" type="Submit"
25. th:text="Войти">
26. </button>
27. </div>
28. <div class="flex-col-c p-t-170 p-b-40">
29. <span class="txt1 p-b-9">Нет аккаунта?</span>
30. <a href="signup" class="txt3">Регистрация</a>
31. </div>
32. </form>
33. </div>
34. </div>
35. </div>

REST часть приложения находится в модуле Restful и состоит из одного файла RecordRestful. В нём реализованы Get, Post и Delete запросы, позволяющие получать в JSON виде все записи о достижениях в принципе, по имени пользователя и по названию игры, принимать данные о результате пользователя и удалять все связанные результаты по названию игры.

1. @RequestMapping("/rest/api")
2. @RequiredArgsConstructor
3. @RestController()
4. public class RecordRestful {
6. @Autowired
7. private final RecordService recordService;
9. @Autowired
10. private final UserService userService;
12. @GetMapping("/records")
13. public List<Record> getAl() {
14. return recordService.getAllRecords();
15. }
17. @GetMapping("/records/username/{username}")
18. public List<Record> getRecordsByUsername(@PathVariable(value = "username") String username) {
19. return recordService.getAllRecordsByUsername(username);
20. }
22. @GetMapping("/records/game/{game}")
23. public List<Record> getRecordsByGame(@PathVariable(value = "game") String game) {
24. return recordService.getAllRecordsByGame(game);
25. }
27. @PostMapping(value = "/records")
28. public ResponseEntity<String> addRecord(String username, String game, Long score) {
29. User user = userService.findUserByUserName(username);
30. if (user == null) {
31. return new ResponseEntity<>("User not found " + username, HttpStatus.NOT\_FOUND);
32. } else {
33. recordService.saveRecordFromPost(new Record(0, game, score, user));
34. return new ResponseEntity<>(HttpStatus.CREATED);
35. }
36. }
38. @DeleteMapping("/records/game")
39. public ResponseEntity<String> deleteRecordsByGame(String game) {
40. Integer byGame = recordService.deleteAllRecordsByGame(game);
41. if (byGame != 0) {
42. return new ResponseEntity<>(HttpStatus.OK);
43. } else {
44. return new ResponseEntity<>("Game not found " + game, HttpStatus.NOT\_FOUND);
45. }
46. }
47. }

На этом можно завершить основную логику приложения и перейти структуре проекта.

## 4.2 Структура проекта

Приложение имеет модульную структуру, которая управляется с помощью Spring аннотаций таких как @Entity, @Controller, @Service, @Autowired, @Repository, @Transactional и других. Точка входа находится в файле Application, который запускает необходимые сервисы для работы сервера. Далее подключаются классы из директорий Config, Controller, Model, Repository и Service. Из директории resources загружаются конфигурационные файлы, а также статичные и шаблонные файлы для отображения (View).

Таким образом приложение реализует концепцию MVC:

1. Model

Созданы модели User, Record и Role, работа с которым реализуется через сервисные классы RecordService, UserHandingService и UserService использующие соответствующие JPA репозитории моделей и конфигурационные классы безопасности SecurityConfiguration и WebSecurityConfiguration.

1. Controller

Контроллеры реализуют промежуточную работу используя сервис классы и декларируют View модель.

1. HomepageController использует UserService и декларирует логику страниц "/main", "/admin", "/homepage" и "/".
2. SignInController использует WebSecurityConfiguration и декларирует логику страницы "/signin".
3. SignUpController использует UserService и декларирует логику страницы "/signup".
4. RecordsController использует RecordService и UserRepository, декларируя логику страницы "/records".
5. GamesController использует RecordService декларируя логику страницы "/games".
6. View

Отображение получает данные от контроллеров и никак не участвуют в обработке данных. В проекте View модели — это файлы с расширением HTML, которые задействуют библиотеку Thymeleaf. Это файлы admin.html, games.html, homepage.html, records.html, signin.html и signup.html.

# 5. Описание приложения

Готовый продукт представляет собой Java-приложение с PostreSQL базой данных. Интерфейс приложения представляет собой сайт, находящийся по адресу сервера на порту 8080. Следующая блок-схема описывает логику работы веб-интерфейса:

Изображение выглядит как снимок экрана

Автоматически созданное описание

Переход по адресу сервера с портом 8080 пересылает на главную страницу, которая так же находится по адресу “/main”. На ней присутствуют две опции на вход и регистрацию. Между ними находятся перекрёстные ссылки. После успешного входа в систему происходит пересылка на домашнюю страницу по адресу “/homepage” где есть опция просмотра рейтинга, добавления и выхода из системы.

REST сервис выполняет POST, GET и DELETE запросы, которые реализуются через HTTP и возвращают JSON объекты. Реализованы следующие запросы:

1. GET
   1. /records – возвращает JSON всех результатов
   2. /records/username/{username} – возвращает JSON всех результатов игрока по никнейму
   3. /records/game/{game} – возвращает JSON всех результатов по названию игры
2. POST
   1. /records?username={username}&game={game}&score={score} – добавляет запись о результате игрока в базу данных
3. DELETE
   1. /records/game?game={game} – удаляет все результаты по игре

Например, следующий запрос:

1. http://localhost:8080/rest/api/records/username/Stanislav

 Вернёт следующий JSON ответ

1. [{"id":2,"game":"Fortnite","score":7367},{"id":5,"game":"Cyberpunk 2077","score":73568},{"id":7,"game":"Call of Duty: Warzone","score":386}]

# 6. Заключение

Таким образом мною был написан веб-сервер для подсчёта и отображения рейтинга игроков с использованием языка программирования Java и фреймворка Spring.

Проект покрывает техническое задание на минимальном уровне и нуждается в существенных доработках как по дизайну, так и по логической части. Далее планируется:

1. Реализовать авторизационные токены для REST API
2. Покрытие юнит-тестами
3. Доработка веб-интерфейса

Таким образом я научился работать с языком программирования Java на высоком уровне, изучил основы Spring Framework, Security и MVC. Изучил работу HTTP протокола и концепцию REST API. На достаточном уровне изучил библиотеки Hibernate и Lombok. Получил базовые знания по HTML5, Javascript, CSS, JQuery, Bootstrap и Thymeleaf. Научился работать по методу канбан.

Исходный код проекта хранится на репозитории <https://github.com/ssstoyanov/WebServer>.

# 7. Список используемой литературы, документации и источников

**Научные статьи**

1. Глод О.Д., Чекулаев А.А. Разработка веб - приложения с использованием Spring Framework и jQuery // ИВД. 2018. №4 (51). URL: https://cyberleninka.ru/article/n/razrabotka-veb-prilozheniya-s-ispolzovaniem-spring-framework-i-jquery (дата обращения: 14.04.2020)

**Интернет-источники и документация**

1. Урванов Ф. Project Lombok [Электронный ресурс]: URL: https://urvanov.ru/2015/09/22/project-lombok/ (дата обращения 10.04.2020)
2. PostgreSQL: Documentation: 9.6: Documentation [Электронный ресурс]: URL: https://www.postgresql.org/docs/9.6/docguide.html (дата обращения 17.04.2020)
3. Spring Framework Documentation [Электронный ресурс]: URL: https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html (дата обращения 05.04.2020)
4. Hibernate ORM 5.1 User Guide [Электронный ресурс]: URL: https://docs.jboss.org/hibernate/orm/5.1/userguide/html\_single/Hibernate\_User\_Guide.html (дата обращения 10.04.2020)

**Книги**

1. Хорстманн, Кей С., Корнелл, Гари Java. Библиотека профессионала, том 2. Расширенные средства, 9-е изд.: Пер. с англ. — М.: (3 0 0 "И. Д. Вильямс", 2014. — 1008 с.: ил. — Парал. тит. англ. ISBN 978-5-8459-1870-3 (рус., том 2)
2. Бауэр К., Кинг Г., Грегори Г. Java Persistence API и Hibernate / пер. с англ. Д. А. Зинкевича; под науч. ред. А. Н. Киселева. – М.: ДМК Пресс, 2017. – 632 с.: ил. ISBN 978-5-97060-180-8
3. Уоллс К. Spring в действии. – М.: ДМК Пресс, 2013. – 752 с.: ил.ISBN 978-5-94074-568-6
4. Козмина Юлиана, Харроп Роб, Шефер Крис, Хо Кларенс Spring 5 для профессионалов.: Пер. с англ. - СПб.: ООО "Диалектика", 2019. - 1120 с.: ил. - Парал. тит. англ. ISBN 978-5-907114-07-4 (рус.)

1. Java Database Connector [↑](#footnote-ref-1)