Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский университет ИТМО»

Итерация №3

По дисциплине «Рефакторинг баз данных и приложений»

Группа: P3412

Выполнили: Балин А.А., Кобелев Р. П.

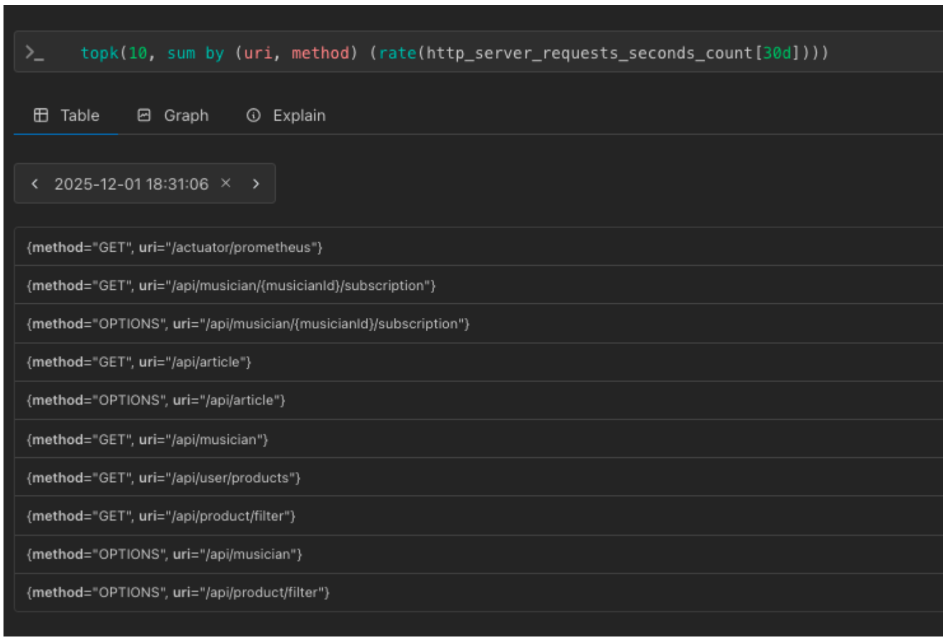
Проверил: Логинов И.П.

Санкт-Петербург
2025г.

Оглавление

Добавление Redis

Настроить Redis для кэширования часто запрашиваемых данных. Снизит нагрузку на базу данных и ускорит отклик сервиса. Чтобы определить, какие запросы будем кешировать, выполним запрос в Prometheus:



```
topk(10, sum by (uri, method) (rate(http_server_requests_seconds_count[30d])))
```

```
{method="GET", uri="/actuator/prometheus"}
{method="GET", uri="/api/musician/{musicianId}/subscription"}
{method="OPTIONS", uri="/api/musician/{musicianId}/subscription"}
{method="GET", uri="/api/article"}
{method="OPTIONS", uri="/api/article"}
{method="GET", uri="/api/musician"}
{method="GET", uri="/api/user/products"}
{method="GET", uri="/api/product/filter"}
{method="OPTIONS", uri="/api/musician"}
{method="OPTIONS", uri="/api/product/filter"}
```

Коммиты: 1

Оптимизация запросов к базе данных

Оптимизировать запросы к базе данных (индексы, оптимизация SQL-запросов). Улучшит производительность при большом количестве пользователей.

Анализ запросов представлен в отдельном файле.

Коммиты: 1, 2, 3, 4, 5, 6

DIFF

git log --oneline 'c6dd1f14b14c769cc14854ea4aa3112ffe74f9f1^..0c966b6731b9e78e44f04db3d9dbe342fc54c196'

```
0c966b6 (HEAD -> main, tag: v3.0.1, origin/main, origin/HEAD) Merge pull request #3 from Romariok/feat/redis
48b6155 (tag: v3.0.0) Merge branch 'main' into feat/redis
d0e813b lint
9d90f61 overall results in report
22cfe2f login optimization
0de8c0a article optimization
8c83af6 musician optimization
2da5120 fix for big test data insert
43286d1 (origin/feat/redis, feat/redis) added 3th iteration doc file
c6dd1f1 added redis to optimize requests
```

git diff 'ac73189220059f7880d62ab4f9cc091c635df818^..276c8eb735fc2193ad5a113f44ef00c19524a871' –stat

```
.docker/.gitignore                                 |   3 +-
 .docker/docker-compose.yaml                       |  17 +
 SQL/initScript.sql                                |   1 +
 SQL/insertTestData.sql                            |  23 +-
 backend/pom.xml                                   |   8 +
 .../cw/GuitarMatchIS/GuitarMatchISApplication.java |   2 +
 .../cw/GuitarMatchIS/config/RedisCacheConfig.java  |  46 ++
 .../itmo/is/cw/GuitarMatchIS/models/Article.java   |   3 +
 .../repository/ArticleRepository.java             |  11 +-
 .../repository/MusicianGenreRepository.java       |   2 +
 .../repository/MusicianProductRepository.java     |   6 +-
 .../MusicianTypeOfMusicianRepository.java         |   2 +
 .../cw/GuitarMatchIS/security/SecurityConfig.java  |  54 +-
 .../cw/GuitarMatchIS/service/ArticleService.java   |  11 +-
 .../is/cw/GuitarMatchIS/service/AuthService.java   |  15 +-
 .../cw/GuitarMatchIS/service/MusicianService.java  | 585 +++++++++++----------
 .../cw/GuitarMatchIS/service/ProductService.java   |  26 +
 .../is/cw/GuitarMatchIS/service/UserService.java   |  37 +-
 backend/src/main/resources/application.properties |  14 +
 docs/refactoring/3-it/1.png                       | Bin 0 -> 63438 bytes
 docs/refactoring/3-it/2.png                       | Bin 0 -> 132998 bytes
 ...5\321\200\320\260\321\206\320\270\321\217.docx" | Bin 0 -> 88788 bytes
 docs/refactoring/3-it/3.png                       | Bin 0 -> 128914 bytes
 docs/refactoring/3-it/4.png                       | Bin 0 -> 155934 bytes
 docs/refactoring/3-it/5.png                       | Bin 0 -> 164283 bytes
 docs/refactoring/3-it/6.png                       | Bin 0 -> 132549 bytes
 docs/refactoring/3-it/7.png                       | Bin 0 -> 385273 bytes
 docs/refactoring/3-it/optimizations.md            | 161 ++++++
 28 files changed, 692 insertions(+), 335 deletions(-)
```

git show ac73189

```
commit c6dd1f14b14c769cc14854ea4aa3112ffe74f9f1
Author: Romariok <ronya-ko@yandex.ru>
Date:   Mon Dec 1 21:11:47 2025 +0300


    added redis to optimize requests
```

```
diff --git a/.docker/.gitignore b/.docker/.gitignore
index 3fce277..0d2f82a 100644
--- a/.docker/.gitignore
+++ b/.docker/.gitignore
@@ -1,3 +1,4 @@
 /InitDatabase
 /pgadmin
-/pgdata
\ No newline at end of file
+/pgdata
+/redis
\ No newline at end of file
diff --git a/.docker/docker-compose.yaml b/.docker/docker-compose.yaml
index efad857..832790c 100644
--- a/.docker/docker-compose.yaml
+++ b/.docker/docker-compose.yaml
@@ -83,6 +83,18 @@ services:
     networks:
       - postgres

+  redis:
+    container_name: redis
+    image: redis:7-alpine
+    command: ["redis-server", "--appendonly", "yes"]
+    ports:
+      - "6379:6379"
+    volumes:
+      - ./redis/data:/data
+    restart: unless-stopped
+    networks:
+      - postgres
+
   cadvisor:
     container_name: cadvisor
     image: gcr.io/cadvisor/cadvisor:latest
@@ -114,6 +126,8 @@ services:
       JWT_SECRET: "dev-secret"
       # Явно указываем путь к собранному Linux-бинарнику парсера внутри контейнера
       APP_EXTERNAL_MARKDOWN_PARSER_PATH: "/opt/app/parser"
+      REDIS_HOST: "redis"
+      REDIS_PORT: "6379"
     volumes:
       - ../backend/logs:/opt/app/logs
     ports:
@@ -121,6 +135,8 @@ services:
     depends_on:
       postgres:
         condition: service_healthy
+      redis:
+        condition: service_started
     restart: unless-stopped
     networks:
       - postgres
diff --git a/backend/pom.xml b/backend/pom.xml
index 5a2c1a8..d720d05 100644
```

```
--- a/backend/pom.xml
+++ b/backend/pom.xml
@@ -23,6 +23,14 @@
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
+               <dependency>
+                       <groupId>org.springframework.boot</groupId>
+                       <artifactId>spring-boot-starter-cache</artifactId>
+               </dependency>
+               <dependency>
+                       <groupId>org.springframework.boot</groupId>
+                       <artifactId>spring-boot-starter-data-redis</artifactId>
+               </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-actuator</artifactId>
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/GuitarMatchISApplication.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/GuitarMatchISApplication.java
index 0aab626..220cccf 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/GuitarMatchISApplication.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/GuitarMatchISApplication.java
@@ -2,9 +2,11 @@ package itmo.is.cw.GuitarMatchIS;

 import org.springframework.boot.SpringApplication;
 import org.springframework.boot.autoconfigure.SpringBootApplication;
+import org.springframework.cache.annotation.EnableCaching;
 import org.springframework.scheduling.annotation.EnableAsync;

 @SpringBootApplication
+@EnableCaching
 @EnableAsync
 public class GuitarMatchISApplication {

diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/config/RedisCacheConfig.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/config/RedisCacheConfig.java
new file mode 100644
index 0000000..ca1ad1c
--- /dev/null
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/config/RedisCacheConfig.java
@@ -0,0 +1,46 @@
+package itmo.is.cw.GuitarMatchIS.config;
+
+import com.fasterxml.jackson.databind.ObjectMapper;
+import com.fasterxml.jackson.databind.SerializationFeature;
+import com.fasterxml.jackson.datatype.jsr310.JavaTimeModule;
+import org.springframework.boot.autoconfigure.cache.CacheProperties;
+import org.springframework.context.annotation.Bean;
+import org.springframework.context.annotation.Configuration;
+import org.springframework.data.redis.cache.RedisCacheConfiguration;
+import org.springframework.data.redis.serializer.GenericJackson2JsonRedisSerializer;
+import org.springframework.data.redis.serializer.RedisSerializationContext;
+import org.springframework.data.redis.serializer.StringRedisSerializer;
+
+@Configuration
+public class RedisCacheConfig {
```

```
+
+  @Bean
+  public RedisCacheConfiguration redisCacheConfiguration(ObjectMapper objectMapper,
+        CacheProperties cacheProperties) {
+     ObjectMapper mapper = objectMapper.copy();
+     mapper.registerModule(new JavaTimeModule());
+     mapper.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
+
+     RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig()
+           .serializeKeysWith(
+                 RedisSerializationContext.SerializationPair.fromSerializer(new StringRedisSerializer()))
+           .serializeValuesWith(RedisSerializationContext.SerializationPair
+                 .fromSerializer(new GenericJackson2JsonRedisSerializer(mapper)));
+
+     CacheProperties.Redis redisProps = cacheProperties.getRedis();
+     if (redisProps.getTimeToLive() != null) {
+        config = config.entryTtl(redisProps.getTimeToLive());
+     }
+     if (redisProps.getKeyPrefix() != null) {
+        config = config.prefixCacheNameWith(redisProps.getKeyPrefix());
+     }
+     if (!redisProps.isCacheNullValues()) {
+        config = config.disableCachingNullValues();
+     }
+     if (!redisProps.isUseKeyPrefix()) {
+        config = config.disableKeyPrefix();
+     }
+     return config;
+  }
+}
+
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/security/SecurityConfig.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/security/SecurityConfig.java
index 44eee9b..9840e7e 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/security/SecurityConfig.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/security/SecurityConfig.java
@@ -36,37 +36,37 @@ public class SecurityConfig {
    @Value("#{'${app.cors.allowed-origins:http://localhost:5173}'.split(',')}")
    private List<String> corsAllowedOrigins;


-
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      return http.csrf(AbstractHttpConfigurer::disable)
-     .cors(cors -> cors.configurationSource(request -> {
-        CorsConfiguration corsConfiguration = new CorsConfiguration();
-        corsConfiguration.setAllowedOrigins(corsAllowedOrigins);
-        corsConfiguration.setAllowedMethods(List.of("POST", "GET", "PUT", "PATCH","DELETE", "OPTIONS"));
-        corsConfiguration.setAllowedHeaders(List.of("Content-Type", "Authorization", "X-Requested-With", "from", "size"));
-        corsConfiguration.setAllowCredentials(true);
-        return corsConfiguration;
-     }))
-     .authorizeHttpRequests(
-           request -> request
-                 .requestMatchers("/index.html", "/favicon.ico", "/static/**").permitAll()
-                 .requestMatchers("/actuator/**").permitAll()
-                   .requestMatchers("api/auth/**", "api/user/role/**", "/ws", "/api/").permitAll()
-                   .requestMatchers(HttpMethod.GET, "api/admin/**").hasRole("ADMIN")
-                   .requestMatchers(HttpMethod.POST, "api/moderate/**").hasRole("ADMIN")
-                   .anyRequest().authenticated()
-     ).userDetailsService(userDetailsService)
-     .sessionManagement(session -> session
-             .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
-     .addFilterBefore(jwtAuthTokenFilter, UsernamePasswordAuthenticationFilter.class)
-     .exceptionHandling(exHandler -> exHandler.authenticationEntryPoint(new JwtAuthEntryPoint()))
-     .build();
+           .cors(cors -> cors.configurationSource(request -> {
+               CorsConfiguration corsConfiguration = new CorsConfiguration();
+               corsConfiguration.setAllowedOrigins(corsAllowedOrigins);
+               corsConfiguration.setAllowedMethods(List.of("POST", "GET", "PUT", "PATCH", "DELETE", "OPTIONS"));
+               corsConfiguration
+                   .setAllowedHeaders(List.of("Content-Type", "Authorization", "X-Requested-With", "from", "size"));
+               corsConfiguration.setAllowCredentials(true);
+               return corsConfiguration;
+           }))
+           .authorizeHttpRequests(
+               request -> request
+                   .requestMatchers("/index.html", "/favicon.ico", "/static/**").permitAll()
+                   .requestMatchers("/actuator/**").permitAll()
+                   .requestMatchers("/api/auth/**", "/api/user/role/**", "/ws", "/api/").permitAll()
+                   .requestMatchers(HttpMethod.GET, "/api/admin/**").hasRole("ADMIN")
+                   .requestMatchers(HttpMethod.POST, "/api/moderate/**").hasRole("ADMIN")
+                   .anyRequest().authenticated())
+           .userDetailsService(userDetailsService)
+           .authenticationProvider(daoAuthenticationProvider())
+           .sessionManagement(session -> session
+               .sessionCreationPolicy(SessionCreationPolicy.STATELESS))
+           .addFilterBefore(jwtAuthTokenFilter, UsernamePasswordAuthenticationFilter.class)
+           .exceptionHandling(exHandler -> exHandler.authenticationEntryPoint(new JwtAuthEntryPoint()))
+           .build();

   }

-  @Bean
-  public DaoAuthenticationProvider authenticationProvider() {
+  private DaoAuthenticationProvider daoAuthenticationProvider() {
     DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

     authProvider.setUserDetailsService(userDetailsService);
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ArticleService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ArticleService.java
index 2981035..bf0ee71 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ArticleService.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ArticleService.java
@@ -10,6 +10,8 @@ import java.util.List;
 import java.util.concurrent.CompletableFuture;
 import java.util.concurrent.ExecutionException;

+import org.springframework.cache.annotation.CacheEvict;
+import org.springframework.cache.annotation.Cacheable;
 import org.springframework.beans.factory.annotation.Value;
 import org.springframework.data.domain.PageRequest;
 import org.springframework.data.domain.Pageable;
@@ -63,6 +65,7 @@ public class ArticleService {
    @Value("${app.external.markdown-parser.path:${user.dir}/parser}")
    private String markdownParserPath;

+   @Cacheable(value = "articles", key = "'accepted:' + #from + ':' + #size + ':' + #sortBy + ':' + #ascending")
    public List<ArticleDTO> getAcceptedArticles(int from, int size, ArticleSort sortBy, boolean ascending) {
      log.info("Fetching accepted articles from: {}, size: {}, sortBy: {}, ascending: {}", from, size, sortBy,
           ascending);
@@ -112,6 +115,7 @@ public class ArticleService {
           .toList();
    }

+   @CacheEvict(value = "articles", allEntries = true)
    public boolean moderateArticle(ModerateArticleDTO moderateArticleDTO, HttpServletRequest request) {
      User moderator = findUserByRequest(request);
      log.info("Moderating article with id: {} by user: {}", moderateArticleDTO.getArticleId(),
@@ -140,6 +144,7 @@ public class ArticleService {
    }

    @Transactional
+   @CacheEvict(value = "articles", allEntries = true)
    public ArticleDTO createArticle(CreateArticleDTO createArticleDTO, HttpServletRequest request) {
      log.info("Creating article with header: {}", createArticleDTO.getHeader());
      if (!productRepository.existsByName(createArticleDTO.getProductName())) {
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
index 85fc21e..7c06da9 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
@@ -1,5 +1,7 @@
 package itmo.is.cw.GuitarMatchIS.service;

+import org.springframework.cache.annotation.CacheEvict;
+import org.springframework.cache.annotation.Cacheable;
 import org.springframework.messaging.simp.SimpMessagingTemplate;
 import org.springframework.security.core.userdetails.UsernameNotFoundException;
 import org.springframework.stereotype.Service;
@@ -55,297 +57,314 @@ import java.util.Comparator;
 @RequiredArgsConstructor
 @Slf4j
 public class MusicianService {
-   private final MusicianRepository musicianRepository;
-   private final MusicianGenreRepository musicianGenreRepository;
-   private final MusicianTypeOfMusicianRepository musicianTypeOfMusicianRepository;
-   private final JwtUtils jwtUtils;
-   private final UserMusicianRepository userMusicianRepository;
-   private final SimpMessagingTemplate simpMessagingTemplate;
-   private final UserRepository userRepository;
-   private final MusicianProductRepository musicianProductRepository;
-   private final ProductRepository productRepository;
-
-   public List<MusicianInfoDTO> getMusician(int from, int size, MusicianSort sortBy, boolean ascending) {
-     log.info("Fetching musicians from: {}, size: {}, sortBy: {}, ascending: {}", from, size, sortBy, ascending);
-     Sort sort = Sort.by(ascending ? Sort.Direction.ASC : Sort.Direction.DESC,
-           sortBy.getFieldName());
-     Pageable page = PageRequest.of(from / size, size, sort);
-
-     List<Musician> musicians = musicianRepository.findAll(page).getContent();
-     return musicians
-           .stream()
-           .map(musician1 -> convertToDTO(musician1,
-                 musicianGenreRepository.findByMusician(musician1),
-                 musicianTypeOfMusicianRepository.findByMusician(musician1),
-                 musicianProductRepository.findByMusician(musician1)))
-           .toList();
-   }
-
-   public Boolean isSubscribed(Long musicianId, HttpServletRequest request) {
-     User user = findUserByRequest(request);
-     log.info("Checking if user {} is subscribed to musician with id: {}", user.getUsername(), musicianId);
-     return userMusicianRepository.existsByUserAndMusician(user,
-           musicianRepository.findById(musicianId).orElseThrow(() -> {
-             log.warn("Musician with id {} not found while checking subscription", musicianId);
-             return new MusicianNotFoundException("Musician with id %s not found".formatted(musicianId));
-           }));
-   }
-
-   @Transactional
-   public MusicianInfoDTO createMusician(CreateMusicianDTO createMusicianDTO, HttpServletRequest request) {
-     User user = findUserByRequest(request);
-     log.info("User {} is creating musician with name: {}", user.getUsername(), createMusicianDTO.getName());
-     if (musicianRepository.existsByName(createMusicianDTO.getName())) {
-       log.warn("Musician with name {} already exists", createMusicianDTO.getName());
-       throw new MusicianAlreadyExistsException("Musician %s already exists".formatted(createMusicianDTO.getName()));
+     private final MusicianRepository musicianRepository;
+     private final MusicianGenreRepository musicianGenreRepository;
+     private final MusicianTypeOfMusicianRepository musicianTypeOfMusicianRepository;
+     private final JwtUtils jwtUtils;
+     private final UserMusicianRepository userMusicianRepository;
+     private final SimpMessagingTemplate simpMessagingTemplate;
+     private final UserRepository userRepository;
+     private final MusicianProductRepository musicianProductRepository;
+     private final ProductRepository productRepository;
+
+     @Cacheable(value = "musicians", key = "'list:' + #from + ':' + #size + ':' + #sortBy + ':' + #ascending")
+     public List<MusicianInfoDTO> getMusician(int from, int size, MusicianSort sortBy, boolean ascending) {
+         log.info("Fetching musicians from: {}, size: {}, sortBy: {}, ascending: {}", from, size, sortBy, ascending);
+         Sort sort = Sort.by(ascending ? Sort.Direction.ASC : Sort.Direction.DESC,
+                 sortBy.getFieldName());
+         Pageable page = PageRequest.of(from / size, size, sort);
+
+         List<Musician> musicians = musicianRepository.findAll(page).getContent();
+         return musicians
+                 .stream()
+                 .map(musician1 -> convertToDTO(musician1,
+                         musicianGenreRepository.findByMusician(musician1),
+                         musicianTypeOfMusicianRepository.findByMusician(musician1),
+                         musicianProductRepository.findByMusician(musician1)))
+                 .toList();
      }
-     Musician musician = Musician.builder()
-           .name(createMusicianDTO.getName())
-           .subscribers(0)
-           .build();

-       musician = musicianRepository.save(musician);
-       log.info("Musician with name {} successfully created with id {}", musician.getName(), musician.getId());
+       public Boolean isSubscribed(Long musicianId, HttpServletRequest request) {
+           User user = findUserByRequest(request);
+           log.info("Checking if user {} is subscribed to musician with id: {}", user.getUsername(), musicianId);
+           return userMusicianRepository.existsByUserAndMusician(user,
+                   musicianRepository.findById(musicianId).orElseThrow(() -> {
+                       log.warn("Musician with id {} not found while checking subscription", musicianId);
+                       return new MusicianNotFoundException(
+                               "Musician with id %s not found".formatted(musicianId));
+                   }));
+       }
+
+       @Transactional
+       @CacheEvict(value = "musicians", allEntries = true)
+       public MusicianInfoDTO createMusician(CreateMusicianDTO createMusicianDTO, HttpServletRequest request) {
+           User user = findUserByRequest(request);
+           log.info("User {} is creating musician with name: {}", user.getUsername(), createMusicianDTO.getName());
+           if (musicianRepository.existsByName(createMusicianDTO.getName())) {
+               log.warn("Musician with name {} already exists", createMusicianDTO.getName());
+               throw new MusicianAlreadyExistsException(
+                       "Musician %s already exists".formatted(createMusicianDTO.getName()));
+           }
+           Musician musician = Musician.builder()
+                   .name(createMusicianDTO.getName())
+                   .subscribers(0)
+                   .build();
+
+           musician = musicianRepository.save(musician);
+           log.info("Musician with name {} successfully created with id {}", musician.getName(), musician.getId());
+
+           for (Genre genre : createMusicianDTO.getGenres()) {
+               musicianGenreRepository.saveByMusicianIdAndGenre(musician.getId(), genre.toString());
+           }
+
+           for (TypeOfMusician typeOfMusician : createMusicianDTO.getTypesOfMusician()) {
+               musicianTypeOfMusicianRepository.saveByMusicianIdAndTypeOfMusician(musician.getId(),
+                       typeOfMusician.toString());
+           }
+
+           simpMessagingTemplate.convertAndSend("/musicians", "New musician added");
+           log.info("Musician creation message sent to websocket");
+
+           return convertToDTOLists(musician, createMusicianDTO.getGenres(), createMusicianDTO.getTypesOfMusician(),
+                   new ArrayList<>());
+       }
+
+       public Boolean subscribeToMusician(SubscribeDTO subscribeDTO, HttpServletRequest request) {
+           User user = findUserByRequest(request);
+           log.info("User {} is subscribing to musician with id: {}", user.getUsername(),
+                   subscribeDTO.getMusicianId());
+
+          Musician musician = musicianRepository.findById(subscribeDTO.getMusicianId())
+                  .orElseThrow(() -> {
+                      log.warn("Musician with id {} not found while subscribing", subscribeDTO.getMusicianId());
+                      return new MusicianNotFoundException(
+                              "Musician with id %s not found".formatted(subscribeDTO.getMusicianId()));
+                  });
+          if (userMusicianRepository.existsByUserAndMusician(user, musician)) {
+              log.warn("User {} is already subscribed to musician {}", user.getUsername(), musician.getName());
+              throw new SubscriptionAlreadyExistsException("You are already subscribed to this musician");
+          }
+          simpMessagingTemplate.convertAndSend("/musicians", "New subscriber to musician");
+          userMusicianRepository.subscribeToMusician(user.getId(), subscribeDTO.getMusicianId());
+          log.info("User {} successfully subscribed to musician {}", user.getUsername(), musician.getName());
+          return true;
+      }
+
+      @Transactional
+      public Boolean unsubscribeFromMusician(SubscribeDTO subscribeDTO, HttpServletRequest request) {
+          User user = findUserByRequest(request);
+          log.info("User {} is unsubscribing from musician with id: {}", user.getUsername(),
+                  subscribeDTO.getMusicianId());
+
+          Musician musician = musicianRepository.findById(subscribeDTO.getMusicianId())
+                  .orElseThrow(() -> {
+                      log.warn("Musician with id {} not found while unsubscribing",
+                              subscribeDTO.getMusicianId());
+                      return new MusicianNotFoundException(
+                              "Musician with id %s not found".formatted(subscribeDTO.getMusicianId()));
+                  });
+          if (!userMusicianRepository.existsByUserAndMusician(user, musician)) {
+              log.warn("User {} is not subscribed to musician {}", user.getUsername(), musician.getName());
+              throw new SubscriptionNotFoundException("You have no subscription to this musician");
+          }
+          simpMessagingTemplate.convertAndSend("/musicians", "Deleted subscription to musician");
+          userMusicianRepository.deleteByUserAndMusician(user, musician);
+          log.info("User {} successfully unsubscribed from musician {}", user.getUsername(), musician.getName());
+          return true;
+      }
+
+      public List<MusicianInfoDTO> searchMusicians(String name, int from, int size) {
+          log.info("Searching for musicians with name containing: {}", name);
+          Pageable page = Pagification.createPageTemplate(from, size);
+          List<Musician> musicians = musicianRepository.findAllByNameContains(name, page).getContent();
+
+          return musicians
+                  .stream()
+                  .map(musician1 -> convertToDTO(musician1,
+                          musicianGenreRepository.findByMusician(musician1),
+                          musicianTypeOfMusicianRepository.findByMusician(musician1),
+                          musicianProductRepository.findByMusician(musician1)))
+                  .sorted(Comparator.comparing(MusicianInfoDTO::getId))
+                  .toList();
+      }
+
+      public MusicianGenreDTO getMusiciansByGenre(Long musicianId) {
+          log.info("Fetching genres for musician with id: {}", musicianId);
+        Musician musician = musicianRepository.findById(musicianId)
+                .orElseThrow(() -> new MusicianNotFoundException(
+                        "Musician with id %s not found".formatted(musicianId)));
+
+        List<MusicianGenre> musicianGenres = musicianGenreRepository.findByMusician(musician);
+
+        return MusicianGenreDTO.builder()
+                .musician(musician)
+                .genres(musicianGenres.stream().map(MusicianGenre::getGenre).toList())
+                .build();
+    }
+
+    public MusicianTypeOfMusicianDTO getMusiciansByTypeOfMusician(Long musicianId) {
+        log.info("Fetching types for musician with id: {}", musicianId);
+        Musician musician = musicianRepository.findById(musicianId)
+                .orElseThrow(() -> new MusicianNotFoundException(
+                        "Musician with id %s not found".formatted(musicianId)));

-    for (Genre genre : createMusicianDTO.getGenres()) {
-      musicianGenreRepository.saveByMusicianIdAndGenre(musician.getId(), genre.toString());
+        List<MusicianTypeOfMusician> musicianTypeOfMusicians = musicianTypeOfMusicianRepository
+                .findByMusician(musician);
+
+        return MusicianTypeOfMusicianDTO.builder()
+                .musician(musician)
+                .typeOfMusicians(musicianTypeOfMusicians.stream().map(MusicianTypeOfMusician::getTypeOfMusician)
+                        .toList())
+                .build();
+    }
+
+    public MusicianProductDTO getMusicianProducts(String name) {
+        log.info("Fetching products for musician with name: {}", name);
+        if (!musicianRepository.existsByName(name))
+            throw new MusicianNotFoundException("Musician with name %s not found".formatted(name));
+
+        Musician musician = musicianRepository.findByName(name);
+
+        List<MusicianProduct> musicianProducts = musicianProductRepository.findByMusician(musician);
+
+        return MusicianProductDTO.builder()
+                .musician(MusicianDTO.builder()
+                        .id(musician.getId())
+                        .name(musician.getName())
+                        .subscribers(musician.getSubscribers())
+                        .build())
+                .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO)
+                        .toList())
+                .build();
+    }
+
+    public MusicianInfoDTO getMusicianInfo(Long musicianId) {
+        log.info("Fetching info for musician with id: {}", musicianId);
+        Musician musician = musicianRepository.findById(musicianId)
+                .orElseThrow(() -> new MusicianNotFoundException(
+                        "Musician with id %s not found".formatted(musicianId)));
+
+           return convertToDTO(musician,
+                   musicianGenreRepository.findByMusician(musician),
+                   musicianTypeOfMusicianRepository.findByMusician(musician),
+                   musicianProductRepository.findByMusician(musician));
        }

-       for (TypeOfMusician typeOfMusician : createMusicianDTO.getTypesOfMusician()) {
-         musicianTypeOfMusicianRepository.saveByMusicianIdAndTypeOfMusician(musician.getId(),
-             typeOfMusician.toString());
+       @Transactional
+       public Boolean addProductToMusician(AddProductMusicianDTO addProductMusicianDTO, HttpServletRequest request) {
+           log.info("Adding product with id {} to musician with name {}", addProductMusicianDTO.getProductId(),
+                   addProductMusicianDTO.getMusicianName());
+           if (!musicianRepository.existsByName(addProductMusicianDTO.getMusicianName()))
+               throw new MusicianNotFoundException(
+                       "Musician with name %s not found".formatted(addProductMusicianDTO.getMusicianName()));
+
+           if (!productRepository.existsById(addProductMusicianDTO.getProductId()))
+               throw new ProductNotFoundException(
+                       "Product with id %s not found".formatted(addProductMusicianDTO.getProductId()));
+
+           User user = findUserByRequest(request);
+           log.info("User {} is performing this action", user.getUsername());
+           Musician musician = musicianRepository.findByName(addProductMusicianDTO.getMusicianName());
+           Product product = productRepository.findById(addProductMusicianDTO.getProductId()).get();
+
+           if (musicianProductRepository.existsByMusicianAndProduct(musician, product)) {
+               log.warn("Product {} already exists in musician {}", product.getName(), musician.getName());
+               throw new ProductMusicianAlreadyExists(
+                       "Product %s already exists in musician %s".formatted(product.getName(),
+                               musician.getName()));
+           }
+           musicianProductRepository
+                   .save(MusicianProduct.builder().musicianId(musician.getId()).productId(product.getId())
+                           .build());
+           log.info("Product {} successfully added to musician {}", product.getName(), musician.getName());
+           return true;
        }

-       simpMessagingTemplate.convertAndSend("/musicians", "New musician added");
-       log.info("Musician creation message sent to websocket");
-
-       return convertToDTOLists(musician, createMusicianDTO.getGenres(), createMusicianDTO.getTypesOfMusician(),
-           new ArrayList<>());
-   }
-
-   public Boolean subscribeToMusician(SubscribeDTO subscribeDTO, HttpServletRequest request) {
-       User user = findUserByRequest(request);
-       log.info("User {} is subscribing to musician with id: {}", user.getUsername(), subscribeDTO.getMusicianId());
-
-       Musician musician = musicianRepository.findById(subscribeDTO.getMusicianId())
-           .orElseThrow(() -> {
-             log.warn("Musician with id {} not found while subscribing", subscribeDTO.getMusicianId());
-             return new MusicianNotFoundException(
-                 "Musician with id %s not found".formatted(subscribeDTO.getMusicianId()));
-           });
-     if (userMusicianRepository.existsByUserAndMusician(user, musician)) {
-         log.warn("User {} is already subscribed to musician {}", user.getUsername(), musician.getName());
-         throw new SubscriptionAlreadyExistsException("You are already subscribed to this musician");
+     @Transactional
+     public Boolean deleteProductFromMusician(AddProductMusicianDTO addProductMusicianDTO,
+                 HttpServletRequest request) {
+         log.info("Deleting product with id {} from musician with name {}", addProductMusicianDTO.getProductId(),
+                     addProductMusicianDTO.getMusicianName());
+         if (!musicianRepository.existsByName(addProductMusicianDTO.getMusicianName()))
+             throw new MusicianNotFoundException(
+                     "Musician with name %s not found".formatted(addProductMusicianDTO.getMusicianName()));
+
+         if (!productRepository.existsById(addProductMusicianDTO.getProductId()))
+             throw new ProductNotFoundException(
+                     "Product with id %s not found".formatted(addProductMusicianDTO.getProductId()));
+         User user = findUserByRequest(request);
+         log.info("User {} is performing this action", user.getUsername());
+         Musician musician = musicianRepository.findByName(addProductMusicianDTO.getMusicianName());
+         Product product = productRepository.findById(addProductMusicianDTO.getProductId()).get();
+
+         if (!musicianProductRepository.existsByMusicianAndProduct(musician, product)) {
+             log.warn("Product {} not found in musician {}", product.getName(), musician.getName());
+             throw new ProductMusicianNotFoundException(
+                     "Product %s not found in musician %s".formatted(product.getName(),
+                             musician.getName()));
+         }
+         musicianProductRepository.deleteByMusicianAndProduct(musician, product);
+         log.info("Product {} successfully deleted from musician {}", product.getName(), musician.getName());
+         return true;
      }
-     simpMessagingTemplate.convertAndSend("/musicians", "New subscriber to musician");
-     userMusicianRepository.subscribeToMusician(user.getId(), subscribeDTO.getMusicianId());
-     log.info("User {} successfully subscribed to musician {}", user.getUsername(), musician.getName());
-     return true;
-   }
-
-   @Transactional
-   public Boolean unsubscribeFromMusician(SubscribeDTO subscribeDTO, HttpServletRequest request) {
-     User user = findUserByRequest(request);
-     log.info("User {} is unsubscribing from musician with id: {}", user.getUsername(), subscribeDTO.getMusicianId());
-
-     Musician musician = musicianRepository.findById(subscribeDTO.getMusicianId())
-         .orElseThrow(() -> {
-             log.warn("Musician with id {} not found while unsubscribing", subscribeDTO.getMusicianId());
-             return new MusicianNotFoundException(
-                 "Musician with id %s not found".formatted(subscribeDTO.getMusicianId()));
-         });
-     if (!userMusicianRepository.existsByUserAndMusician(user, musician)) {
-         log.warn("User {} is not subscribed to musician {}", user.getUsername(), musician.getName());
-         throw new SubscriptionNotFoundException("You have no subscription to this musician");
+
+     private User findUserByRequest(HttpServletRequest request) {
+         String username = jwtUtils.getUserNameFromJwtToken(jwtUtils.parseJwt(request));
+         return userRepository.findByUsername(username)
+                     .orElseThrow(() -> new UsernameNotFoundException(
+                             String.format("Username %s not found", username)));
```

```
        }
-       simpMessagingTemplate.convertAndSend("/musicians", "Deleted subscription to musician");
-       userMusicianRepository.deleteByUserAndMusician(user, musician);
-       log.info("User {} successfully unsubscribed from musician {}", user.getUsername(), musician.getName());
-       return true;
-   }
-
-   public List<MusicianInfoDTO> searchMusicians(String name, int from, int size) {
-       log.info("Searching for musicians with name containing: {}", name);
-       Pageable page = Pagification.createPageTemplate(from, size);
-       List<Musician> musicians = musicianRepository.findAllByNameContains(name, page).getContent();
-
-       return musicians
-               .stream()
-               .map(musician1 -> convertToDTO(musician1,
-                       musicianGenreRepository.findByMusician(musician1),
-                       musicianTypeOfMusicianRepository.findByMusician(musician1),
-                       musicianProductRepository.findByMusician(musician1)))
-               .sorted(Comparator.comparing(MusicianInfoDTO::getId))
-               .toList();
-   }
-
-   public MusicianGenreDTO getMusiciansByGenre(Long musicianId) {
-       log.info("Fetching genres for musician with id: {}", musicianId);
-       Musician musician = musicianRepository.findById(musicianId)
-               .orElseThrow(() -> new MusicianNotFoundException(
-                       "Musician with id %s not found".formatted(musicianId)));
-
-       List<MusicianGenre> musicianGenres = musicianGenreRepository.findByMusician(musician);
-
-       return MusicianGenreDTO.builder()
-               .musician(musician)
-               .genres(musicianGenres.stream().map(MusicianGenre::getGenre).toList())
-               .build();
-   }
-
-   public MusicianTypeOfMusicianDTO getMusiciansByTypeOfMusician(Long musicianId) {
-       log.info("Fetching types for musician with id: {}", musicianId);
-       Musician musician = musicianRepository.findById(musicianId)
-               .orElseThrow(() -> new MusicianNotFoundException(
-                       "Musician with id %s not found".formatted(musicianId)));
-
-       List<MusicianTypeOfMusician> musicianTypeOfMusicians = musicianTypeOfMusicianRepository.findByMusician(musician);
-
-       return MusicianTypeOfMusicianDTO.builder()
-               .musician(musician)
-               .typeOfMusicians(musicianTypeOfMusicians.stream().map(MusicianTypeOfMusician::getTypeOfMusician).toList())
-               .build();
-   }
-
-   public MusicianProductDTO getMusicianProducts(String name) {
-       log.info("Fetching products for musician with name: {}", name);
-       if (!musicianRepository.existsByName(name))
-           throw new MusicianNotFoundException("Musician with name %s not found".formatted(name));
-
-       Musician musician = musicianRepository.findByName(name);
```

```
-
-    List<MusicianProduct> musicianProducts = musicianProductRepository.findByMusician(musician);
-
-    return MusicianProductDTO.builder()
-        .musician(MusicianDTO.builder()
-            .id(musician.getId())
-            .name(musician.getName())
-            .subscribers(musician.getSubscribers())
-            .build())
-        .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO).toList())
-        .build();
-  }
-
-  public MusicianInfoDTO getMusicianInfo(Long musicianId) {
-    log.info("Fetching info for musician with id: {}", musicianId);
-    Musician musician = musicianRepository.findById(musicianId)
-        .orElseThrow(() -> new MusicianNotFoundException(
-            "Musician with id %s not found".formatted(musicianId)));
-
-    return convertToDTO(musician,
-        musicianGenreRepository.findByMusician(musician),
-        musicianTypeOfMusicianRepository.findByMusician(musician),
-        musicianProductRepository.findByMusician(musician));
-  }
-
-  @Transactional
-  public Boolean addProductToMusician(AddProductMusicianDTO addProductMusicianDTO, HttpServletRequest request) {
-    log.info("Adding product with id {} to musician with name {}", addProductMusicianDTO.getProductId(),
-        addProductMusicianDTO.getMusicianName());
-    if (!musicianRepository.existsByName(addProductMusicianDTO.getMusicianName()))
-      throw new MusicianNotFoundException(
-          "Musician with name %s not found".formatted(addProductMusicianDTO.getMusicianName()));
-
-    if (!productRepository.existsById(addProductMusicianDTO.getProductId()))
-      throw new ProductNotFoundException(
-          "Product with id %s not found".formatted(addProductMusicianDTO.getProductId()));
-
-    User user = findUserByRequest(request);
-    log.info("User {} is performing this action", user.getUsername());
-    Musician musician = musicianRepository.findByName(addProductMusicianDTO.getMusicianName());
-    Product product = productRepository.findById(addProductMusicianDTO.getProductId()).get();
-
-    if (musicianProductRepository.existsByMusicianAndProduct(musician, product)) {
-      log.warn("Product {} already exists in musician {}", product.getName(), musician.getName());
-      throw new ProductMusicianAlreadyExists("Product %s already exists in musician %s".formatted(product.getName(),
-          musician.getName()));
+
+     private ProductDTO convertToDTO(Product product) {
+         return ProductDTO.builder()
+                 .id(product.getId())
+                 .name(product.getName())
+                 .description(product.getDescription())
+                 .rate(product.getRate())
+                 .brand(BrandDTO.builder()
+                         .id(product.getBrand().getId())
+                         .name(product.getBrand().getName())
+                               .country(product.getBrand().getCountry())
+                               .website(product.getBrand().getWebsite())
+                               .email(product.getBrand().getEmail())
+                               .build())
+                       .guitarForm(product.getGuitarForm())
+                       .typeOfProduct(product.getTypeOfProduct())
+                       .lads(product.getLads())
+                       .avgPrice(product.getAvgPrice())
+                       .color(product.getColor())
+                       .strings(product.getStrings())
+                       .tipMaterial(product.getTipMaterial())
+                       .bodyMaterial(product.getBodyMaterial())
+                       .pickupConfiguration(product.getPickupConfiguration())
+                       .typeComboAmplifier(product.getTypeComboAmplifier())
+                       .build();
     }
-    musicianProductRepository
-        .save(MusicianProduct.builder().musicianId(musician.getId()).productId(product.getId()).build());
-    log.info("Product {} successfully added to musician {}", product.getName(), musician.getName());
-    return true;
-  }
-
-  @Transactional
-  public Boolean deleteProductFromMusician(AddProductMusicianDTO addProductMusicianDTO, HttpServletRequest request) {
-    log.info("Deleting product with id {} from musician with name {}", addProductMusicianDTO.getProductId(),
-        addProductMusicianDTO.getMusicianName());
-    if (!musicianRepository.existsByName(addProductMusicianDTO.getMusicianName()))
-      throw new MusicianNotFoundException(
-          "Musician with name %s not found".formatted(addProductMusicianDTO.getMusicianName()));
-
-    if (!productRepository.existsById(addProductMusicianDTO.getProductId()))
-      throw new ProductNotFoundException(
-          "Product with id %s not found".formatted(addProductMusicianDTO.getProductId()));
-    User user = findUserByRequest(request);
-    log.info("User {} is performing this action", user.getUsername());
-    Musician musician = musicianRepository.findByName(addProductMusicianDTO.getMusicianName());
-    Product product = productRepository.findById(addProductMusicianDTO.getProductId()).get();
-
-    if (!musicianProductRepository.existsByMusicianAndProduct(musician, product)) {
-      log.warn("Product {} not found in musician {}", product.getName(), musician.getName());
-      throw new ProductMusicianNotFoundException("Product %s not found in musician %s".formatted(product.getName(),
-          musician.getName()));
+
+    private MusicianInfoDTO convertToDTO(Musician musician, List<MusicianGenre> musicianGenres,
+               List<MusicianTypeOfMusician> musicianTypes, List<MusicianProduct> musicianProducts) {
+        return MusicianInfoDTO.builder()
+                       .id(musician.getId())
+                       .name(musician.getName())
+                       .subscribers(musician.getSubscribers())
+                       .genres(musicianGenres.stream().map(MusicianGenre::getGenre).toList())
+                       .typesOfMusicians(
+                               musicianTypes.stream().map(MusicianTypeOfMusician::getTypeOfMusician).toList())
+                       .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO)
+                               .toList())
+                       .build();
+    }
```

```
+
+     private MusicianInfoDTO convertToDTOLists(Musician musician, List<Genre> genres,
+                 List<TypeOfMusician> typesOfMusician, List<MusicianProduct> musicianProducts) {
+         return MusicianInfoDTO.builder()
+                     .id(musician.getId())
+                     .name(musician.getName())
+                     .subscribers(musician.getSubscribers())
+                     .genres(genres)
+                     .typesOfMusicians(typesOfMusician)
+                     .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO)
+                             .toList())
+                     .build();
     }
-    musicianProductRepository.deleteByMusicianAndProduct(musician, product);
-    log.info("Product {} successfully deleted from musician {}", product.getName(), musician.getName());
-    return true;
-  }
-
-  private User findUserByRequest(HttpServletRequest request) {
-    String username = jwtUtils.getUserNameFromJwtToken(jwtUtils.parseJwt(request));
-    return userRepository.findByUsername(username)
-         .orElseThrow(() -> new UsernameNotFoundException(
-              String.format("Username %s not found", username)));
-  }
-
-  private ProductDTO convertToDTO(Product product) {
-    return ProductDTO.builder()
-         .id(product.getId())
-         .name(product.getName())
-         .description(product.getDescription())
-         .rate(product.getRate())
-         .brand(BrandDTO.builder()
-              .id(product.getBrand().getId())
-              .name(product.getBrand().getName())
-              .country(product.getBrand().getCountry())
-              .website(product.getBrand().getWebsite())
-              .email(product.getBrand().getEmail())
-              .build())
-         .guitarForm(product.getGuitarForm())
-         .typeOfProduct(product.getTypeOfProduct())
-         .lads(product.getLads())
-         .avgPrice(product.getAvgPrice())
-         .color(product.getColor())
-         .strings(product.getStrings())
-         .tipMaterial(product.getTipMaterial())
-         .bodyMaterial(product.getBodyMaterial())
-         .pickupConfiguration(product.getPickupConfiguration())
-         .typeComboAmplifier(product.getTypeComboAmplifier())
-         .build();
-  }
-
-  private MusicianInfoDTO convertToDTO(Musician musician, List<MusicianGenre> musicianGenres,
-       List<MusicianTypeOfMusician> musicianTypes, List<MusicianProduct> musicianProducts) {
-    return MusicianInfoDTO.builder()
-         .id(musician.getId())
-         .name(musician.getName())
-           .subscribers(musician.getSubscribers())
-           .genres(musicianGenres.stream().map(MusicianGenre::getGenre).toList())
-           .typesOfMusicians(musicianTypes.stream().map(MusicianTypeOfMusician::getTypeOfMusician).toList())
-           .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO).toList())
-           .build();
-   }
-
-   private MusicianInfoDTO convertToDTOLists(Musician musician, List<Genre> genres,
-           List<TypeOfMusician> typesOfMusician, List<MusicianProduct> musicianProducts) {
-       return MusicianInfoDTO.builder()
-           .id(musician.getId())
-           .name(musician.getName())
-           .subscribers(musician.getSubscribers())
-           .genres(genres)
-           .typesOfMusicians(typesOfMusician)
-           .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO).toList())
-           .build();
-   }
 }
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ProductService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ProductService.java
index 8730525..d97c096 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ProductService.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ProductService.java
@@ -1,5 +1,6 @@
 package itmo.is.cw.GuitarMatchIS.service;

+import org.springframework.cache.annotation.Cacheable;
 import org.springframework.stereotype.Service;
 import org.springframework.data.domain.PageRequest;
 import org.springframework.data.domain.Pageable;
@@ -204,6 +205,31 @@ public class ProductService {
                 boolean ascending,
                 int from, int size) {

+           return getProductsByFilterCached(name, minRate, maxRate, brandId, guitarForm, typeOfProduct, lads, minPrice,
+                   maxPrice, color, strings, tipMaterial, bodyMaterial, pickupConfiguration, typeComboAmplifier,
+                   sortBy, ascending, from, size);
+   }
+
+   @Cacheable(value = "products_filter", key = "'filter:' + #name + ':' + #minRate + ':' + #maxRate + ':' + #brandId + ':' + #guitarForm + ':' + #typeOfProduct + ':' + #lads + ':' + #minPrice + ':' + #maxPrice + ':' + #color + ':' + #strings + ':' + #tipMaterial + ':' + #bodyMaterial + ':' + #pickupConfiguration + ':' + #typeComboAmplifier + ':' + #sortBy + ':' + #ascending + ':' + #from + ':' + #size")
+   public List<ProductDTO> getProductsByFilterCached(String name,
+           Float minRate,
+           Float maxRate,
+           Long brandId,
+           GuitarForm guitarForm,
+           TypeOfProduct typeOfProduct,
+           Integer lads,
+           Double minPrice,
+           Double maxPrice,
+           Color color,
+           Integer strings,
+           TipMaterial tipMaterial,
+           BodyMaterial bodyMaterial,
+                PickupConfiguration pickupConfiguration,
+                TypeComboAmplifier typeComboAmplifier,
+                ProductSort sortBy,
+                boolean ascending,
+                int from, int size) {
+
         log.info("Fetching products by filter");
         Specification<Product> specification = Specification.where(ProductSpecification.hasBrand(brandId))
                 .and(ProductSpecification.hasName(name))
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/UserService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/UserService.java
index 10106d8..c40a241 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/UserService.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/UserService.java
@@ -154,7 +154,8 @@ public class UserService {
     public Boolean deleteProductFromUser(AddUserProductDTO addUserProductDTO, HttpServletRequest request) {
         log.info("Deleting product with id {} from user", addUserProductDTO.getProductId());
         if (!productRepository.existsById(addUserProductDTO.getProductId())) {
-            log.warn("Product with id {} not found while deleting from user", addUserProductDTO.getProductId());
+            log.warn("Product with id {} not found while deleting from user",
+                    addUserProductDTO.getProductId());
             throw new ProductNotFoundException(
                     "Product with id %s not found".formatted(addUserProductDTO.getProductId()));
         }
diff --git a/backend/src/main/resources/application.properties b/backend/src/main/resources/application.properties
index a2922a5..2bc68b4 100644
--- a/backend/src/main/resources/application.properties
+++ b/backend/src/main/resources/application.properties
@@ -38,3 +38,15 @@ management.endpoint.health.show-details=always
 management.prometheus.metrics.export.enabled=true
 management.metrics.distribution.percentiles-histogram.http.server.requests=true
 management.metrics.tags.application=${spring.application.name}
+
+# Cache / Redis
+spring.cache.type=redis
+spring.data.redis.host=${REDIS_HOST:localhost}
+spring.data.redis.port=${REDIS_PORT:6379}
+spring.data.redis.password=${REDIS_PASSWORD:}
+spring.cache.redis.time-to-live=60s
+spring.cache.redis.use-key-prefix=true
+spring.cache.redis.key-prefix=GMatch::
+spring.cache.redis.cache-null-values=false
+spring.data.redis.repositories.enabled=false
+spring.jpa.open-in-view=false
```

git show 43286d1

```
commit 43286d1a739798df80c59a048eaca86e3416a905 (origin/feat/redis, feat/redis)
Author: Romariok <ronya-ko@yandex.ru>
Date:   Mon Dec 1 21:44:37 2025 +0300

    added 3th iteration doc file

diff --git "a/docs/refactoring/3-it/3 \320\270\321\202\320\265\321\200\320\260\321\206\320\270\321\217.docx" "b/docs/refactoring/3-it/3 \320\270\321\202\320\265\321\200\320\260\321\206\320\270\321\217.docx"
new file mode 100644
```

```
index 0000000..e29f238
Binary files /dev/null and "b/docs/refactoring/3-it/3 \320\270\321\202\320\265\321\200\320\260\321\206\320\270\321\217.docx" differ
```

git show 2da5120

```
commit 2da5120dc00af39f27c1805746948ec240aa93b6
Author: ta4ilka69 <artem_balin_2004@mail.ru>
Date:   Mon Dec 1 23:09:32 2025 +0300

    fix for big test data insert

diff --git a/SQL/insertTestData.sql b/SQL/insertTestData.sql
index 8c5aa99..87180b0 100644
--- a/SQL/insertTestData.sql
+++ b/SQL/insertTestData.sql
@@ -53,13 +53,21 @@ SELECT
 FROM generate_series(1, 1000);

 -- Добавление отзывов
+-- Ограничение chk_feedback_target требует, чтобы был либо product_id, либо article_id, но не оба сразу.
+-- Чётные отзывы делаем по продуктам, нечётные — по статьям.
 INSERT INTO feedback (author_id, product_id, article_id, text, stars, created_at)
 SELECT
-    floor(random() * 13000 + 1)::int,
-    floor(random() * 130000 + 1)::int,
-    floor(random() * 1000 + 1)::int,
+    floor(random() * 13000 + 1)::int AS author_id, -- 1..13000, все существующие пользователи
+    CASE
+        WHEN generate_series % 2 = 0 THEN floor(random() * 130000 + 1)::int  -- только для чётных: ссылка на product
+        ELSE NULL
+    END AS product_id,
+    CASE
+        WHEN generate_series % 2 = 1 THEN floor(random() * 1000 + 1)::int     -- только для нечётных: ссылка на article
+        ELSE NULL
+    END AS article_id,
     'Feedback text ' || generate_series,
-    floor(random() * 5 + 1)::int,
+    floor(random() * 6)::int, -- 0..5, укладываемся в CHECK(stars between 0 and 5)
     current_date - (random() * 365)::int
 FROM generate_series(1, 10000);
```

git show 8c83af6

```
commit 8c83af65cf2b176c3b7eefef5d5cb77ab6879796
Author: ta4ilka69 <artem_balin_2004@mail.ru>
Date:   Mon Dec 1 23:33:13 2025 +0300

    musician optimization

diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianGenreRepository.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianGenreRepository.java
index e54f0b6..13d3a66 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianGenreRepository.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianGenreRepository.java
```

```
@@ -15,6 +15,8 @@ import org.springframework.data.repository.query.Param;
 public interface MusicianGenreRepository extends JpaRepository<MusicianGenre, MusicianGenreId> {
     List<MusicianGenre> findByMusician(Musician musician);

+    List<MusicianGenre> findByMusicianIdIn(List<Long> musicianIds);
+
     @Transactional
     @Modifying
     @Query(value = "INSERT INTO musician_genre (musician_id, genre) VALUES (:musicianId, CAST(:genre AS genre_enum))", nativeQuery = true)
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianProductRepository.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianProductRepository.java
index c91fca9..bf1e00e 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianProductRepository.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianProductRepository.java
@@ -2,18 +2,20 @@ package itmo.is.cw.GuitarMatchIS.repository;

 import java.util.List;

+import org.springframework.data.domain.Page;
+import org.springframework.data.domain.Pageable;
 import org.springframework.data.jpa.repository.JpaRepository;

 import itmo.is.cw.GuitarMatchIS.models.Musician;
 import itmo.is.cw.GuitarMatchIS.models.MusicianProduct;
 import itmo.is.cw.GuitarMatchIS.models.Product;
 import itmo.is.cw.GuitarMatchIS.models.keys.MusicianProductId;
-import org.springframework.data.domain.Page;
-import org.springframework.data.domain.Pageable;

 public interface MusicianProductRepository extends JpaRepository<MusicianProduct, MusicianProductId> {
     List<MusicianProduct> findByMusician(Musician musician);

+    List<MusicianProduct> findByMusicianIdIn(List<Long> musicianIds);
+
     Page<MusicianProduct> findByProduct(Product product, Pageable pageable);

     boolean existsByMusicianAndProduct(Musician musician, Product product);
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianTypeOfMusicianRepository.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianTypeOfMusicianRepository.java
index f4729e1..a14aa29 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianTypeOfMusicianRepository.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/MusicianTypeOfMusicianRepository.java
@@ -15,6 +15,8 @@ import jakarta.transaction.Transactional;
 public interface MusicianTypeOfMusicianRepository extends JpaRepository<MusicianTypeOfMusician, MusicianTypeOfMusicianId> {
     List<MusicianTypeOfMusician> findByMusician(Musician musician);

+    List<MusicianTypeOfMusician> findByMusicianIdIn(List<Long> musicianIds);
+
     @Transactional
     @Modifying
     @Query(value = "INSERT INTO type_of_musician_musician (musician_id, type_of_musician) VALUES (:musicianId, CAST(:typeOfMusician AS type_of_musician_enum))", nativeQuery = true)
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
index 85fc21e..a7ad82f 100644
```

```
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
@@ -50,6 +50,8 @@ import lombok.extern.slf4j.Slf4j;
 import java.util.List;
 import java.util.ArrayList;
 import java.util.Comparator;
+import java.util.Map;
+import java.util.stream.Collectors;

 @Service
 @RequiredArgsConstructor
@@ -70,15 +72,9 @@ public class MusicianService {
     Sort sort = Sort.by(ascending ? Sort.Direction.ASC : Sort.Direction.DESC,
         sortBy.getFieldName());
     Pageable page = PageRequest.of(from / size, size, sort);
-
     List<Musician> musicians = musicianRepository.findAll(page).getContent();
-    return musicians
-        .stream()
-        .map(musician1 -> convertToDTO(musician1,
-            musicianGenreRepository.findByMusician(musician1),
-            musicianTypeOfMusicianRepository.findByMusician(musician1),
-            musicianProductRepository.findByMusician(musician1)))
-        .toList();
+
+    return buildMusicianInfoDTOs(musicians);
   }

   public Boolean isSubscribed(Long musicianId, HttpServletRequest request) {
@@ -169,16 +165,42 @@ public class MusicianService {
     Pageable page = Pagification.createPageTemplate(from, size);
     List<Musician> musicians = musicianRepository.findAllByNameContains(name, page).getContent();

-    return musicians
+    return buildMusicianInfoDTOs(musicians)
         .stream()
-        .map(musician1 -> convertToDTO(musician1,
-            musicianGenreRepository.findByMusician(musician1),
-            musicianTypeOfMusicianRepository.findByMusician(musician1),
-            musicianProductRepository.findByMusician(musician1)))
         .sorted(Comparator.comparing(MusicianInfoDTO::getId))
         .toList();
   }

+  private List<MusicianInfoDTO> buildMusicianInfoDTOs(List<Musician> musicians) {
+    if (musicians.isEmpty()) {
+      return List.of();
+    }
+
+    List<Long> musicianIds = musicians.stream()
+        .map(Musician::getId)
+        .toList();
+
+    List<MusicianGenre> musicianGenres = musicianGenreRepository.findByMusicianIdIn(musicianIds);
+    Map<Long, List<MusicianGenre>> musicianGenresByMusicianId = musicianGenres.stream()
+        .collect(Collectors.groupingBy(MusicianGenre::getMusicianId));
```

```
+
+     List<MusicianTypeOfMusician> musicianTypes = musicianTypeOfMusicianRepository.findByMusicianIdIn(musicianIds);
+     Map<Long, List<MusicianTypeOfMusician>> musicianTypesByMusicianId = musicianTypes.stream()
+          .collect(Collectors.groupingBy(MusicianTypeOfMusician::getMusicianId));
+
+     List<MusicianProduct> musicianProducts = musicianProductRepository.findByMusicianIdIn(musicianIds);
+     Map<Long, List<MusicianProduct>> musicianProductsByMusicianId = musicianProducts.stream()
+          .collect(Collectors.groupingBy(MusicianProduct::getMusicianId));
+
+     return musicians.stream()
+          .map(musician -> convertToDTO(
+               musician,
+               musicianGenresByMusicianId.getOrDefault(musician.getId(), List.of()),
+               musicianTypesByMusicianId.getOrDefault(musician.getId(), List.of()),
+               musicianProductsByMusicianId.getOrDefault(musician.getId(), List.of())))
+          .toList();
+  }
+
   public MusicianGenreDTO getMusiciansByGenre(Long musicianId) {
     log.info("Fetching genres for musician with id: {}", musicianId);
     Musician musician = musicianRepository.findById(musicianId)
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/UserService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/UserService.java
index 10106d8..a44d18f 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/UserService.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/UserService.java
@@ -1,6 +1,8 @@
 package itmo.is.cw.GuitarMatchIS.service;

 import java.util.List;
+import java.util.Map;
+import java.util.stream.Collectors;

 import org.springframework.security.core.userdetails.UsernameNotFoundException;
 import org.springframework.stereotype.Service;
@@ -113,14 +115,36 @@ public class UserService {
         User user = findUserByRequest(request);
         log.info("Getting subscribed musicians for user: {}", user.getUsername());
         List<Musician> musicians = userMusicianRepository.findByUser(user).stream()
-                .map(UserMusician::getMusician).toList();
+                .map(UserMusician::getMusician)
+                .toList();
+
+        if (musicians.isEmpty()) {
+            return List.of();
+        }
+
+        List<Long> musicianIds = musicians.stream()
+                .map(Musician::getId)
+                .toList();
+
+        List<MusicianGenre> musicianGenres = musicianGenreRepository.findByMusicianIdIn(musicianIds);
+        Map<Long, List<MusicianGenre>> musicianGenresByMusicianId = musicianGenres.stream()
+                .collect(Collectors.groupingBy(MusicianGenre::getMusicianId));
+
+        List<MusicianTypeOfMusician> musicianTypes = musicianTypeOfMusicianRepository.findByMusicianIdIn(musicianIds);
+               Map<Long, List<MusicianTypeOfMusician>> musicianTypesByMusicianId = musicianTypes.stream()
+                           .collect(Collectors.groupingBy(MusicianTypeOfMusician::getMusicianId));
+
+               List<MusicianProduct> musicianProducts = musicianProductRepository.findByMusicianIdIn(musicianIds);
+               Map<Long, List<MusicianProduct>> musicianProductsByMusicianId = musicianProducts.stream()
+                           .collect(Collectors.groupingBy(MusicianProduct::getMusicianId));

               return musicians
                           .stream()
-                          .map(musician1 -> convertToDTO(musician1,
-                                      musicianGenreRepository.findByMusician(musician1),
-                                      musicianTypeOfMusicianRepository.findByMusician(musician1),
-                                      musicianProductRepository.findByMusician(musician1)))
+                           .map(musician1 -> convertToDTO(
+                                       musician1,
+                                       musicianGenresByMusicianId.getOrDefault(musician1.getId(), List.of()),
+                                       musicianTypesByMusicianId.getOrDefault(musician1.getId(), List.of()),
+                                       musicianProductsByMusicianId.getOrDefault(musician1.getId(), List.of())))
                          .toList();
       }

diff --git a/docs/refactoring/3-it/1.png b/docs/refactoring/3-it/1.png
new file mode 100644
index 0000000..e999fc7
Binary files /dev/null and b/docs/refactoring/3-it/1.png differ
diff --git a/docs/refactoring/3-it/2.png b/docs/refactoring/3-it/2.png
new file mode 100644
index 0000000..08628d0
Binary files /dev/null and b/docs/refactoring/3-it/2.png differ
diff --git a/docs/refactoring/3-it/3.png b/docs/refactoring/3-it/3.png
new file mode 100644
index 0000000..dbeda35
Binary files /dev/null and b/docs/refactoring/3-it/3.png differ
diff --git a/docs/refactoring/3-it/optimizations.md b/docs/refactoring/3-it/optimizations.md
new file mode 100644
index 0000000..3d43f1f
--- /dev/null
+++ b/docs/refactoring/3-it/optimizations.md
@@ -0,0 +1,68 @@
+# Оптимизации
+
+## Анализ времени выполнения
+
+Для начала мы целых 7 минут с помощью специального запроса из [скрипта](../../../SQL/insertTestData.sql) вставляли тестовые данные в нашу БД. Оценку количества и соотношения строк в таблицах мы проводили ещё во время разработки данного приложения
+
+Далее с помощью Prometheus мы можем посмотреть самые тяжелые запросы по 95-му перцентилю времени ответа командой:
+
+```sql
+topk(10,
+  histogram_quantile(
+    0.95,
+    sum by (uri, le) (
+      http_server_requests_seconds_bucket{uri!~"/actuator/.*"}
+    )
+  )
```

+)

+```

+

+и увидели следующие результаты:

+

+![График](./1.png)

+

+Благодаря уже имеющимся индексам любой фильтр по продуктам работает быстрее, чем авторизация, причём в 4 раза! Кроме того, видим также довольно медленную обработку запроса при следующих действиях:

+

+- открытие страницы с музыкантами(-ом)

+- авторизация

+- открытие страницы со статьями

+

+Проанализируем каждую из проблем по отдельности и посмотрим, как можно это решить

+

+### Оптимизация страницы с музыкантами

+

+Основная причина медленной работы страницы с музыкантами заключалась в паттерне N+1 запросов:

+

+- **Список музыкантов** (`MusicianService.getMusician`) загружал страницу сущностей `Musician` через `MusicianRepository.findAll(page)`, а затем

+ для *каждого* музыканта отдельно делал запросы в таблицы `musician_genre`, `type_of_musician_musician` и `musician_product` через соответствующие репозитории.

+- **Список подписок пользователя** (`UserService.getSubscribedMusicians`) повторял ту же схему: для каждого музыканта в подписках выполнялись отдельные запросы за жанрами, типами и продуктами.

+

+Это приводило к росту количества запросов к БД линейно от числа музыкантов на странице (3 дополнительных запроса на музыканта только по связям, не считая загрузки самих продуктов).

+

+Мы переписали эту логику на батчевую выборку:

+

+- В `MusicianGenreRepository`, `MusicianTypeOfMusicianRepository` и `MusicianProductRepository` добавлены методы `findByMusicianIdIn(List<Long> musicianIds)`, которые делают один запрос с `WHERE musician_id IN (...)` вместо множества одиночных запросов.

+- В `MusicianService` добавлен приватный метод `buildMusicianInfoDTOs(...)`, который:

+ - собирает все `musician_id` с текущей страницы,

+ - одним батчем загружает все жанры, типы и продукты для этих музыкантов,

+ - группирует результаты в мапы `musician_id -> список сущностей` и

+ - на их основе формирует `MusicianInfoDTO` без дополнительных обращений к БД.

+- В `UserService.getSubscribedMusicians` реализована аналогичная батчевая агрегация по списку музыкантов, на которых подписан пользователь.

+

+Таким образом, вместо \(1 + 3 \cdot N\) запросов (страница музыкантов + три запроса по связям на каждого музыканта) теперь выполняется всего 4 запроса независимо от размера страницы. Это уменьшает нагрузку на БД и сокращает время ответа для эндпоинтов, связанных со списком музыкантов.

+

+### Результаты оптимизации

+

+Было:

+

+![Было музыканты](./2.png)

+

+![Стало музыканты](./3.png)

+

+Таким образом, снизили время ответа более чем в 2 раза!

+

```
+Остальные метрики тоже стали чуть лучше, потому что для оптимизации этой метрики другие запросы тоже пришлось поправить, но об этом далее
+
+### Оптимизация страницы со статьями
+
+
```

git show 0de8c0a

```
commit 0de8c0afa9e9a901c96423344974893eb4a6da2d
Author: ta4ilka69 <artem_balin_2004@mail.ru>
Date:   Tue Dec 2 00:07:19 2025 +0300

    article optimization

diff --git a/SQL/initScript.sql b/SQL/initScript.sql
index d6da3ee..fb210cd 100644
--- a/SQL/initScript.sql
+++ b/SQL/initScript.sql
@@ -158,6 +158,7 @@ CREATE TABLE articles (
     author_id INTEGER NOT NULL,
     created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
     accepted BOOLEAN DEFAULT FALSE NOT NULL,
+    html_content TEXT,
     CONSTRAINT fk_articles_user FOREIGN KEY (author_id) REFERENCES app_user (id) ON DELETE SET NULL
 );

diff --git a/SQL/insertTestData.sql b/SQL/insertTestData.sql
index 87180b0..73c1f38 100644
--- a/SQL/insertTestData.sql
+++ b/SQL/insertTestData.sql
@@ -43,13 +43,16 @@ SELECT
 FROM generate_series(1, 130000);

 -- Добавление статей
-INSERT INTO articles (header, text, author_id, created_at, accepted)
+INSERT INTO articles (header, text, author_id, created_at, accepted, html_content)
 SELECT
     'Article Header ' || generate_series,
     'Article Text ' || generate_series,
     floor(random() * 13000 + 1)::int,
     current_date - (random() * 365)::int,
-    random() > 0.5
+    random() > 0.5,
+    -- Для тестовых данных html_content оставляем равным text,
+    -- чтобы не тратить время на внешний парсер при выборке
+    'Article Text ' || generate_series
 FROM generate_series(1, 1000);

 -- Добавление отзывов
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/models/Article.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/models/Article.java
index 3eeb12f..ee933f1 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/models/Article.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/models/Article.java
@@ -32,4 +32,7 @@ public class Article {
    @Column(name = "accepted", nullable = false)
    private Boolean accepted;
+
+   @Column(name = "html_content")
+   private String htmlContent;
 }
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/ArticleRepository.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/ArticleRepository.java
index 6dfb831..affa016 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/ArticleRepository.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/repository/ArticleRepository.java
@@ -1,15 +1,16 @@
 package itmo.is.cw.GuitarMatchIS.repository;

+import org.springframework.data.domain.Page;
+import org.springframework.data.domain.Pageable;
+import org.springframework.data.jpa.repository.EntityGraph;
 import org.springframework.data.jpa.repository.JpaRepository;
 import org.springframework.data.jpa.repository.Query;
 import org.springframework.stereotype.Repository;
+
 import itmo.is.cw.GuitarMatchIS.models.Article;

 import java.util.List;

-import org.springframework.data.domain.Page;
-import org.springframework.data.domain.Pageable;
-
 @Repository
 public interface ArticleRepository extends JpaRepository<Article, Long> {
    boolean existsByHeader(String header);
@@ -17,12 +18,16 @@ public interface ArticleRepository extends JpaRepository<Article, Long> {
    @Query("SELECT moderate_article(:articleId, :accepted, :moderatorId)")
    boolean moderateArticle(Long articleId, boolean accepted, Long moderatorId);

+   @EntityGraph(attributePaths = "author")
    Page<Article> findByHeaderContainingAndAccepted(String header, boolean accepted, Pageable page);

+   @EntityGraph(attributePaths = "author")
    Page<Article> findByAuthorIdAndAccepted(Long authorId, boolean accepted, Pageable page);

+   @EntityGraph(attributePaths = "author")
    Page<Article> findByAccepted(boolean accepted, Pageable page);

+   @EntityGraph(attributePaths = "author")
    List<Article> findByAccepted(boolean accepted);

 }
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ArticleService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ArticleService.java
index 2981035..acf6002 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ArticleService.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/ArticleService.java
@@ -162,6 +162,7 @@ public class ArticleService {
          .author(author)
          .createdAt(LocalDateTime.now())
```

```
            .accepted(false)
+           .htmlContent(convertMarkdownToHtml(createArticleDTO.getText()))
            .build();
        articleRepository.save(article);
        log.info("Article with header {} saved with id {}", article.getHeader(), article.getId());
@@ -203,7 +204,10 @@ public class ArticleService {
    }

    private ArticleDTO convertToDTO(Article article) {
-       String htmlContent = convertMarkdownToHtml(article.getText());
+       String htmlContent = article.getHtmlContent();
+       if (htmlContent == null || htmlContent.isBlank()) {
+           htmlContent = convertMarkdownToHtml(article.getText());
+       }

        return ArticleDTO.builder()
            .id(article.getId())
diff --git a/docs/refactoring/3-it/4.png b/docs/refactoring/3-it/4.png
new file mode 100644
index 0000000..75d0688
Binary files /dev/null and b/docs/refactoring/3-it/4.png differ
diff --git a/docs/refactoring/3-it/5.png b/docs/refactoring/3-it/5.png
new file mode 100644
index 0000000..c2fd754
Binary files /dev/null and b/docs/refactoring/3-it/5.png differ
diff --git a/docs/refactoring/3-it/optimizations.md b/docs/refactoring/3-it/optimizations.md
index 3d43f1f..b6c4175 100644
--- a/docs/refactoring/3-it/optimizations.md
+++ b/docs/refactoring/3-it/optimizations.md
@@ -29,7 +29,9 @@ topk(10,

 Проанализируем каждую из проблем по отдельности и посмотрим, как можно это решить

-### Оптимизация страницы с музыкантами
+## Оптимизация страницы с музыкантами
+
+### План оптимизации по музыкантам

 Основная причина медленной работы страницы с музыкантами заключалась в паттерне N+1 запросов:

@@ -51,7 +53,7 @@ topk(10,
```

Таким образом, вместо \(1 + 3 \cdot N\) запросов (страница музыкантов + три запроса по связям на каждого музыканта) теперь выполняется всего 4 запроса независимо от размера страницы. Это уменьшает нагрузку на БД и сокращает время ответа для эндпоинтов, связанных со списком музыкантов.

```
-### Результаты оптимизации
+### Результаты оптимизации по музыкантам

 Было:

@@ -63,6 +65,56 @@ topk(10,
```

Остальные метрики тоже стали чуть лучше, потому что для оптимизации этой метрики другие запросы тоже пришлось поправить, но об этом далее

-### Оптимизация страницы со статьями

+## Оптимизация страницы со статьями

+

+### План оптимизации по статьям

+

+Для эндпоинта `/api/article` основной вклад во время ответа давала не база данных, а CPU-нагрузка на бэкенд:

+

+- Для каждой статьи в списке вызывался метод `convertToDTO`, который в свою очередь запускал внешний Markdown-парсер через `ProcessBuilder` (метод `convertMarkdownToHtml`).

+- Это происходило **для каждой статьи в выдаче** и каждый раз при запросе списка, то есть на один HTTP-запрос приходилось десятки запусков внешнего процесса.

+- Дополнительно присутствовал паттерн N+1 по авторам статей: для каждой статьи ленивая загрузка `author` порождала отдельный SELECT.

+

+Мы оптимизировали этот путь следующим образом:

+

+- Добавили в таблицу `articles` поле `html_content TEXT` и соответствующее поле в сущность `Article`.

+- При создании статьи (`ArticleService.createArticle`) HTML-представление текста теперь вычисляется **один раз**:

+

```
+  ```java
+  Article article = Article.builder()
+        .header(createArticleDTO.getHeader())
+        .text(createArticleDTO.getText())
+        .author(author)
+        .createdAt(LocalDateTime.now())
+        .accepted(false)
+        .htmlContent(convertMarkdownToHtml(createArticleDTO.getText()))
+        .build();
+  ```
```

+

+- В методе `convertToDTO` мы больше не запускаем парсер для каждой статьи:

+ - сначала берём уже сохранённое `article.getHtmlContent()`,

+ - и только если оно по каким-то причинам пустое (старые данные), вычисляем HTML через парсер и используем его.

+- В `ArticleRepository` на методы выборки (`findByAccepted`, `findByHeaderContainingAndAccepted`, `findByAuthorIdAndAccepted`) добавлен `@EntityGraph(attributePaths = "author")`, чтобы автор подгружался одним JOIN-ом, без N+1.

+- В тестовом скрипте `insertTestData.sql` поле `html_content` для 1000 сгенерированных статей заполняется сразу (`html_content = text`), чтобы при бенчмарках не тратить время на парсинг вообще.

+

+Итог:

+

+- При запросе списка статей `/api/article` теперь нет многократных запусков внешнего парсера — мы просто читаем уже готовый HTML из поля `html_content`.

+- Число SQL-запросов для одной страницы статей снижено до фиксированного количества за счёт использования `@EntityGraph` вместо ленивой загрузки автора для каждой статьи.

+- Это существенно уменьшает среднее и p95 время ответа для `/api/article` и освобождает CPU для обработки других запросов.

+

+### Результаты оптимизации по статьям

+

+Было:

+

+![Было статьи](./5.png)

+

+![Стало статьи](./4.png)

+

+Видим, что было 0.13, стало 0.03, что стало является существенным ускорением! Конечно, мы увеличили размер таблицы на целую строчку, статьи могут быть довольно большими, однако размер html-рендера не сильно больше, чем размер исходной markdown-разметки. Также не стоит обращать внимания на другие метрики, так как я их оптимизировал независимо. В конце будет сравнение, когда будут работать сразу все оптимизации (вдруг станет плохо?)

```
+
+## Авторизация

+### План оптимизации по авторизации

+### Результаты по оптимазции авторизации
```

git show 22cfe2f

```
commit 22cfe2fd3daab9da0a4b161e38bc747e7cea1151
Author: ta4ilka69 <artem_balin_2004@mail.ru>
Date:   Tue Dec 2 00:31:50 2025 +0300

    login optimization

diff --git a/.docker/docker-compose.yaml b/.docker/docker-compose.yaml
index efad857..ac96aba 100644
--- a/.docker/docker-compose.yaml
+++ b/.docker/docker-compose.yaml
@@ -112,6 +112,7 @@ services:
       POSTGRES_PASSWORD: "pgpwd"
       SPRING_DATASOURCE_URL: "jdbc:postgresql://postgres:5432/postgres"
       JWT_SECRET: "dev-secret"
+      BCRYPT_STRENGTH: "8"
       # Явно указываем путь к собранному Linux-бинарнику парсера внутри контейнера
       APP_EXTERNAL_MARKDOWN_PARSER_PATH: "/opt/app/parser"
     volumes:
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/security/SecurityConfig.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/security/SecurityConfig.java
index 44eee9b..983822a 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/security/SecurityConfig.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/security/SecurityConfig.java
@@ -36,6 +36,9 @@ public class SecurityConfig {
   @Value("#{'${app.cors.allowed-origins:http://localhost:5173}'.split(',')}")
   private List<String> corsAllowedOrigins;

+  @Value("${app.security.bcrypt-strength:10}")
+  private int bcryptStrength;
+

   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
@@ -77,7 +80,7 @@ public class SecurityConfig {

   @Bean
   public PasswordEncoder passwordEncoder() {
-    return new BCryptPasswordEncoder();
+    return new BCryptPasswordEncoder(bcryptStrength);
   }

   @Bean
diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/AuthService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/AuthService.java
index 5a97741..8603d5b 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/AuthService.java
```

```
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/AuthService.java
@@ -6,6 +6,7 @@ import java.time.LocalDateTime;

 import org.springframework.security.authentication.AuthenticationManager;
 import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
+import org.springframework.security.core.Authentication;
 import org.springframework.security.crypto.password.PasswordEncoder;

 import itmo.is.cw.GuitarMatchIS.dto.AuthResponseDTO;
@@ -14,7 +15,6 @@ import itmo.is.cw.GuitarMatchIS.models.User;
 import itmo.is.cw.GuitarMatchIS.repository.UserRepository;
 import itmo.is.cw.GuitarMatchIS.security.jwt.JwtUtils;
 import itmo.is.cw.GuitarMatchIS.utils.exceptions.UserAlreadyExistException;
-import itmo.is.cw.GuitarMatchIS.utils.exceptions.UserNotFoundException;
 import lombok.RequiredArgsConstructor;
 import lombok.extern.slf4j.Slf4j;

@@ -57,17 +57,14 @@ public class AuthService {

        public AuthResponseDTO login(UserDTO loginUserDto) {
                log.info("Logging in user with username: {}", loginUserDto.getUsername());
-               User user = userRepository.findByUsername(loginUserDto.getUsername())
-                               .orElseThrow(() -> {
-                                       log.warn("User with username {} not found", loginUserDto.getUsername());
-                                       return new UserNotFoundException(
-                                                       String.format("Username %s not found", loginUserDto.getUsername()));
-                               });

-               authenticationManager.authenticate(
-                               new UsernamePasswordAuthenticationToken(loginUserDto.getUsername(),
+               Authentication authentication = authenticationManager.authenticate(
+                               new UsernamePasswordAuthenticationToken(
+                                               loginUserDto.getUsername(),
                                                loginUserDto.getPassword()));

+               User user = (User) authentication.getPrincipal();
+
                String token = jwtUtils.generateJwtToken(user.getUsername());
                log.info("User with username {} successfully logged in", user.getUsername());
                return new AuthResponseDTO(
diff --git a/backend/src/main/resources/application.properties b/backend/src/main/resources/application.properties
index a2922a5..c65a7ef 100644
--- a/backend/src/main/resources/application.properties
+++ b/backend/src/main/resources/application.properties
@@ -24,6 +24,8 @@ app.websocket.allowed-origins=http://localhost:5173
 app.messaging.feedback-topic=/feedbacks
 app.messaging.articles-topic=/articles

+app.security.bcrypt-strength=${BCRYPT_STRENGTH:10}
+
 app.external.markdown-parser.path=${user.dir}/parser

 #logging
diff --git a/docs/refactoring/3-it/6.png b/docs/refactoring/3-it/6.png
new file mode 100644
index 0000000..2589ac9
```

Binary files /dev/null and b/docs/refactoring/3-it/6.png differ

diff --git a/docs/refactoring/3-it/7.png b/docs/refactoring/3-it/7.png

new file mode 100644

index 0000000..9c6b5f4

Binary files /dev/null and b/docs/refactoring/3-it/7.png differ

diff --git a/docs/refactoring/3-it/optimizations.md b/docs/refactoring/3-it/optimizations.md

index b6c4175..3123bb3 100644

--- a/docs/refactoring/3-it/optimizations.md

+++ b/docs/refactoring/3-it/optimizations.md

@@ -117,4 +117,43 @@ topk(10,

 ### План оптимизации по авторизации

-### Результаты по оптимазции авторизации

+Для авторизации проблемным оказался путь `/api/auth` (регистрация и логин). По графику Prometheus видно, что p95 по времени ответа здесь заметно выше, чем у большинства других эндпоинтов.

+

+План был таким:

+

+- **Не трогать бизнес-логику, связанную с безопасностью**, но убрать лишние накладные расходы:

+ - избавиться от лишнего SQL-запроса при логине;

+ - сделать «силу» BCrypt конфигурируемой через настройки приложения и переменные окружения.

+- **Уменьшить CPU-нагрузку на тестовом окружении** (Docker) за счёт уменьшения «стоимости» BCrypt:

+ - по умолчанию оставить безопасное значение (10 раундов),

+ - в docker-окружении для нагрузочного тестирования выставить меньшее значение (8), чтобы показать влияние параметра на производительность.

+

+Конкретные изменения:

+

+- В `AuthService.login` убран лишний запрос к БД:

+ - раньше сначала выполнялся `userRepository.findByUsername(...)`, а потом `authenticationManager.authenticate(...)`, который снова грузил пользователя через `UserDetailsService`;

+ - теперь мы один раз вызываем `authenticationManager.authenticate(...)` и берём авторизованного пользователя из `authentication.getPrincipal()` (это тот же `User`, реализующий `UserDetails`), избавившись от лишнего SELECT.

+- В `SecurityConfig` параметр BCrypt вынесен в конфигурацию:

+ - добавлено свойство `app.security.bcrypt-strength` (в `application.properties` оно берётся из переменной окружения `BCRYPT_STRENGTH`, по умолчанию — `10`);

+ - бин `PasswordEncoder` создаётся как `new BCryptPasswordEncoder(bcryptStrength)`.

+- В `.docker/docker-compose.yaml` для сервиса `backend` установлено `BCRYPT_STRENGTH: "8"`, что ускоряет регистрацию/логин в тестовом стенде без изменения дефолтной конфигурации приложения.

+

+### Результаты по оптимизации авторизации

+

+Теперь проведём тест всех оптимизаций. Вспомним, как было до них:

+

+![Было всё](./1.png)

+

+А стало:

+

+![Стало всё](./6.png)

+

+В первое время пару запросов были долгими, но приложение было только запущено, а кеш и БД не были "прогреты". Дальше видим, что среднее время ответа (по p95) от любой ручки не превышает 0.2. Можно заметить это и по дашборду:

+

+![Дашборд](./7.png)

+

+На графике отмечено тестирование до оптимизаций и финальное тестирование всех оптимизаций, среднее время ответа по всем rest-ручкам снижено в 2.5-3 раза.

+

+## Заключение

+

+

git show 9d90f61

```
commit 9d90f61efe81c1ecf2ad21cad30a29251ede9793
Author: ta4ilka69 <artem_balin_2004@mail.ru>
Date:   Tue Dec 2 00:33:23 2025 +0300

    overall results in report

diff --git a/docs/refactoring/3-it/optimizations.md b/docs/refactoring/3-it/optimizations.md
index 3123bb3..e3e9656 100644
--- a/docs/refactoring/3-it/optimizations.md
+++ b/docs/refactoring/3-it/optimizations.md
@@ -156,4 +156,6 @@ topk(10,

 ## Заключение
```

+В рамках данного этапа мы последовательно проанализировали самые «тяжёлые» пути по p95 времени ответа и оптимизировали их на трёх уровнях: запросы к БД, бизнес-логику в сервисах и конфигурацию инфраструктуры. Для страницы с музыкантами мы устранили N+1-запросы за жанрами, типами и продуктами, сведя число SQL-запросов на страницу к фиксированному количеству и сократив время ответа более чем в два раза. Для страницы со статьями мы вынесли парсинг Markdown из hot-path'а, ввели предрасчитанный `html_content` и убрали N+1 по авторам, что снизило p95 по `/api/article` с ~0.13 до ~0.03 секунды.

+В авторизации мы избавились от лишнего запроса к базе при логине и сделали «силу» BCrypt конфигурируемой, что уменьшило CPU-нагрузку и время ответа для `/api/auth` без ущерба для безопасности. Итоговое сравнение метрик в Prometheus показывает, что после применения всех оптимизаций совокупное среднее время ответа по REST-ручкам снизилось примерно в 2.5–3 раза, а ни одна из внесённых доработок не ухудшила стабильность и функциональность системы, что позволяет считать цели этапа по оптимизации производительности достигнутыми.

git show d0e813b

```
commit d0e813bd1c64f2cbfb482bf040ad0cdccee15ce1
Author: ta4ilka69 <artem_balin_2004@mail.ru>
Date:   Tue Dec 2 01:10:15 2025 +0300

    lint

diff --git a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
index a7ad82f..dca5bb7 100644
--- a/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
+++ b/backend/src/main/java/itmo/is/cw/GuitarMatchIS/service/MusicianService.java
@@ -57,317 +57,333 @@ import java.util.stream.Collectors;
 @RequiredArgsConstructor
 @Slf4j
 public class MusicianService {
-   private final MusicianRepository musicianRepository;
-   private final MusicianGenreRepository musicianGenreRepository;
-   private final MusicianTypeOfMusicianRepository musicianTypeOfMusicianRepository;
-   private final JwtUtils jwtUtils;
-   private final UserMusicianRepository userMusicianRepository;
-   private final SimpMessagingTemplate simpMessagingTemplate;
-   private final UserRepository userRepository;
-   private final MusicianProductRepository musicianProductRepository;
-   private final ProductRepository productRepository;
-
-   public List<MusicianInfoDTO> getMusician(int from, int size, MusicianSort sortBy, boolean ascending) {
-     log.info("Fetching musicians from: {}, size: {}, sortBy: {}, ascending: {}", from, size, sortBy, ascending);
-     Sort sort = Sort.by(ascending ? Sort.Direction.ASC : Sort.Direction.DESC,
-         sortBy.getFieldName());
-     Pageable page = PageRequest.of(from / size, size, sort);
-     List<Musician> musicians = musicianRepository.findAll(page).getContent();
-
-     return buildMusicianInfoDTOs(musicians);
-   }
-
-   public Boolean isSubscribed(Long musicianId, HttpServletRequest request) {
-     User user = findUserByRequest(request);
-     log.info("Checking if user {} is subscribed to musician with id: {}", user.getUsername(), musicianId);
-     return userMusicianRepository.existsByUserAndMusician(user,
-         musicianRepository.findById(musicianId).orElseThrow(() -> {
-           log.warn("Musician with id {} not found while checking subscription", musicianId);
-           return new MusicianNotFoundException("Musician with id %s not found".formatted(musicianId));
-         }));
-   }
-
-   @Transactional
-   public MusicianInfoDTO createMusician(CreateMusicianDTO createMusicianDTO, HttpServletRequest request) {
-     User user = findUserByRequest(request);
-     log.info("User {} is creating musician with name: {}", user.getUsername(), createMusicianDTO.getName());
-     if (musicianRepository.existsByName(createMusicianDTO.getName())) {
-       log.warn("Musician with name {} already exists", createMusicianDTO.getName());
-       throw new MusicianAlreadyExistsException("Musician %s already exists".formatted(createMusicianDTO.getName()));
+     private final MusicianRepository musicianRepository;
+     private final MusicianGenreRepository musicianGenreRepository;
+     private final MusicianTypeOfMusicianRepository musicianTypeOfMusicianRepository;
+     private final JwtUtils jwtUtils;
+     private final UserMusicianRepository userMusicianRepository;
+     private final SimpMessagingTemplate simpMessagingTemplate;
+     private final UserRepository userRepository;
+     private final MusicianProductRepository musicianProductRepository;
+     private final ProductRepository productRepository;
+
+     public List<MusicianInfoDTO> getMusician(int from, int size, MusicianSort sortBy, boolean ascending) {
+         log.info("Fetching musicians from: {}, size: {}, sortBy: {}, ascending: {}", from, size, sortBy, ascending);
+         Sort sort = Sort.by(ascending ? Sort.Direction.ASC : Sort.Direction.DESC,
+                 sortBy.getFieldName());
+         Pageable page = PageRequest.of(from / size, size, sort);
+         List<Musician> musicians = musicianRepository.findAll(page).getContent();
+
+         return buildMusicianInfoDTOs(musicians);
      }
-     Musician musician = Musician.builder()
-         .name(createMusicianDTO.getName())
-         .subscribers(0)
-         .build();

-     musician = musicianRepository.save(musician);
-    log.info("Musician with name {} successfully created with id {}", musician.getName(), musician.getId());
+    public Boolean isSubscribed(Long musicianId, HttpServletRequest request) {
+        User user = findUserByRequest(request);
+        log.info("Checking if user {} is subscribed to musician with id: {}", user.getUsername(), musicianId);
+        return userMusicianRepository.existsByUserAndMusician(user,
+                musicianRepository.findById(musicianId).orElseThrow(() -> {
+                    log.warn("Musician with id {} not found while checking subscription", musicianId);
+                    return new MusicianNotFoundException(
+                            "Musician with id %s not found".formatted(musicianId));
+                }));
+    }
+
+    @Transactional
+    public MusicianInfoDTO createMusician(CreateMusicianDTO createMusicianDTO, HttpServletRequest request) {
+        User user = findUserByRequest(request);
+        log.info("User {} is creating musician with name: {}", user.getUsername(), createMusicianDTO.getName());
+        if (musicianRepository.existsByName(createMusicianDTO.getName())) {
+            log.warn("Musician with name {} already exists", createMusicianDTO.getName());
+            throw new MusicianAlreadyExistsException(
+                    "Musician %s already exists".formatted(createMusicianDTO.getName()));
+        }
+        Musician musician = Musician.builder()
+                .name(createMusicianDTO.getName())
+                .subscribers(0)
+                .build();
+
+        musician = musicianRepository.save(musician);
+        log.info("Musician with name {} successfully created with id {}", musician.getName(), musician.getId());
+
+        for (Genre genre : createMusicianDTO.getGenres()) {
+            musicianGenreRepository.saveByMusicianIdAndGenre(musician.getId(), genre.toString());
+        }
+
+        for (TypeOfMusician typeOfMusician : createMusicianDTO.getTypesOfMusician()) {
+            musicianTypeOfMusicianRepository.saveByMusicianIdAndTypeOfMusician(musician.getId(),
+                    typeOfMusician.toString());
+        }
+
+        simpMessagingTemplate.convertAndSend("/musicians", "New musician added");
+        log.info("Musician creation message sent to websocket");
+
+        return convertToDTOLists(musician, createMusicianDTO.getGenres(), createMusicianDTO.getTypesOfMusician(),
+                new ArrayList<>());
+    }

-    for (Genre genre : createMusicianDTO.getGenres()) {
-      musicianGenreRepository.saveByMusicianIdAndGenre(musician.getId(), genre.toString());
+    public Boolean subscribeToMusician(SubscribeDTO subscribeDTO, HttpServletRequest request) {
+        User user = findUserByRequest(request);
+        log.info("User {} is subscribing to musician with id: {}", user.getUsername(),
+                subscribeDTO.getMusicianId());
+
+        Musician musician = musicianRepository.findById(subscribeDTO.getMusicianId())
+                .orElseThrow(() -> {
+                    log.warn("Musician with id {} not found while subscribing", subscribeDTO.getMusicianId());
+                    return new MusicianNotFoundException(
+                               "Musician with id %s not found".formatted(subscribeDTO.getMusicianId()));
+                   });
+           if (userMusicianRepository.existsByUserAndMusician(user, musician)) {
+               log.warn("User {} is already subscribed to musician {}", user.getUsername(), musician.getName());
+               throw new SubscriptionAlreadyExistsException("You are already subscribed to this musician");
+           }
+           simpMessagingTemplate.convertAndSend("/musicians", "New subscriber to musician");
+           userMusicianRepository.subscribeToMusician(user.getId(), subscribeDTO.getMusicianId());
+           log.info("User {} successfully subscribed to musician {}", user.getUsername(), musician.getName());
+           return true;
       }

-     for (TypeOfMusician typeOfMusician : createMusicianDTO.getTypesOfMusician()) {
-       musicianTypeOfMusicianRepository.saveByMusicianIdAndTypeOfMusician(musician.getId(),
-           typeOfMusician.toString());
+      @Transactional
+      public Boolean unsubscribeFromMusician(SubscribeDTO subscribeDTO, HttpServletRequest request) {
+           User user = findUserByRequest(request);
+           log.info("User {} is unsubscribing from musician with id: {}", user.getUsername(),
+                   subscribeDTO.getMusicianId());
+
+           Musician musician = musicianRepository.findById(subscribeDTO.getMusicianId())
+                   .orElseThrow(() -> {
+                       log.warn("Musician with id {} not found while unsubscribing",
+                               subscribeDTO.getMusicianId());
+                       return new MusicianNotFoundException(
+                               "Musician with id %s not found".formatted(subscribeDTO.getMusicianId()));
+                   });
+           if (!userMusicianRepository.existsByUserAndMusician(user, musician)) {
+               log.warn("User {} is not subscribed to musician {}", user.getUsername(), musician.getName());
+               throw new SubscriptionNotFoundException("You have no subscription to this musician");
+           }
+           simpMessagingTemplate.convertAndSend("/musicians", "Deleted subscription to musician");
+           userMusicianRepository.deleteByUserAndMusician(user, musician);
+           log.info("User {} successfully unsubscribed from musician {}", user.getUsername(), musician.getName());
+           return true;
       }

-     simpMessagingTemplate.convertAndSend("/musicians", "New musician added");
-     log.info("Musician creation message sent to websocket");
-
-     return convertToDTOLists(musician, createMusicianDTO.getGenres(), createMusicianDTO.getTypesOfMusician(),
-         new ArrayList<>());
-   }
-
-   public Boolean subscribeToMusician(SubscribeDTO subscribeDTO, HttpServletRequest request) {
-     User user = findUserByRequest(request);
-     log.info("User {} is subscribing to musician with id: {}", user.getUsername(), subscribeDTO.getMusicianId());
-
-     Musician musician = musicianRepository.findById(subscribeDTO.getMusicianId())
-         .orElseThrow(() -> {
-           log.warn("Musician with id {} not found while subscribing", subscribeDTO.getMusicianId());
-           return new MusicianNotFoundException(
-               "Musician with id %s not found".formatted(subscribeDTO.getMusicianId()));
-         });
-     if (userMusicianRepository.existsByUserAndMusician(user, musician)) {
```

```
-        log.warn("User {} is already subscribed to musician {}", user.getUsername(), musician.getName());
-        throw new SubscriptionAlreadyExistsException("You are already subscribed to this musician");
+     public List<MusicianInfoDTO> searchMusicians(String name, int from, int size) {
+         log.info("Searching for musicians with name containing: {}", name);
+         Pageable page = Pagification.createPageTemplate(from, size);
+         List<Musician> musicians = musicianRepository.findAllByNameContains(name, page).getContent();
+
+         return buildMusicianInfoDTOs(musicians)
+                 .stream()
+                 .sorted(Comparator.comparing(MusicianInfoDTO::getId))
+                 .toList();
      }
-     simpMessagingTemplate.convertAndSend("/musicians", "New subscriber to musician");
-     userMusicianRepository.subscribeToMusician(user.getId(), subscribeDTO.getMusicianId());
-     log.info("User {} successfully subscribed to musician {}", user.getUsername(), musician.getName());
-     return true;
-   }
-
-   @Transactional
-   public Boolean unsubscribeFromMusician(SubscribeDTO subscribeDTO, HttpServletRequest request) {
-     User user = findUserByRequest(request);
-     log.info("User {} is unsubscribing from musician with id: {}", user.getUsername(), subscribeDTO.getMusicianId());
-
-     Musician musician = musicianRepository.findById(subscribeDTO.getMusicianId())
-         .orElseThrow(() -> {
-           log.warn("Musician with id {} not found while unsubscribing", subscribeDTO.getMusicianId());
-           return new MusicianNotFoundException(
-               "Musician with id %s not found".formatted(subscribeDTO.getMusicianId()));
-         });
-     if (!userMusicianRepository.existsByUserAndMusician(user, musician)) {
-        log.warn("User {} is not subscribed to musician {}", user.getUsername(), musician.getName());
-        throw new SubscriptionNotFoundException("You have no subscription to this musician");
+
+     private List<MusicianInfoDTO> buildMusicianInfoDTOs(List<Musician> musicians) {
+         if (musicians.isEmpty()) {
+             return List.of();
+         }
+
+         List<Long> musicianIds = musicians.stream()
+                 .map(Musician::getId)
+                 .toList();
+
+         List<MusicianGenre> musicianGenres = musicianGenreRepository.findByMusicianIdIn(musicianIds);
+         Map<Long, List<MusicianGenre>> musicianGenresByMusicianId = musicianGenres.stream()
+                 .collect(Collectors.groupingBy(MusicianGenre::getMusicianId));
+
+         List<MusicianTypeOfMusician> musicianTypes = musicianTypeOfMusicianRepository
+                 .findByMusicianIdIn(musicianIds);
+         Map<Long, List<MusicianTypeOfMusician>> musicianTypesByMusicianId = musicianTypes.stream()
+                 .collect(Collectors.groupingBy(MusicianTypeOfMusician::getMusicianId));
+
+         List<MusicianProduct> musicianProducts = musicianProductRepository.findByMusicianIdIn(musicianIds);
+         Map<Long, List<MusicianProduct>> musicianProductsByMusicianId = musicianProducts.stream()
+                 .collect(Collectors.groupingBy(MusicianProduct::getMusicianId));
+
+         return musicians.stream()
+                     .map(musician -> convertToDTO(
+                             musician,
+                             musicianGenresByMusicianId.getOrDefault(musician.getId(), List.of()),
+                             musicianTypesByMusicianId.getOrDefault(musician.getId(), List.of()),
+                             musicianProductsByMusicianId.getOrDefault(musician.getId(), List.of())))
+                     .toList();
     }
-    simpMessagingTemplate.convertAndSend("/musicians", "Deleted subscription to musician");
-    userMusicianRepository.deleteByUserAndMusician(user, musician);
-    log.info("User {} successfully unsubscribed from musician {}", user.getUsername(), musician.getName());
-    return true;
-  }
-
-  public List<MusicianInfoDTO> searchMusicians(String name, int from, int size) {
-    log.info("Searching for musicians with name containing: {}", name);
-    Pageable page = Pagification.createPageTemplate(from, size);
-    List<Musician> musicians = musicianRepository.findAllByNameContains(name, page).getContent();
-
-    return buildMusicianInfoDTOs(musicians)
-        .stream()
-        .sorted(Comparator.comparing(MusicianInfoDTO::getId))
-        .toList();
-  }
-
-  private List<MusicianInfoDTO> buildMusicianInfoDTOs(List<Musician> musicians) {
-    if (musicians.isEmpty()) {
-      return List.of();
+
+     public MusicianGenreDTO getMusiciansByGenre(Long musicianId) {
+         log.info("Fetching genres for musician with id: {}", musicianId);
+         Musician musician = musicianRepository.findById(musicianId)
+                 .orElseThrow(() -> new MusicianNotFoundException(
+                         "Musician with id %s not found".formatted(musicianId)));
+
+         List<MusicianGenre> musicianGenres = musicianGenreRepository.findByMusician(musician);
+
+         return MusicianGenreDTO.builder()
+                 .musician(musician)
+                 .genres(musicianGenres.stream().map(MusicianGenre::getGenre).toList())
+                 .build();
     }

-    List<Long> musicianIds = musicians.stream()
-        .map(Musician::getId)
-        .toList();
-
-    List<MusicianGenre> musicianGenres = musicianGenreRepository.findByMusicianIdIn(musicianIds);
-    Map<Long, List<MusicianGenre>> musicianGenresByMusicianId = musicianGenres.stream()
-        .collect(Collectors.groupingBy(MusicianGenre::getMusicianId));
-
-    List<MusicianTypeOfMusician> musicianTypes = musicianTypeOfMusicianRepository.findByMusicianIdIn(musicianIds);
-    Map<Long, List<MusicianTypeOfMusician>> musicianTypesByMusicianId = musicianTypes.stream()
-        .collect(Collectors.groupingBy(MusicianTypeOfMusician::getMusicianId));
-
-    List<MusicianProduct> musicianProducts = musicianProductRepository.findByMusicianIdIn(musicianIds);
-    Map<Long, List<MusicianProduct>> musicianProductsByMusicianId = musicianProducts.stream()
-           .collect(Collectors.groupingBy(MusicianProduct::getMusicianId));
-
-       return musicians.stream()
-           .map(musician -> convertToDTO(
-               musician,
-               musicianGenresByMusicianId.getOrDefault(musician.getId(), List.of()),
-               musicianTypesByMusicianId.getOrDefault(musician.getId(), List.of()),
-               musicianProductsByMusicianId.getOrDefault(musician.getId(), List.of())))
-           .toList();
-   }
-
-   public MusicianGenreDTO getMusiciansByGenre(Long musicianId) {
-       log.info("Fetching genres for musician with id: {}", musicianId);
-       Musician musician = musicianRepository.findById(musicianId)
-           .orElseThrow(() -> new MusicianNotFoundException(
-               "Musician with id %s not found".formatted(musicianId)));
-
-       List<MusicianGenre> musicianGenres = musicianGenreRepository.findByMusician(musician);
-
-       return MusicianGenreDTO.builder()
-           .musician(musician)
-           .genres(musicianGenres.stream().map(MusicianGenre::getGenre).toList())
-           .build();
-   }
-
-   public MusicianTypeOfMusicianDTO getMusiciansByTypeOfMusician(Long musicianId) {
-       log.info("Fetching types for musician with id: {}", musicianId);
-       Musician musician = musicianRepository.findById(musicianId)
-           .orElseThrow(() -> new MusicianNotFoundException(
-               "Musician with id %s not found".formatted(musicianId)));
-
-       List<MusicianTypeOfMusician> musicianTypeOfMusicians = musicianTypeOfMusicianRepository.findByMusician(musician);
-
-       return MusicianTypeOfMusicianDTO.builder()
-           .musician(musician)
-           .typeOfMusicians(musicianTypeOfMusicians.stream().map(MusicianTypeOfMusician::getTypeOfMusician).toList())
-           .build();
-   }
-
-   public MusicianProductDTO getMusicianProducts(String name) {
-       log.info("Fetching products for musician with name: {}", name);
-       if (!musicianRepository.existsByName(name))
-           throw new MusicianNotFoundException("Musician with name %s not found".formatted(name));
-
-       Musician musician = musicianRepository.findByName(name);
-
-       List<MusicianProduct> musicianProducts = musicianProductRepository.findByMusician(musician);
-
-       return MusicianProductDTO.builder()
-           .musician(MusicianDTO.builder()
-               .id(musician.getId())
-               .name(musician.getName())
-               .subscribers(musician.getSubscribers())
-               .build())
-           .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO).toList())
-           .build();
-  }
-
-  public MusicianInfoDTO getMusicianInfo(Long musicianId) {
-    log.info("Fetching info for musician with id: {}", musicianId);
-    Musician musician = musicianRepository.findById(musicianId)
-        .orElseThrow(() -> new MusicianNotFoundException(
-            "Musician with id %s not found".formatted(musicianId)));
-
-    return convertToDTO(musician,
-        musicianGenreRepository.findByMusician(musician),
-        musicianTypeOfMusicianRepository.findByMusician(musician),
-        musicianProductRepository.findByMusician(musician));
-  }
-
-  @Transactional
-  public Boolean addProductToMusician(AddProductMusicianDTO addProductMusicianDTO, HttpServletRequest request) {
-    log.info("Adding product with id {} to musician with name {}", addProductMusicianDTO.getProductId(),
-        addProductMusicianDTO.getMusicianName());
-    if (!musicianRepository.existsByName(addProductMusicianDTO.getMusicianName()))
-      throw new MusicianNotFoundException(
-          "Musician with name %s not found".formatted(addProductMusicianDTO.getMusicianName()));
-
-    if (!productRepository.existsById(addProductMusicianDTO.getProductId()))
-      throw new ProductNotFoundException(
-          "Product with id %s not found".formatted(addProductMusicianDTO.getProductId()));
-
-    User user = findUserByRequest(request);
-    log.info("User {} is performing this action", user.getUsername());
-    Musician musician = musicianRepository.findByName(addProductMusicianDTO.getMusicianName());
-    Product product = productRepository.findById(addProductMusicianDTO.getProductId()).get();
-
-    if (musicianProductRepository.existsByMusicianAndProduct(musician, product)) {
-      log.warn("Product {} already exists in musician {}", product.getName(), musician.getName());
-      throw new ProductMusicianAlreadyExists("Product %s already exists in musician %s".formatted(product.getName(),
-          musician.getName()));
+    public MusicianTypeOfMusicianDTO getMusiciansByTypeOfMusician(Long musicianId) {
+        log.info("Fetching types for musician with id: {}", musicianId);
+        Musician musician = musicianRepository.findById(musicianId)
+                .orElseThrow(() -> new MusicianNotFoundException(
+                        "Musician with id %s not found".formatted(musicianId)));
+
+        List<MusicianTypeOfMusician> musicianTypeOfMusicians = musicianTypeOfMusicianRepository
+                .findByMusician(musician);
+
+        return MusicianTypeOfMusicianDTO.builder()
+                .musician(musician)
+                .typeOfMusicians(musicianTypeOfMusicians.stream().map(MusicianTypeOfMusician::getTypeOfMusician)
+                        .toList())
+                .build();
+    }
+
+    public MusicianProductDTO getMusicianProducts(String name) {
+        log.info("Fetching products for musician with name: {}", name);
+        if (!musicianRepository.existsByName(name))
+            throw new MusicianNotFoundException("Musician with name %s not found".formatted(name));
+
+           Musician musician = musicianRepository.findByName(name);
+
+           List<MusicianProduct> musicianProducts = musicianProductRepository.findByMusician(musician);
+
+           return MusicianProductDTO.builder()
+                   .musician(MusicianDTO.builder()
+                           .id(musician.getId())
+                           .name(musician.getName())
+                           .subscribers(musician.getSubscribers())
+                           .build())
+                   .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO)
+                           .toList())
+                   .build();
      }
-     musicianProductRepository
-         .save(MusicianProduct.builder().musicianId(musician.getId()).productId(product.getId()).build());
-     log.info("Product {} successfully added to musician {}", product.getName(), musician.getName());
-     return true;
-   }
-
-   @Transactional
-   public Boolean deleteProductFromMusician(AddProductMusicianDTO addProductMusicianDTO, HttpServletRequest request) {
-     log.info("Deleting product with id {} from musician with name {}", addProductMusicianDTO.getProductId(),
-         addProductMusicianDTO.getMusicianName());
-     if (!musicianRepository.existsByName(addProductMusicianDTO.getMusicianName()))
-       throw new MusicianNotFoundException(
-           "Musician with name %s not found".formatted(addProductMusicianDTO.getMusicianName()));
-
-     if (!productRepository.existsById(addProductMusicianDTO.getProductId()))
-       throw new ProductNotFoundException(
-           "Product with id %s not found".formatted(addProductMusicianDTO.getProductId()));
-     User user = findUserByRequest(request);
-     log.info("User {} is performing this action", user.getUsername());
-     Musician musician = musicianRepository.findByName(addProductMusicianDTO.getMusicianName());
-     Product product = productRepository.findById(addProductMusicianDTO.getProductId()).get();
-
-     if (!musicianProductRepository.existsByMusicianAndProduct(musician, product)) {
-       log.warn("Product {} not found in musician {}", product.getName(), musician.getName());
-       throw new ProductMusicianNotFoundException("Product %s not found in musician %s".formatted(product.getName(),
-           musician.getName()));
+
+     public MusicianInfoDTO getMusicianInfo(Long musicianId) {
+           log.info("Fetching info for musician with id: {}", musicianId);
+           Musician musician = musicianRepository.findById(musicianId)
+                   .orElseThrow(() -> new MusicianNotFoundException(
+                           "Musician with id %s not found".formatted(musicianId)));
+
+           return convertToDTO(musician,
+                   musicianGenreRepository.findByMusician(musician),
+                   musicianTypeOfMusicianRepository.findByMusician(musician),
+                   musicianProductRepository.findByMusician(musician));
+     }
+
+     @Transactional
+     public Boolean addProductToMusician(AddProductMusicianDTO addProductMusicianDTO, HttpServletRequest request) {
+           log.info("Adding product with id {} to musician with name {}", addProductMusicianDTO.getProductId(),
+                 addProductMusicianDTO.getMusicianName());
+         if (!musicianRepository.existsByName(addProductMusicianDTO.getMusicianName()))
+             throw new MusicianNotFoundException(
+                     "Musician with name %s not found".formatted(addProductMusicianDTO.getMusicianName()));
+
+         if (!productRepository.existsById(addProductMusicianDTO.getProductId()))
+             throw new ProductNotFoundException(
+                     "Product with id %s not found".formatted(addProductMusicianDTO.getProductId()));
+
+         User user = findUserByRequest(request);
+         log.info("User {} is performing this action", user.getUsername());
+         Musician musician = musicianRepository.findByName(addProductMusicianDTO.getMusicianName());
+         Product product = productRepository.findById(addProductMusicianDTO.getProductId()).get();
+
+         if (musicianProductRepository.existsByMusicianAndProduct(musician, product)) {
+             log.warn("Product {} already exists in musician {}", product.getName(), musician.getName());
+             throw new ProductMusicianAlreadyExists(
+                     "Product %s already exists in musician %s".formatted(product.getName(),
+                             musician.getName()));
+         }
+         musicianProductRepository
+                 .save(MusicianProduct.builder().musicianId(musician.getId()).productId(product.getId())
+                         .build());
+         log.info("Product {} successfully added to musician {}", product.getName(), musician.getName());
+         return true;
+     }
+
+     @Transactional
+     public Boolean deleteProductFromMusician(AddProductMusicianDTO addProductMusicianDTO,
+             HttpServletRequest request) {
+         log.info("Deleting product with id {} from musician with name {}", addProductMusicianDTO.getProductId(),
+                 addProductMusicianDTO.getMusicianName());
+         if (!musicianRepository.existsByName(addProductMusicianDTO.getMusicianName()))
+             throw new MusicianNotFoundException(
+                     "Musician with name %s not found".formatted(addProductMusicianDTO.getMusicianName()));
+
+         if (!productRepository.existsById(addProductMusicianDTO.getProductId()))
+             throw new ProductNotFoundException(
+                     "Product with id %s not found".formatted(addProductMusicianDTO.getProductId()));
+         User user = findUserByRequest(request);
+         log.info("User {} is performing this action", user.getUsername());
+         Musician musician = musicianRepository.findByName(addProductMusicianDTO.getMusicianName());
+         Product product = productRepository.findById(addProductMusicianDTO.getProductId()).get();
+
+         if (!musicianProductRepository.existsByMusicianAndProduct(musician, product)) {
+             log.warn("Product {} not found in musician {}", product.getName(), musician.getName());
+             throw new ProductMusicianNotFoundException(
+                     "Product %s not found in musician %s".formatted(product.getName(),
+                             musician.getName()));
+         }
+         musicianProductRepository.deleteByMusicianAndProduct(musician, product);
+         log.info("Product {} successfully deleted from musician {}", product.getName(), musician.getName());
+         return true;
+     }
+
+     private User findUserByRequest(HttpServletRequest request) {
```

```
+        String username = jwtUtils.getUserNameFromJwtToken(jwtUtils.parseJwt(request));
+        return userRepository.findByUsername(username)
+                .orElseThrow(() -> new UsernameNotFoundException(
+                        String.format("Username %s not found", username)));
+    }
+
+    private ProductDTO convertToDTO(Product product) {
+        return ProductDTO.builder()
+                .id(product.getId())
+                .name(product.getName())
+                .description(product.getDescription())
+                .rate(product.getRate())
+                .brand(BrandDTO.builder()
+                        .id(product.getBrand().getId())
+                        .name(product.getBrand().getName())
+                        .country(product.getBrand().getCountry())
+                        .website(product.getBrand().getWebsite())
+                        .email(product.getBrand().getEmail())
+                        .build())
+                .guitarForm(product.getGuitarForm())
+                .typeOfProduct(product.getTypeOfProduct())
+                .lads(product.getLads())
+                .avgPrice(product.getAvgPrice())
+                .color(product.getColor())
+                .strings(product.getStrings())
+                .tipMaterial(product.getTipMaterial())
+                .bodyMaterial(product.getBodyMaterial())
+                .pickupConfiguration(product.getPickupConfiguration())
+                .typeComboAmplifier(product.getTypeComboAmplifier())
+                .build();
+    }
+
+    private MusicianInfoDTO convertToDTO(Musician musician, List<MusicianGenre> musicianGenres,
+            List<MusicianTypeOfMusician> musicianTypes, List<MusicianProduct> musicianProducts) {
+        return MusicianInfoDTO.builder()
+                .id(musician.getId())
+                .name(musician.getName())
+                .subscribers(musician.getSubscribers())
+                .genres(musicianGenres.stream().map(MusicianGenre::getGenre).toList())
+                .typesOfMusicians(
+                        musicianTypes.stream().map(MusicianTypeOfMusician::getTypeOfMusician).toList())
+                .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO)
+                        .toList())
+                .build();
+    }
+
+    private MusicianInfoDTO convertToDTOLists(Musician musician, List<Genre> genres,
+            List<TypeOfMusician> typesOfMusician, List<MusicianProduct> musicianProducts) {
+        return MusicianInfoDTO.builder()
+                .id(musician.getId())
+                .name(musician.getName())
+                .subscribers(musician.getSubscribers())
+                .genres(genres)
+                .typesOfMusicians(typesOfMusician)
+                .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO)
+                        .toList())
+                    .build();
     }
-    musicianProductRepository.deleteByMusicianAndProduct(musician, product);
-    log.info("Product {} successfully deleted from musician {}", product.getName(), musician.getName());
-    return true;
-  }
-
-  private User findUserByRequest(HttpServletRequest request) {
-    String username = jwtUtils.getUserNameFromJwtToken(jwtUtils.parseJwt(request));
-    return userRepository.findByUsername(username)
-        .orElseThrow(() -> new UsernameNotFoundException(
-            String.format("Username %s not found", username)));
-  }
-
-  private ProductDTO convertToDTO(Product product) {
-    return ProductDTO.builder()
-        .id(product.getId())
-        .name(product.getName())
-        .description(product.getDescription())
-        .rate(product.getRate())
-        .brand(BrandDTO.builder()
-            .id(product.getBrand().getId())
-            .name(product.getBrand().getName())
-            .country(product.getBrand().getCountry())
-            .website(product.getBrand().getWebsite())
-            .email(product.getBrand().getEmail())
-            .build())
-        .guitarForm(product.getGuitarForm())
-        .typeOfProduct(product.getTypeOfProduct())
-        .lads(product.getLads())
-        .avgPrice(product.getAvgPrice())
-        .color(product.getColor())
-        .strings(product.getStrings())
-        .tipMaterial(product.getTipMaterial())
-        .bodyMaterial(product.getBodyMaterial())
-        .pickupConfiguration(product.getPickupConfiguration())
-        .typeComboAmplifier(product.getTypeComboAmplifier())
-        .build();
-  }
-
-  private MusicianInfoDTO convertToDTO(Musician musician, List<MusicianGenre> musicianGenres,
-      List<MusicianTypeOfMusician> musicianTypes, List<MusicianProduct> musicianProducts) {
-    return MusicianInfoDTO.builder()
-        .id(musician.getId())
-        .name(musician.getName())
-        .subscribers(musician.getSubscribers())
-        .genres(musicianGenres.stream().map(MusicianGenre::getGenre).toList())
-        .typesOfMusicians(musicianTypes.stream().map(MusicianTypeOfMusician::getTypeOfMusician).toList())
-        .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO).toList())
-        .build();
-  }
-
-  private MusicianInfoDTO convertToDTOLists(Musician musician, List<Genre> genres,
-      List<TypeOfMusician> typesOfMusician, List<MusicianProduct> musicianProducts) {
-    return MusicianInfoDTO.builder()
-        .id(musician.getId())
-           .name(musician.getName())
-           .subscribers(musician.getSubscribers())
-           .genres(genres)
-           .typesOfMusicians(typesOfMusician)
-           .products(musicianProducts.stream().map(MusicianProduct::getProduct).map(this::convertToDTO).toList())
-           .build();
-  }
 }
```

Заключение

На этом этапе мы оптимизировали запросы к бэкенду и базе данных. Провели анализ запросов. Для бэкенда был добавлен Redis.