ДОДАТОК Б

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

“ЗАТВЕРДЖЕНО”

Керівник роботи

_____ Ілля АХАЛАДЗЕ

“___” _____ 2024

р.

Музична платформа

Текст програми

КПІ.ІП-1325.045440.05.13

“ПОГОДЖЕНО”

Керівник роботи: Виконавець:

_____ Ілля АХАЛАДЗЕ _____ Олександр ПАЛАМАРЧУК

Київ – 2024

```
package com.aleh1s.backend.audio;

import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/audios")
public class AudioController {

    private final AudioService audioService;

    @GetMapping("/{id}")
    public ResponseEntity<?> streamAudio(@PathVariable("id") String id,
@RequestHeader(name = "range", required = false) String rangeStr) {
        return audioService.streamAudio(id, AudioRange.parse(rangeStr));
    }
}

package com.aleh1s.backend.audio;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;
import lombok.ToString;

import java.util.UUID;
```

```
@Entity
@Getter
@Setter
@ToString
@Table(name = "audio")
@NoArgsConstructor
public class AudioEntity {

    @Id
    @Column(name = "id", nullable = false)
    private String id;

    @Column(name = "extension", nullable = false)
    private String extension;

    @Column(name = "length", nullable = false)
    private long length;

    @Lob
    @Column(name = "data", nullable = false)
    private byte[] data;

    public AudioEntity(String extension, long length, byte[] data) {
        this.extension = extension;
        this.length = length;
        this.data = data;
    }
```

```
    @PrePersist
    private void prePersist() {
        this.id = UUID.randomUUID().toString();
    }
}

package com.aleh1s.backend.audio;

import lombok.Getter;
import lombok.Setter;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import static java.util.Objects.nonNull;

@Getter
@Setter
public class AudioRange {

    private static final Pattern RANGE_PATTERN =
Pattern.compile("^bytes=(\\d+)-(\\d+)?/?(\\d+)?$");

    private long start;
    private Long end;
    private Long length;

    private AudioRange(long start, Long end, Long length) {
        this.start = start;
```

```
        this.end = end;
        this.length = length;
    }

    public static AudioRange parse(String range) {
        if (range != null) {
            Matcher matcher = RANGE_PATTERN.matcher(range);

            if (matcher.find()) {
                String startStr = matcher.group(1);
                String endStr = matcher.group(2);
                String lengthStr = matcher.group(3);

                return new AudioRange(
                        nonNull(startStr) ? Long.parseLong(startStr) : 0,
                        nonNull(endStr) ? Long.parseLong(endStr) : null,
                        nonNull(lengthStr) ? Long.parseLong(lengthStr) : null
                );
            }
        }

        return null;
    }
}
```

```
package com.aleh1s.backend.audio;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
@Repository
public interface AudioRepository extends JpaRepository<AudioEntity, String>
{
}

package com.aleh1s.backend.audio;

import com.aleh1s.backend.exception.InvalidResourceException;
import com.aleh1s.backend.exception.ResourceNotFoundException;
import com.aleh1s.backend.util.ArrayUtils;
import com.aleh1s.backend.util.FileUtils;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.util.Set;

import static java.util.Objects.isNull;

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class AudioService {
```

```
    @Value("#{'${media.audio.supported-extensions}'.split(',')}")
    private Set<String> supportedAudioExtensions;

    private final AudioRepository audioRepository;

    @Transactional
    public String saveAudio(MultipartFile audio) throws IOException {
        String originalFilename = audio.getOriginalFilename();

        String fileExtension = FileUtils.getFileExtension(originalFilename)
                .orElseThrow(() -> new InvalidResourceException("File extension is
not supported. It should be one of: %s".formatted(supportedAudioExtensions)));
        if (!supportedAudioExtensions.contains(fileExtension)) {
            throw new InvalidResourceException("File extension is not supported. It
should be one of: %s".formatted(supportedAudioExtensions));
        }

        AudioEntity newAudio = new AudioEntity(fileExtension, audio.getSize(),
audio.getBytes());
        audioRepository.save(newAudio);

        return newAudio.getId();
    }

    public ResponseEntity<?> streamAudio(String id, AudioRange audioRange)
{
        AudioEntity audio = getAudioById(id);
```

```
        byte[] data = audio.getData();
        if (isNull(audioRange)) {
            return ResponseEntity.status(HttpStatus.OK)
                    .header("Content-Type", "audio/mpeg")
                    .header("Content-Length", String.valueOf(data.length))
                    .body(data);
        }

        long rangeStart = audioRange.getStart();
        Long rangeEnd = audioRange.getEnd();

        if (isNull(rangeEnd) || data.length < rangeEnd) {
            rangeEnd = (long) data.length - 1;
        }

        data = ArrayUtils.readByteRange(data, rangeStart, rangeEnd);
        return ResponseEntity.status(HttpStatus.PARTIAL_CONTENT)
                .header("Content-Type", "audio/mpeg")
                .header("Accept-Ranges", "bytes")
                .header("Content-Length", String.valueOf(data.length))
                .header("Content-Range", "bytes %d-%d/%d".formatted(rangeStart,
rangeEnd, audio.getLength()))
                .body(data);
    }

    public AudioEntity getAudioById(String id) {
        return audioRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Media with id
%s does not exist".formatted(id)));
```

```
    }

    public void deleteAudioById(String id) {
        audioRepository.deleteById(id);
    }
}

package com.aleh1s.backend.audio;

public record UploadAudioResponse(
        String id
) {
}

package com.aleh1s.backend.auth;

import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/auth")
public class AuthController {
```

```
    private final AuthService authService;

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody LoginRequest
loginRequest) {
        LoginResponse loginResponse = authService.login(loginRequest);
        return ResponseEntity.ok()
                .header(HttpHeaders.AUTHORIZATION, loginResponse.jwt())
                .body(loginResponse);
    }
}

package com.aleh1s.backend.auth;

import com.aleh1s.backend.dto.DtoMapper;
import com.aleh1s.backend.jwt.JwtUtil;
import com.aleh1s.backend.user.UserDto;
import com.aleh1s.backend.user.UserEntity;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthentication
Token;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class AuthService {
```

```
    private final AuthenticationManager authenticationManager;
    private final DtoMapper dtoMapper;
    private final JwtUtil jwtUtil;

    public LoginResponse login(LoginRequest loginRequest) {
        Authentication authenticate = authenticationManager.authenticate(new
UsernamePasswordAuthenticationToken(
                loginRequest.username(),
                loginRequest.password()
        ));

        UserEntity customer = (UserEntity) authenticate.getPrincipal();
        UserDto customerDto = dtoMapper.toUserDto(customer);

        String jwt = jwtUtil.issueToken(
                customerDto.username(),
                customerDto.roles()
        );

        return new LoginResponse(jwt, customerDto.username());
    }
}

package com.aleh1s.backend.auth;

public record LoginRequest(
        String username,
        String password
```

```
) {
}

package com.aleh1s.backend.auth;

public record LoginResponse(
        String jwt,
        String username
) {
}

package com.aleh1s.backend.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class Config {

    @Bean
    public RestTemplate restTemplate() {
        return new RestTemplate();
    }
}

package com.aleh1s.backend.dto;

import com.aleh1s.backend.playlist.*;
```

```
import com.aleh1s.backend.registration.RegistrationRequest;
import com.aleh1s.backend.song.*;
import com.aleh1s.backend.user.UserDto;
import com.aleh1s.backend.user.UserEntity;
import com.aleh1s.backend.user.UserRole;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.stereotype.Component;

@Component
public class DtoMapper {

    public UserDto toUserDto(UserEntity userEntity) {
        return new UserDto(
                userEntity.getId(),
                userEntity.getName(),
                userEntity.getEmail(),
                userEntity.getAuthProvider(),
                userEntity.getAuthorities().stream()
                        .map(GrantedAuthority::getAuthority)
                        .toList(),
                userEntity.getUsername(),
                userEntity.isBlocked()
        );
    }

    public SongEntity toSong(CreateSongRequest createSongRequest) {
        return new SongEntity(
                createSongRequest.name(),
                createSongRequest.artist(),
```

```
            MusicCategory.getCategoryById(createSongRequest.categoryId()),
            createSongRequest.tags(),
            createSongRequest.text()
        );
    }

    public SongMinView toSongMinView(SongEntity songEntity) {
        return new SongMinView(
            songEntity.getId(),
            songEntity.getName(),
            songEntity.getArtist(),
            toMusicCategoryView(songEntity.getCategory()),
            songEntity.getPreviewId(),
            songEntity.getDurationInSeconds()
        );
    }

    public MusicCategoryView toMusicCategoryView(MusicCategory
musicCategory) {
        return new MusicCategoryView(
            musicCategory.getId(),
            musicCategory.getCategoryName()
        );
    }

    public SongFullView toSongFullView(SongEntity songEntity) {
        return new SongFullView(
            songEntity.getId(),
            songEntity.getName(),
```

```
            songEntity.getArtist(),
            toMusicCategoryView(songEntity.getCategory()),
            songEntity.getTags(),
            songEntity.getText(),
            songEntity.getPreviewId(),
            songEntity.getAudioId(),
            songEntity.getDurationInSeconds(),
            songEntity.isLiked()
        );
    }

    public PlaylistEntity toPlaylist(CreatePlaylistRequest createPlaylistRequest)
{
        return new PlaylistEntity(createPlaylistRequest.name());
    }

    public PlaylistFullView toPlaylistFullView(PlaylistEntity playlist) {
        return new PlaylistFullView(
            playlist.getId(),
            playlist.getName(),
            playlist.getTotalSongs(),
            playlist.getTotalDurationInSeconds(),
            playlist.isLikedSongsPlaylist(),
            playlist.getPreviewId()
        );
    }

    public PlaylistMinView toPlaylistMinView(PlaylistEntity playlist) {
        return new PlaylistMinView(
```

```
            playlist.getId(),
            playlist.getName(),
            playlist.getTotalSongs(),
            playlist.isLikedSongsPlaylist(),
            playlist.getPreviewId()
        );
    }

    public PlaylistRelatedSongView toPlaylistRelatedSongView(PlaylistEntity
playlist) {
        return new PlaylistRelatedSongView(
            playlist.getId(),
            playlist.getName(),
            playlist.getPreviewId(),
            playlist.getTotalSongs(),
            playlist.isLikedSongsPlaylist(),
            playlist.isContainRelatedSong()
        );
    }

    public UserEntity toUser(RegistrationRequest request) {
        return new UserEntity(
            request.name(),
            request.email(),
            request.password(),
            UserRole.USER
        );
    }
}
```

```
package com.aleh1s.backend.exception;

import jakarta.servlet.http.HttpServletRequest;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.authentication.InsufficientAuthenticationExceptio
n;
import org.springframework.validation.FieldError;
import org.springframework.web.HttpRequestMethodNotSupportedException;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

import java.time.LocalDateTime;
import java.util.stream.Collectors;

import static org.springframework.http.HttpStatus.*;

@ControllerAdvice
public class DefaultExceptionHandler {

    private static final Logger log =
LoggerFactory.getLogger(DefaultExceptionHandler.class);

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ApiError>
handleException(MethodArgumentNotValidException e,
```

```
                                        HttpServletRequest request) {
        String message = e.getBindingResult().getAllErrors().stream()
                .map((error) -> "%s - %s".formatted(
                        ((FieldError) error).getField(),
                        error.getDefaultMessage()
                )).collect(Collectors.joining(";"));
        ApiError apiError = new ApiError(
                request.getRequestURI(),
                message,
                BAD_REQUEST.value(),
                LocalDateTime.now()
        );
        return ResponseEntity.status(BAD_REQUEST).body(apiError);
    }

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ApiError>
handleException(ResourceNotFoundException e,
                                        HttpServletRequest request) {
        ApiError apiError = new ApiError(
                request.getRequestURI(),
                e.getMessage(),
                NOT_FOUND.value(),
                LocalDateTime.now()
        );
        return ResponseEntity.status(NOT_FOUND).body(apiError);
    }

    @ExceptionHandler(InvalidResourceException.class)
```

```
    public ResponseEntity<ApiError>
handleException(InvalidResourceException e,
                                    HttpServletRequest request) {
        ApiError apiError = new ApiError(
                request.getRequestURI(),
                e.getMessage(),
                BAD_REQUEST.value(),
                LocalDateTime.now()
        );
        return ResponseEntity.status(BAD_REQUEST).body(apiError);
    }

    @ExceptionHandler(DuplicateResourceException.class)
    public ResponseEntity<ApiError>
handleException(DuplicateResourceException e,
                                    HttpServletRequest request) {
        ApiError apiError = new ApiError(
                request.getRequestURI(),
                e.getMessage(),
                CONFLICT.value(),
                LocalDateTime.now()
        );
        return ResponseEntity.status(CONFLICT).body(apiError);
    }

    @ExceptionHandler(ForbiddenException.class)
    public ResponseEntity<ApiError> handleException(ForbiddenException e,
                                    HttpServletRequest request) {
        ApiError apiError = new ApiError(
```

```
                request.getRequestURI(),
                e.getMessage(),
                FORBIDDEN.value(),
                LocalDateTime.now()
        );
        return ResponseEntity.status(FORBIDDEN).body(apiError);
    }

    @ExceptionHandler(HttpRequestMethodNotSupportedException.class)
    public ResponseEntity<ApiError>
handleException(HttpRequestMethodNotSupportedException e,
                                        HttpServletRequest request) {
        ApiError apiError = new ApiError(
                request.getRequestURI(),
                e.getMessage(),
                METHOD_NOT_ALLOWED.value(),
                LocalDateTime.now()
        );
        return
ResponseEntity.status(METHOD_NOT_ALLOWED).body(apiError);
    }

    @ExceptionHandler(InsufficientAuthenticationException.class)
    public ResponseEntity<ApiError>
handleException(InsufficientAuthenticationException e,
                                        HttpServletRequest request) {
        ApiError apiError = new ApiError(
                request.getRequestURI(),
                e.getMessage(),
```

```
                FORBIDDEN.value(),
                LocalDateTime.now()
        );
        return ResponseEntity.status(FORBIDDEN).body(apiError);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ApiError> handleException(Exception e,
                                                    HttpServletRequest request) {
        log.error(e.getMessage(), e);
        ApiError apiError = new ApiError(
                request.getRequestURI(),
                e.getMessage(),
                INTERNAL_SERVER_ERROR.value(),
                LocalDateTime.now()
        );
        return
ResponseEntity.status(INTERNAL_SERVER_ERROR).body(apiError);
    }
}

package com.aleh1s.backend.image;

import com.aleh1s.backend.exception.InvalidResourceException;
import com.aleh1s.backend.exception.ResourceNotFoundException;
import com.aleh1s.backend.util.FileUtils;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
```

```
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.Set;

import static java.util.Objects.isNull;

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class ImageService {

    @Value("#{'${media.image.supported-extensions}'.split(',')}")
    private Set<String> supportedImageExtensions;

    private final ImageRepository imageRepository;

    @Transactional
    public String saveImage(MultipartFile image) throws IOException {
        String contentType = image.getContentType();
        if (isNull(contentType) || !contentType.startsWith("image")) {
            throw new InvalidResourceException("Content type is not present or is
not image");
        }

        String originalFilename = image.getOriginalFilename();
```

```
        String fileExtension = FileUtils.getFileExtension(originalFilename)
                .orElseThrow(() -> new InvalidResourceException("File name have
no extension"));

        if (!supportedImageExtensions.contains(fileExtension)) {
            throw new InvalidResourceException("File extension is not supported. It
should be one of: %s".formatted(supportedImageExtensions));
        }

        BufferedImage bufferedImage = ImageIO.read(image.getInputStream());

        int height = bufferedImage.getHeight();
        int width = bufferedImage.getWidth();

        if (height != 500 || width != 500) {
            throw new InvalidResourceException("Image should have 500x500
resolution");
        }

        ImageEntity newImage = new ImageEntity(fileExtension,
image.getBytes());
        imageRepository.save(newImage);

        return newImage.getId();
    }

    public ImageEntity getImageById(String id) {
        return imageRepository.findById(id)
```

```
            .orElseThrow(() -> new ResourceNotFoundException("Media with id
%s does not exist".formatted(id)));
    }

    public void deleteImageById(String id) {
        imageRepository.deleteById(id);
    }
}

package com.aleh1s.backend.jwt;

import com.aleh1s.backend.user.UserDetailsServiceImpl;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;
import
org.springframework.security.authentication.UsernamePasswordAuthentication
Token;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSou
rce;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;
```

```
import java.io.IOException;

import static java.util.Objects.isNull;
import static java.util.Objects.nonNull;

@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private static final String TOKEN_PREFIX = "Bearer ";

    private final JwtUtil jwtUtil;
    private final UserDetailsServiceImpl userDetailsServiceImpl;

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request,
                                    @NonNull HttpServletResponse response,
                                    @NonNull FilterChain filterChain) throws
ServletException, IOException {
        String authorizationHeader = request.getHeader("Authorization");

        if (nonNull(authorizationHeader) &&
authorizationHeader.startsWith(TOKEN_PREFIX)) {
            String jwt = authorizationHeader.substring(TOKEN_PREFIX.length());
            String email = jwtUtil.extractSubject(jwt);

            if (nonNull(email) && jwtUtil.isTokenNotExpired(jwt) &&
isNull(SecurityContextHolder.getContext().getAuthentication())) {
```

```
            UserDetails userDetails =
userDetailsServiceImpl.loadUserByUsername(email);

            UsernamePasswordAuthenticationToken authenticationToken =
                    new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());

            authenticationToken.setDetails(
                    new WebAuthenticationDetailsSource().buildDetails(request)
            );


SecurityContextHolder.getContext().setAuthentication(authenticationToken);
        }
    }

    filterChain.doFilter(request, response);
  }
}

package com.aleh1s.backend.oauth2;

import
com.aleh1s.backend.exception.OAuth2AuthenticationProcessingException;
import com.aleh1s.backend.jwt.JwtUtil;
import com.aleh1s.backend.util.CookieUtils;
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
```

```
import lombok.RequiredArgsConstructor;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import
org.springframework.security.web.authentication.SimpleUrlAuthenticationSucc
essHandler;
import org.springframework.stereotype.Component;
import org.springframework.web.util.UriComponentsBuilder;

import java.io.IOException;
import java.net.URI;
import java.util.Optional;
import java.util.stream.Stream;

@Component
@RequiredArgsConstructor
public class OAuth2AuthenticationSuccessHandler extends
SimpleUrlAuthenticationSuccessHandler {

    private static final Logger logger =
LogManager.getLogger(OAuth2AuthenticationSuccessHandler.class);

    private final JwtUtil jwtUtil;
    private final HttpCookieOAuth2AuthorizationRequestRepository
httpCookieOAuth2AuthorizationRequestRepository;

    @Value("${app.oauth2.authorized-redirect-uri}")
```

```
    private String authorizedRedirectUris;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws
IOException {
        String targetUrl = determineTargetUrl(request, response, authentication);

        if (response.isCommitted()) {
            logger.debug("Response has already been committed. Unable to redirect
to " + targetUrl);
            return;
        }

        clearAuthenticationAttributes(request, response);
        getRedirectStrategy().sendRedirect(request, response, targetUrl);
    }

    protected String determineTargetUrl(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) {
        Optional<String> redirectUri = CookieUtils.getCookie(request,
HttpCookieOAuth2AuthorizationRequestRepository.REDIRECT_URI_PARA
M_COOKIE_NAME)
                .map(Cookie::getValue);

        if (redirectUri.isPresent() && !isAuthorizedRedirectUri(redirectUri.get()))
{
            throw new OAuth2AuthenticationProcessingException("Sorry! We've
got an Unauthorized Redirect URI and can't proceed with the authentication");
```

```
        }

        String targetUrl = redirectUri.orElse(getDefaultTargetUrl());

        String token = jwtUtil.issueToken(authentication);

        return UriComponentsBuilder.fromUriString(targetUrl)
                .queryParam("token", token)
                .build().toUriString();
    }

    protected void clearAuthenticationAttributes(HttpServletRequest request,
HttpServletResponse response) {
        super.clearAuthenticationAttributes(request);

httpCookieOAuth2AuthorizationRequestRepository.removeAuthorizationReque
stCookies(request, response);
    }

    private boolean isAuthorizedRedirectUri(String uri) {
        URI clientRedirectUri = URI.create(uri);

        return Stream.of(authorizedRedirectUris)
                .anyMatch(authorizedRedirectUri -> {
                    // Only validate host and port. Let the clients use different paths if
they want to
                    URI authorizedURI = URI.create(authorizedRedirectUri);
                    return
authorizedURI.getHost().equalsIgnoreCase(clientRedirectUri.getHost())
```

```
                    && authorizedURI.getPort() == clientRedirectUri.getPort();
            });
    }
}
```

```
package com.aleh1s.backend.playlist;

import com.aleh1s.backend.dto.DtoMapper;
import com.aleh1s.backend.song.SongMinView;
import com.aleh1s.backend.song.SongService;
import com.aleh1s.backend.util.PaginationUtils;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.util.List;
import java.util.Set;
import java.util.function.Function;

import static java.util.Objects.isNull;

@RestController
@RequiredArgsConstructor
@RequestMapping("/api/v1/playlists")
```

```
public class PlaylistController {

    private final PlaylistService playlistService;
    private final DtoMapper dtoMapper;
    private final SongService songService;

    @GetMapping
    public ResponseEntity<?> getPlaylists(
            @RequestParam(value = "related_song", required = false) String
relatedSongId
    ) {
        Set<PlaylistEntity> playlists = playlistService.getPlaylists(relatedSongId);
        Function<PlaylistEntity, ?> mapper = isNull(relatedSongId)
                ? dtoMapper::toPlaylistMinView
                : dtoMapper::toPlaylistRelatedSongView;
        List<?> body = playlists.stream()
                .map(mapper)
                .toList();
        return ResponseEntity.ok(body);
    }

    @PostMapping
    public ResponseEntity<?> createPlaylist(
            @Valid @RequestPart("playlist") CreatePlaylistRequest
createPlaylistRequest,
            @RequestPart("preview") MultipartFile preview
    ) throws IOException {
        PlaylistEntity playlist = dtoMapper.toPlaylist(createPlaylistRequest);
        playlistService.savePlaylist(playlist, preview);
```

```
        return ResponseEntity.status(HttpStatus.CREATED).build();
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<?> deletePlaylist(@PathVariable("id") Long id) {
        playlistService.deletePlaylistById(id);
        return ResponseEntity.ok().build();
    }

    @PutMapping("/{id}")
    public ResponseEntity<?> updatePlaylist(
            @PathVariable("id") Long id,
            @Valid @RequestPart("playlist") CreatePlaylistRequest request,
            @RequestPart(name = "preview", required = false) MultipartFile
preview
    ) throws IOException {
        playlistService.updatePlaylist(id, dtoMapper.toPlaylist(request), preview);
        return ResponseEntity.ok().build();
    }

    @GetMapping("/{id}")
    public ResponseEntity<?> getPlaylistById(@PathVariable("id") Long id)
throws IOException {
        PlaylistEntity playlist =
playlistService.getPlaylistByIdFetchTotalDurationInSeconds(id);
        return ResponseEntity.ok(dtoMapper.toPlaylistFullView(playlist));
    }

    @GetMapping("/{id}/songs")
```

```
    public ResponseEntity<?> getSongsByPlaylistId(
            @PathVariable("id") Long id,
            @RequestParam(value = "limit", defaultValue = "10") int limit,
            @RequestParam(value = "page", defaultValue = "0") int page
    ) throws IOException {
        Page<SongMinView> songs = songService.getSongsByPlaylistId(
                id, PaginationUtils.getPageRequest(page, limit)
        ).map(dtoMapper::toSongMinView);
        return ResponseEntity.ok(songs);
    }

    @PostMapping("/{playlist-id}/songs/{song-id}")
    public ResponseEntity<?> addSongToPlaylist(
            @PathVariable("playlist-id") Long playlistId,
            @PathVariable("song-id") String songId
    ) {
        playlistService.addSongToPlaylist(playlistId, songId);
        return ResponseEntity.ok().build();
    }

    @DeleteMapping("/{playlist-id}/songs/{song-id}")
    public ResponseEntity<?> deleteSongFromPlaylist(
            @PathVariable("playlist-id") Long playlistId,
            @PathVariable("song-id") String songId
    ) {
        playlistService.deleteSongFromPlaylist(playlistId, songId);
        return ResponseEntity.ok().build();
    }
}
```