BACK END

SECURE CONNECTION

proxy

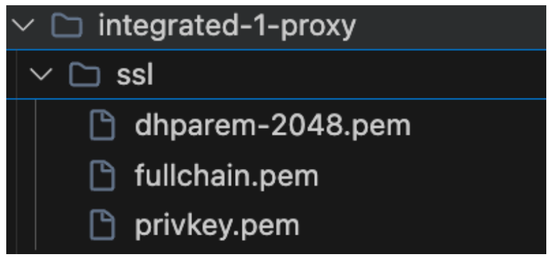
```
server {
    listen 80;
    server_name ip23sj1.sit.kmutt.ac.th;
    client_max_body_size 210M;
    # Redirect HTTP to HTTPS
    return 301 https://$host$request_uri;
}
  ⌘I for Command, ⌘L for Cascade
server {
    listen 443 ssl;
    server_name ip23sj1.sit.kmutt.ac.th;
    client_max_body_size 210M;
    # SSL certificate configuration
    ssl_certificate /ssl/fullchain.pem;
    ssl_certificate_key /ssl/privkey.pem;
    # SSL protocols
    ssl_protocols TLSv1.3;
    ssl_prefer_server_ciphers on;

    # HTTP Strict Transport Security (HSTS)
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

    location / {
        proxy_pass http://frontend/;
    }

    location /api/ {
        proxy_pass http://backend:8080/;
    }

    location /sj1/ {
        proxy_pass http://frontend/;
    }

    location /sj1/api/ {
        proxy_pass http://backend:8080/;
    }
```

จะมีการ implement TLSv1.3 รวมถึงการที่เราดึงข้อมูลของ key ต่างๆมาใช้ผ่านการที่เราทำการ volumes
ไว้

compose.yaml

```yaml
proxy:
  build: ./integrated-1-proxy
  container_name: proxy
  restart: unless-stopped
  ports:
    - 80:80
    - 443:443
  networks:
    - integrated_net
  depends_on:
    - frontend
    - backend
    - database
  volumes:
    - /home/sysadmin/deploy/integrated-1-proxy/ssl:/ssl
```

ทำการ volumes ตัว ssl ต่างๆเข้าไปใน container ของ proxy เพื่อ implement



key ต่างๆที่เตรียมไว้เพื่อเตรียม implement https

PASSWORD/TOKEN MANAGEMENT

When login we use authenticationManager to validate password

```java
@PostMapping("/login")
public ResponseEntity<Object> login(@RequestBody @Valid AuthRequest request) {
    Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(request.getUserName(), request.getPassword())
    );
    User user = jwtUserDetailsService.loadUserByUsername(request.getUserName());
    if (user == null) {
        throw new UsernameNotFoundException("User not found");
    }
    String access_token = jwtService.generateAccessToken(user);
    String refresh_token = jwtService.generateRefreshToken(user);

    AuthResponse authResponse = new AuthResponse(access_token, refresh_token);
    return ResponseEntity.ok(authResponse);
}
```

```java
public enum AuthType {
    LOCAL, AZURE
}
```

```java
@PostMapping("/token")
public ResponseEntity<AuthResponse> refreshToken(
        @RequestHeader("Authorization") String requestTokenHeader,
        @RequestHeader("Auth-Type") String authType
) throws UnauthenticatedException {
    String refreshToken = requestTokenHeader.substring(7);
    if (authType != null && authType.equals("AZURE")) {
        AuthResponse response = azureService.refreshAccessToken(refreshToken);
        if (response == null) {
            return ResponseEntity.status(401).build();
        }
        return ResponseEntity.ok(response);
    }

    String oid = jwtService.getClaimValueFromToken(refreshToken, jwtService.getRefreshKey(), "oid");
    if (oid != null && jwtService.validateRefreshToken(refreshToken, oid)) {
        User user = jwtUserDetailsService.loadUserByOid(oid);
        String newToken = jwtService.generateAccessToken(user);

        AuthResponse authResponse = new AuthResponse(newToken, null);
        return ResponseEntity.ok(authResponse);
    }

    return ResponseEntity.status(401).build();
}
}
```

```java
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse respons
    String authHeader = request.getHeader("Authorization");
    String authType = request.getHeader("Auth-Type");

    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }

    String token = authHeader.substring(7);
    if ("AZURE".equals(authType)) {
        handleAzureAuth(request, response, filterChain, token);
    } else {
        handleLocalAuth(request, response, filterChain, token);
    }
}
```

```java
private void handleLocalAuth(HttpServletRequest request, HttpServletResponse response, FilterChain
    String oid;

    oid = jwtService.getClaimValueFromToken(token, jwtService.getAccessKey(), "oid");


    if (oid != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        User user = userDetailsService.loadUserByOid(oid);
        user.setAuthType(AuthType.LOCAL);
        user.setAccessToken(token);

        if (jwtService.validateAccessToken(token, oid)) {
            setAuthentication(request, user);
        } else {
            return;
        }
    }

    filterChain.doFilter(request, response);
}
```

CREATE/VALIDATE TOKEN(ITBKK)

```
public enum AuthType {
    LOCAL, AZURE
}
```

In ITBKK we have 2 secrets in .env for generate access_token and refresh_token

```
#SECRET FOR TOKEN
SECRET=N7KgseMPtJ26AEved0ahUKEwj4563eioyFAxUyUGwGHbTODx0Q4dUDCBA
SECRET_REFRESH=N7KgseMPtJ26AEved0ahUKEwj4563eioyFAxUyUGwGHbTODx0Q4dEUNDF
```

```
private String createToken(User user, long expiration, Key key) {
    Map<String, Object> claims = Map.of(
            "oid", user.getOid(),
            "name", user.getName(),
            "email", user.getEmail(),
            "role", user.getRole()
    );
    return doGenerateToken(claims, expiration, key);
}

private String doGenerateToken(Map<String, Object> claims, long expiration,Key key) {
    return Jwts.builder().setHeaderParam("typ", "JWT")
            .setClaims(claims)
            .setIssuer(ISS)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + expiration))
            .signWith(key).compact();
}
```

In JwtAuthFilter

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse respons
    String authHeader = request.getHeader("Authorization");
    String authType = request.getHeader("Auth-Type");

    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }

    String token = authHeader.substring(7);
    if ("AZURE".equals(authType)) {
        handleAzureAuth(request, response, filterChain, token);
    } else {
        handleLocalAuth(request, response, filterChain, token);
    }
}
```

```
private void handleLocalAuth(HttpServletRequest request, HttpServletResponse response, FilterChain
    String oid;

    oid = jwtService.getClaimValueFromToken(token, jwtService.getAccessKey(), "oid");


    if (oid != null && SecurityContextHolder.getContext().getAuthentication() == null) {
        User user = userDetailsService.loadUserByOid(oid);
        user.setAuthType(AuthType.LOCAL);
        user.setAccessToken(token);

        if (jwtService.validateAccessToken(token, oid)) {
            setAuthentication(request, user);
        } else {
            return;
        }
    }

    filterChain.doFilter(request, response);
}
```

```
public boolean validateAccessToken(String token, String oid) {
    return oid.equals(getClaimValueFromToken(token, getAccessKey(), "oid")) && !isTokenExpired(token, getAccessKey());
}

public boolean validateRefreshToken(String token, String oid) {
    return oid.equals(getClaimValueFromToken(token, getRefreshKey(), "oid")) && !isTokenExpired(token, getRefreshKey());
}
```

CREATE/VALIDATE TOKEN(MSIP)

```
public enum AuthType {
    LOCAL, AZURE
}
```

We use MSAL in Front-end to authenticate and generate access_token and refresh_token from MSIP

```
const tokenKeys = JSON.parse(localStorage.getItem(`msal.token.keys.${import.meta.env.VITE_CLIENT_ID}`))
const accessKey = JSON.parse(localStorage.getItem(`${tokenKeys.accessToken[0]}`))
const refreshKey = JSON.parse(localStorage.getItem(`${tokenKeys.refreshToken[0]}`))
const idKey = JSON.parse(localStorage.getItem(`${tokenKeys.idToken[0]}`))
```

In JwtAuthFilter

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse respons
    String authHeader = request.getHeader("Authorization");
    String authType = request.getHeader("Auth-Type");

    if (authHeader == null || !authHeader.startsWith("Bearer ")) {
        filterChain.doFilter(request, response);
        return;
    }

    String token = authHeader.substring(7);
    if ("AZURE".equals(authType)) {
        handleAzureAuth(request, response, filterChain, token);
    } else {
        handleLocalAuth(request, response, filterChain, token);
    }
}
```

```
public User fetchUserDetails(String accessToken) {
    String endpoint = "https://graph.microsoft.com/v1.0/me";
    try {
        String response = sendGetRequest(endpoint, accessToken);

        String oid = extractJsonValue(response, "id");
        String email = extractJsonValue(response, "mail");
        String name = extractJsonValue(response, "displayName");

        if (isNullOrEmpty(oid) || isNullOrEmpty(email) || isNullOrEmpty(name)) {
            throw new UnauthenticatedException("Failed to retrieve user information from Azure.");
        }

        String username = generateUsername(name);

        return buildUser(oid, name, email, username);
    } catch (IOException e) {
        throw new UnauthenticatedException("Failed to retrieve user information from Azure.");
    }
}
```

```
private void handleAzureAuth(HttpServletRequest request, HttpServle
    User user = azureService.fetchUserDetails(token);

    if (user == null) {
        return;
    }

    user.setAuthType(AuthType.AZURE);
    user.setAccessToken(token);
    setAuthentication(request, user);

    azureService.cacheUserDetails(user);

    filterChain.doFilter(request, response);
}
```

```
public void cacheUserDetails(User user) {
    userCacheService.save(new UserCache(user.getOid(), user.getName(), user.getUsername(), user.getEmail()));

}
```

CREATE/VALIDATE TOKEN(MSIP)

```java
@PostMapping("/token")
public ResponseEntity<AuthResponse> refreshToken(
        @RequestHeader("Authorization") String requestTokenHeader,
        @RequestHeader("Auth-Type") String authType
) throws UnauthenticatedException {
    String refreshToken = requestTokenHeader.substring(7);
    if (authType != null && authType.equals("AZURE")) {
        AuthResponse response = azureService.refreshAccessToken(refreshToken);
        if (response == null) {
            return ResponseEntity.status(401).build();
        }
        return ResponseEntity.ok(response);
    }

    String oid = jwtService.getClaimValueFromToken(refreshToken, jwtService.getRefreshKey(), "oid");
    if (oid != null && jwtService.validateRefreshToken(refreshToken, oid)) {
        User user = jwtUserDetailsService.loadUserByOid(oid);
        String newToken = jwtService.generateAccessToken(user);

        AuthResponse authResponse = new AuthResponse(newToken, null);
        return ResponseEntity.ok(authResponse);
    }

    return ResponseEntity.status(401).build();
}

}
```

```java
public AuthResponse refreshAccessToken(String refreshToken) {
    try {
        String tokenEndpoint = buildTokenEndpoint();

        String requestBody = buildRefreshTokenRequestBody(refreshToken);

        String response = sendPostRequest(tokenEndpoint, requestBody);

        String accessToken = extractJsonValue(response, "access_token");
        refreshToken = extractJsonValue(response, "refresh_token");

        return buildAuthResponse(accessToken, refreshToken, AuthType.AZURE);
    } catch (IOException e) {
        throw new UnauthenticatedException("Failed to refresh access token.");
    }
}
```

AUTHORIZATION FLOW

```
private Board permissionCheck(String authorizationHeader, String type, String bid, String method, Boolean isCollabCanDoOperation) {
    String userId = null;
    if (authorizationHeader != null) userId = getOidFromHeader(authorizationHeader, type);
    Board board = boardService.getBoard(bid);
    if ((!Objects.equals(method, "get") && board.getVisibility().equals(Visibility.PUBLIC) && userId != null) || board.getVisibility().equals(Visibility.PRIVATE))
        oidCheck(board, userId, method, board.getVisibility(), isCollabCanDoOperation);
    }
    return board;
}
```

```
private AccessRight oidCheck(Board board, String userOid, String method, Visibility visibility, Boolean isCollabCanDoOperation) {
    boolean isOwner = Objects.equals(board.getOid(), userOid);
    Collab collab = isOwner ? null : collabService.getCollabOfBoard(board.getId(), userOid, false);

    boolean isCollab = isOwner || (collab != null && collab.getStatus() == CollabStatus.JOINED);
    boolean isWriteAccess = isOwner || (collab != null && collab.getAccessRight() == AccessRight.WRITE);
    boolean isCanDoOp = Objects.equals(method, "get") || isCollabCanDoOperation;
    if (!Objects.equals(method, "get") && !isWriteAccess && !Objects.equals(board.getId(), "kanbanbase")) {
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, "You don't have permission on this board");
    }
    if (!isOwner && !isCanDoOp && !Objects.equals(board.getId(), "kanbanbase")) {
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, "You don't have permission on this board");
    }

    if (visibility == Visibility.PRIVATE) {
        if (!isCollab && !Objects.equals(board.getId(), "kanbanbase")) {
            throw new ResponseStatusException(HttpStatus.FORBIDDEN, "You don't have permission on this board");
        }
    }

    return isOwner || (collab != null && collab.getStatus() == CollabStatus.JOINED && collab.getAccessRight() == AccessRight.WRITE) ? AccessRight.WRITE : AccessRight.READ;
}
```

```
@GetMapping("/{id}/access")
public ResponseEntity<Object> getBoardAccess(@RequestHeader(value = "Authorization", required = false) String authorizationHeader, @RequestHeader("Auth-Type") String authType, @PathVa
    Board board = permissionCheck(authorizationHeader, authType, id, "get", true);
    AccessRightDTO output = new AccessRightDTO();
    output.setAccessRight(String.valueOf(oidCheck(board, getOidFromHeader(authorizationHeader, authType), "get", board.getVisibility(), true)));
    return ResponseEntity.ok(output);
}
```

PENDING COLLAB/DECLINE INVITE

```
CREATE TABLE collabs (
    boardId VARCHAR(10),
    ownerId VARCHAR(36),
    status ENUM('PENDING', 'JOINED') DEFAULT 'PENDING',
    access_right ENUM('READ', 'WRITE') DEFAULT 'READ',
    createdOn DATETIME DEFAULT CURRENT_TIMESTAMP,
    updatedOn DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (boardId, ownerId)
) ENGINE=InnoDB;
```

We manage this by adding a field in collabs table called status that stores the status of pending or joined.

```
@PostMapping("/{id}/collabs")
public ResponseEntity<Object> createCollab(@RequestHeader(value = "Authorization") String authorizationHeader, @RequestHeader("Auth-Type")
    Board board = permissionCheck(authorizationHeader, String.valueOf(authType), id, "post", false);
    User user = collabService.findUserByEmail(input.getEmail(),authType,authorizationHeader.substring(7));
    CollabOutputDTO newCollab = collabService.mapOutputDTO(collabService.createNewCollab(board, input, user));
    String userName = null;
    if (authorizationHeader != null) userName = getNameFromHeader(authorizationHeader, String.valueOf(authType));
    emailService.sendInviteEmail(input.getEmail(), userName, newCollab.getAccessRight(), board);
    return ResponseEntity.status(HttpStatus.CREATED).body(newCollab);
}
```

PENDING COLLAB/DECLINE INVITE

We manage this by adding a field in collabs table called status that stores the status of pending or joined.

```
@Transactional("firstTransactionManager")
public Collab updateCollabStatus(String boardId, String userId, Boolean isAccept) {
    Collab collab = getCollabOfBoard(boardId, userId, true);
    if (isAccept) {
        collab.setStatus(CollabStatus.JOINED);
        repository.save(collab);
    } else {
        deleteCollab(boardId, userId);
    }
    return collab;
}
```

Use this function in ColllabService to check if the user accepts the invitation. If yes, set the Status to joined. If not, delete the collab in table

```
@PatchMapping("/{id}/collabs/invitation/{isAccept}")
public ResponseEntity<Object> acceptedInvite(@RequestHeader(value = "Authorization", required = false) String authorizationHeader,
@RequestHeader("Auth-Type") String authType, @PathVariable String id, @PathVariable Boolean isAccept) {
    String oid = getOidFromHeader(authorizationHeader, authType);

    CollabOutputDTO collab = collabService.mapOutputDTO(collabService.updateCollabStatus(id, oid, isAccept));
    return ResponseEntity.ok(collab);
}
```

FILE MANAGEMENT

FileStorageProperties

```
4 usages
@ConfigurationProperties(prefix = "file")
@Getter
@Setter
public class FileStorageProperties {
    private String uploadDir;
}
```

config ที่ช่วยทำให้สามารถ config prefix ของไฟล์ได้ เมื่ออัพโหลดไฟล์แล้วจะไปอยู่ที่ไหน

ApplicationProperties

```
file.upload-dir=./public/testfile
```

config ใส่ path ที่เราต้องการให้ไฟล์ต่างๆเก็บไว้

AttachmentEntity

```
@Entity
@Table(name = "attachments")
@Getter
@Setter
public class Attachment {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer attachmentId;


    @JsonIgnore
    @ManyToOne
    @JoinColumns({
            @JoinColumn(name = "taskId", referencedColumnName = "taskId"),
            @JoinColumn(name = "boardId", referencedColumnName = "boardId")
    })
    private Task task;

    @Column(nullable = false)
    private String location;

    @Column(nullable = false)
    private Integer fileSize;

    private LocalDateTime uploadDate = LocalDateTime.now();
}
```

ตัว entity ที่มีโครงสร้างเหมือนกับ database ที่เราใช้จัดเก็บข้อมูลต่างๆสำหรับ attachement

FILE MANAGEMENT

Fileservice

```java
@Service
public class FileService {
    6 usages
    private final Path rootStorageLocation;
    @Autowired
    private AttachmentRepository attachmentRepository;
    @Autowired
    private TaskRepository taskRepository;

    @Autowired
    public FileService(FileStorageProperties fileStorageProperties,
                       AttachmentRepository attachmentRepository,
                       TaskRepository taskRepository) {
        this.attachmentRepository = attachmentRepository;
        this.taskRepository = taskRepository;
        this.rootStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
                .toAbsolutePath().normalize();
        try {
            if (!Files.exists(this.rootStorageLocation))
                Files.createDirectories(this.rootStorageLocation);
        } catch (IOException ex) {
            throw new RuntimeException("Could not create the root directory for uploaded files", ex);
        }
    }
    1 usage
```

storeAttachment (Fileservice)

```java
@Transactional("firstTransactionManager")
public String storeAttachment(MultipartFile file, Integer taskID) throws IOException {
    String originalFileName = Objects.requireNonNull(file.getOriginalFilename());
    if (originalFileName.contains("..")) {...}

    Optional<Task> taskOptional = taskRepository.findById(taskID);
    if (taskOptional.isEmpty()) {
        throw new RuntimeException("Task not found with ID: " + taskID);
    }
    Task task = taskOptional.get();
    String boardId = task.getBoard().getId();

    Path taskDirectory = rootStorageLocation.resolve(Paths.get(boardId, taskID.toString()));
    if (!Files.exists(taskDirectory)) {
        Files.createDirectories(taskDirectory);
    }
    Path targetLocation = taskDirectory.resolve(originalFileName);
    Files.copy(file.getInputStream(), targetLocation, StandardCopyOption.REPLACE_EXISTING);

    Optional<Attachment> existingAttachment = attachmentRepository.findByLocationAndTaskId(targetLocation.to
    Attachment attachment;
    if (existingAttachment.isPresent()) {
        attachment = existingAttachment.get();
    } else {
        attachment = new Attachment();
    }
    attachment.setTask(task);
    attachment.setLocation(targetLocation.toString());
    attachment.setFileSize((int) file.getSize());
    attachment.setUploadDate(LocalDateTime.now());
    attachmentRepository.save(attachment);
    return "File uploaded successfully to nested task directory: " + targetLocation.toString();
}
```

เป็น method ที่ใช้สำหรับการ store attachemnt ซึ่งเมื่อมีไฟล์ส่งเข้ามานั้นจะทำการนำชื่อของ board และ taskId มาสร้างให้เป็น path ดังนี้ :boardId/:taskId

FILE MANAGEMENT

loadFile (Fileservice)

```java
@Transactional("firstTransactionManager")
public ResponseEntity<Resource> loadFile(Integer attachmentId) throws Exception {
    Attachment attachment = attachmentRepository.findById(attachmentId)
            .orElseThrow(() -> new Exception("File not found with ID: " + attachmentId));
    Path filePath = Paths.get(attachment.getLocation()).toAbsolutePath();
    Resource resource = new UrlResource(filePath.toUri());
    if (resource.exists() && resource.isReadable()) {
        String filename = resource.getFilename();
        try {
            String encodedFilename = URLEncoder.encode(filename, enc: "UTF-8").replaceAll( regex: "\\+", repla
            String fileExtension = filename.substring( beginIndex: filename.lastIndexOf( str: ".") + 1).toLowerC
            MediaType mediaType;
            switch (fileExtension) {
                case "pdf": mediaType = MediaType.APPLICATION_PDF;
                    break;
                case "jpg":
                case "jpeg": mediaType = MediaType.IMAGE_JPEG;
                    break;
                case "png": mediaType = MediaType.IMAGE_PNG;
                    break;
                case "gif": mediaType = MediaType.IMAGE_GIF;
                    break;
                default: mediaType = MediaType.APPLICATION_OCTET_STREAM;
            }
            return ResponseEntity.ok()
                    .contentType(mediaType)
                    .header(HttpHeaders.CONTENT_DISPOSITION, ...headerValues: "attachment; filename=\"" + encod
                    .body(resource);
        } catch (UnsupportedEncodingException e) {
            throw new Exception("Error encoding the filename: " + e.getMessage());
        }
    } else {
        throw new Exception("File not found or not readable: " + attachment.getLocation());
```

เป็น method ที่ใช้ในการสำหรับดาวโหลดข้อมูลจาก server เมื่อดาวโหลดไฟล์ประเภทต่างๆ

deleteFile (Fileservice)

```java
@Transactional("firstTransactionManager")
public String deleteFile(Integer attachmentId) throws Exception {
    Attachment attachment = attachmentRepository.findById(attachmentId)
            .orElseThrow(() -> new Exception("File not found with ID: " + attachmentId));

    Path filePath = Paths.get(attachment.getLocation()).toAbsolutePath();
    Path taskDirectoryPath = filePath.getParent();
    Path boardDirectoryPath = taskDirectoryPath.getParent();
    try {
        Files.deleteIfExists(filePath);
    } catch (Exception e) {
        throw new Exception("Failed to delete file: " + e.getMessage());
    }
    attachmentRepository.deleteById(attachmentId);
    try (DirectoryStream<Path> taskDirStream = Files.newDirectoryStream(taskDirectoryPath)) {
        if (!taskDirStream.iterator().hasNext()) {
            Files.deleteIfExists(taskDirectoryPath);
        }
    } catch (Exception e) {
        throw new Exception("Failed to delete task directory: " + e.getMessage());
    }

    try (DirectoryStream<Path> boardDirStream = Files.newDirectoryStream(boardDirectoryPath)) {
        if (!boardDirStream.iterator().hasNext()) {
            Files.deleteIfExists(boardDirectoryPath);
        }
    } catch (Exception e) {
        throw new Exception("Failed to delete board directory: " + e.getMessage());
    }
    return "File and related directories deleted successfully if empty.";

}
```

ใช้ในการ deleteFIle ที่อยู่ในตัว server

FILE MANAGEMENT

loadAsResource (Fileservice)

```java
@Transactional("firstTransactionManager")
public Resource loadFileAsResource(Integer taskId, String fileName) {
    Task task = taskRepository.findById(taskId)
            .orElseThrow(() -> new RuntimeException("Task not found with ID: " + taskId));

    Optional<Attachment> attachmentOpt = task.getAttachments().stream()
            .filter(attachment -> attachment.getLocation().endsWith(fileName))
            .findFirst();

    if (attachmentOpt.isPresent()) {
        Attachment attachment = attachmentOpt.get();
        Path filePath = Paths.get(attachment.getLocation()).toAbsolutePath();

        try {
            Resource resource = new UrlResource(filePath.toUri());
            if (resource.exists() && resource.isReadable()) {
                return resource;
            } else {
                throw new RuntimeException("File not found or not readable: " + fileName);
            }
        } catch (Exception ex) {
            throw new RuntimeException("Could not load file: " + fileName, ex);
        }
    } else {
        throw new RuntimeException("Attachment not found with filename: " + fileName);
    }
}
```

ใช้สำหรับการอ่านข้อมูลภายใน File นั้นๆ

removeAllFileOfTask(Fileservice)

```java
@Transactional("firstTransactionManager")
public void removeAllFileOfTask(Task task) throws Exception {
    List<Attachment> attachments = attachmentRepository.findAllByTask(task);
    for (Attachment attachment : attachments) {
        deleteFile(attachment.getAttachmentId());
    }
}
```

เป็น method ในการที่จะลบไฟล์ทั้งหมดเมื่อมีการลบตัว Task ออก

FILE MANAGEMENT

attachmentRepository

```
5 usages
@Repository
public interface AttachmentRepository extends JpaRepository<Attachment, Integer> {
    1 usage
    public Optional<Attachment> findByLocationAndTaskId(String location, Integer taskId);

    1 usage
    @Query("SELECT SUM(a.fileSize) FROM Attachment a WHERE a.task.id = :taskId")
    public Long getTotalFileSizeByTaskId(Integer taskId);

    1 usage
    public List<Attachment> findAllByTask(Task task);

}
```

เป็น repository ที่เราสามารถดึงฟังก์ชั่นต่างๆที่เชื่อมโยง กับตัว entity ซึ่งในที่นี้ มีการใช้ @Query ของ sql เพื่อที่จะดึง ขนาด attachment ทั้งหมดที่อยู่ใน taskId นั้นเพื่อ handle

loadFile (controller)

```
@GetMapping("/{id}/tasks/{taskId}/attachments/{attachmentId}")
public ResponseEntity<Resource> loadFile(@RequestHeader(value = "Authorization", required = false)
                                         String authorizationHeader,
                                         @RequestHeader("Auth-Type") String authType,
                                         @PathVariable String id,
                                         @PathVariable Integer attachmentId) {
    Board board = permissionCheck(authorizationHeader, authType, id, method: "get", isCollabCanDoOperation: true);
    try {
        return fileService.loadFile(attachmentId);
    } catch (Exception e) {
        return ResponseEntity.notFound().build();
    }
}
```

เป็น controller ที่ใช้ในการดาวน์โหลดไฟล์จาก server

FILE MANAGEMENT

loadFileDisplay (controller)

```
@GetMapping("/{id}/tasks/{taskId}/attachments/displays/{filename:.+}")
public ResponseEntity<Resource> loadFileDisplay(@RequestHeader(value = "Authorization", required = false) String au
                                                @RequestHeader("Auth-Type") String authType,
                                                @PathVariable String id,
                                                @PathVariable Integer taskId, @PathVariable String filename) throws

    Board board = permissionCheck(authorizationHeader, authType, id, method: "get", isCollabCanDoOperation: true);
    Resource fileResource = fileService.loadFileAsResource(taskId, filename);
    String contentType = MediaType.APPLICATION_OCTET_STREAM_VALUE;

    // Encode filename in UTF-8 for compatibility
    String encodedFilename = URLEncoder.encode(Objects.requireNonNull(fileResource.getFilename()), StandardCharsets

    return ResponseEntity.ok().contentType(MediaType.parseMediaType(contentType)).header(HttpHeaders.CONTENT_DISPOS
}
```

เป็น controller ที่ใช้สำหรับในการดึงไฟล์ใาแสดงผลในฝั่ง client แสดงเป็น thumbnail

deleteFile (contoller)

```
@DeleteMapping("/{id}/tasks/{taskId}/attachments/{attachmentId}")
public ResponseEntity<String> deleteFile(@RequestHeader(value = "Authorization", required = false) String authoriza
                                         @RequestHeader("Auth-Type") String authType,
                                         @PathVariable String id,
                                         @PathVariable Integer attachmentId) {
    Board board = permissionCheck(authorizationHeader, authType, id, method: "get", isCollabCanDoOperation: true);
    try {
        String result = fileService.deleteFile(attachmentId);
        return ResponseEntity.ok(result);
    } catch (Exception e) {
        return ResponseEntity.status(500).body("Error: " + e.getMessage());
    }
}
```

เป็น controller ที่ใช้สำหรับการ delete file

FILE MANAGEMENT

updateTask (controller)

```
@PutMapping(path = "/{id}/tasks/{taskId}")
public ResponseEntity<Object> updateTask(@RequestHeader(value = "Authorization") String authorizationHeader, @Reque
    Board board = permissionCheck(authorizationHeader, authType, id, method: "put", isCollabCanDoOperation: true);
    Task oldTask = taskService.findByID(taskId);

    long taskFileSize = Optional.ofNullable(attachmentRepository.getTotalFileSizeByTaskId(taskId)).orElse( other: 0L)
    final long MAX_ATTACHMENT_SIZE = 20 * 1024 * 1024 * 10;
    long totalSize = 0;

    List<String> requestAttachmentLocations = new ArrayList<>();
    List<RequestAttachment> requestAttachments = new ArrayList<>();

    for (Attachment attachmentFile : input.getAttachments()) {
        String fileName = Paths.get(attachmentFile.getLocation()).getFileName().toString();
        if (fileName != null) {
            RequestAttachment newAtt = new RequestAttachment();
            newAtt.setName(fileName);
            newAtt.setFileSize(attachmentFile.getFileSize());
            requestAttachments.add(newAtt);
            requestAttachmentLocations.add(fileName);
        }
    }
```

```
    if (taskFileSize + totalSize > MAX_ATTACHMENT_SIZE) {
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body("Total attachment size exceeds the 20MB limit.");
    }

    if (attachmentFiles != null) {
        for (MultipartFile attachmentFile : attachmentFiles) {
            try {
                fileService.storeAttachment(attachmentFile, taskId);
            } catch (IOException e) {
                return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to store attachment: " +
            }
        }
    }

    TaskOutputAllFieldDTO outputDTO = modelMapper.map(task, TaskOutputAllFieldDTO.class);
    boardService.updateInBoard(id);

    return ResponseEntity.ok(outputDTO);
}
```

```
if (oldTask.getAttachments() != null && !oldTask.getAttachments().isEmpty()) {
    for (Attachment attachment : oldTask.getAttachments()) {
        String fileName = Paths.get(attachment.getLocation()).getFileName().toString();
        if (!requestAttachmentLocations.contains(fileName)) {
            try {
                fileService.deleteFile(attachment.getAttachmentId());
            } catch (Exception e) {
                return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to delete unused attachm
            }
        } else {
            for (RequestAttachment requestAttachment : requestAttachments) {
                if (Objects.equals(requestAttachment.getName(), fileName) && !Objects.equals(requestAttachment.getFi
                    fileService.deleteFile(attachment.getAttachmentId());
                }
            }
        }
    }
}

Task task = taskService.updateTask(taskId, input, id);
```

controller นี้เป็นการอัพเดท Task ซึ่ง task นั้นจะเก็บตัว attachment ไว้ด้วยซึ่งทำให้เวลาที่เราอัพเดท task สามารถที่จะ store ตัวของ attachement ซึ่งจะทำการลูปไฟล์ที่ส่งมาเป็น array ซึ่ง controller นี้จะทำการเช็คก่อนว่า Size เกิดนขนาดหรือจำนวนเกินหรือไม่