- **Classe principale ExempleApplication**

La classe principale de ton application Spring Boot doit permettre de créer automatiquement un dossier files à la racine du projet s'il n'existe pas encore.

```
package com.wakana.exemple;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.transaction.annotation.EnableTransactionManagement;



import java.io.File;

@SpringBootApplication
public class ExempleApplication implements CommandLineRunner {

    public static void main(String[] args) {
        SpringApplication.run(NotalinkApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        //  Créer le dossier "files" si inexistant
        File directory = new File("files");
        if (!directory.exists()) {
            directory.mkdirs();
            System.out.println("Dossier 'files' créé avec succès !");
        } else {
            System.out.println(" Dossier 'files' déjà existant.");
        }
    }
}
```

- **Classe utilitaire FileTransferUtil**

Voici une version simplifiée de ton fichier principal, sans la partie de création d'admin par défaut, et avec un nom d'application plus générique (par exemple NotalinkApplication) :

```
package com.wakana.exemple.util;
```

```
import com.jcraft.jsch.*;
import org.springframework.util.StringUtils;
import org.springframework.web.multipart.MultipartFile;

import java.io.*;
import java.nio.file.*;
import java.util.*;

public class FileTransferUtil {



   private static final String FILE_UPLOAD_DIRECTORY = "./files/";
   private static final String REMOTE_DIR = "/coopachat/";
   private static final String SFTP_HOST = "185.170.213.160";
   private static final int SFTP_PORT = 22;
   private static final String SFTP_USER = "root";
   private static final String SFTP_PASSWORD = "26tXALkVPyGxMEAu#";

   /**
    * Upload a single file locally and transfer it to remote server.
    */
   public static String handleFileUpload(MultipartFile file) throws IOException {
       if (file == null || file.isEmpty()) return "";

       String fileName = generateUniqueFileName(file.getOriginalFilename());
       File uploadDir = new File(FILE_UPLOAD_DIRECTORY);
       if (!uploadDir.exists()) uploadDir.mkdirs();

       Path localFilePath = Paths.get(FILE_UPLOAD_DIRECTORY, fileName);
       Files.copy(file.getInputStream(), localFilePath,
StandardCopyOption.REPLACE_EXISTING);

       transferFileToRemote(localFilePath.toString(), getRemotePath(fileName));
       return fileName;
   }

   /**
    * Upload multiple pictures.
    */
   public static List<String> uploadPictures(List<MultipartFile> pictures) throws
IOException {
       List<String> pictureUrls = new ArrayList<>();
       if (pictures == null || pictures.isEmpty()) return pictureUrls;

       for (MultipartFile picture : pictures) {
```

```
            if (picture != null && !picture.isEmpty()) {
                String fileName = handleFileUpload(picture);
                if (!fileName.isEmpty()) {
                    pictureUrls.add(fileName);
                }
            }
        }
        return pictureUrls;
    }

    /**
     * Generate unique file name with original extension.
     */
    public static String generateUniqueFileName(String originalFilename) {
        String extension = StringUtils.getFilenameExtension(originalFilename);
        return UUID.randomUUID().toString() + (extension != null ? "." + extension :
"");
    }

    /**
     * Transfer file to remote server using SFTP.
     */
    public static void transferFileToRemote(String localFilePath, String
remoteFilePath) {
        try (SftpClient sftp = new SftpClient()) {
            sftp.uploadFile(localFilePath, remoteFilePath);
        } catch (Exception e) {
            System.err.println("[ERREUR] Transfert échoué : " + e.getMessage());
            e.printStackTrace();
        }
    }

    /**
     * Delete a remote file using SFTP.
     */
    public static void deleteRemoteFile(String remoteFilePath) {
        try (SftpClient sftp = new SftpClient()) {
            sftp.deleteFile(remoteFilePath);
            System.out.println("[INFO] Fichier supprimé : " + remoteFilePath);
        } catch (Exception e) {
            System.err.println("[ERREUR] Suppression échouée : " + e.getMessage());
            e.printStackTrace();
        }
    }
```

```
    public static String getRemotePath(String fileName) {
        return REMOTE_DIR + fileName;
    }

    /**
     * Internal SFTP Client to avoid code repetition.
     */
    private static class SftpClient implements AutoCloseable {
        private final Session session;
        private final ChannelSftp sftpChannel;

        public SftpClient() throws JSchException {
            JSch jsch = new JSch();
            session = jsch.getSession(SFTP_USER, SFTP_HOST, SFTP_PORT);
            session.setPassword(SFTP_PASSWORD);

            Properties config = new Properties();
            config.put("StrictHostKeyChecking", "no");
            session.setConfig(config);
            session.connect();

            Channel channel = session.openChannel("sftp");
            channel.connect();
            sftpChannel = (ChannelSftp) channel;
        }

        public void uploadFile(String localFilePath, String remoteFilePath) throws
SftpException, IOException {
            try (InputStream inputStream = new FileInputStream(localFilePath)) {
                sftpChannel.put(inputStream, remoteFilePath);
            }
        }

        public void deleteFile(String remoteFilePath) throws SftpException {
            sftpChannel.rm(remoteFilePath);
        }

        @Override
        public void close() {
            if (sftpChannel != null && sftpChannel.isConnected())
sftpChannel.disconnect();
            if (session != null && session.isConnected()) session.disconnect();
        }
    }
}
```

Documentation – Upload d’images (Single & Multiple)

Objectif

Cette documentation explique comment mettre en place un endpoint permettant d’envoyer une ou plusieurs images au serveur à l’aide de Spring Boot.

Définition du DTO

On définit un objet `FileRequest` qui contient :

- un champ texte d’exemple (`title`),
- un champ pour une seule image (`oneImg`),
- un champ pour plusieurs images (`imgs`).

Exemple :

```
@Data
public class FileRequest {
    private String title;              // Exemple d'attribut texte
    private MultipartFile oneImg;      // Pour une seule image
    private List<MultipartFile> imgs;  // Pour plusieurs images
}
```

📌 Endpoint REST

On crée un contrôleur `FileController` qui expose un endpoint `/upload`.
Cet endpoint accepte des données en `multipart/form-data` grâce à l’annotation

`@PostMapping`.

Exemple :

```
@RestController
@RequestMapping("/api/files")
public class FileController {

    private final FileService fileService;

    public FileController(FileService fileService) {
        this.fileService = fileService;
    }

    @PostMapping(value = "/upload", consumes =
MediaType.MULTIPART_FORM_DATA_VALUE)
    public ResponseEntity<FileResponse> upload(@ModelAttribute
FileRequest request) {
        return ResponseEntity.ok(fileService.upload(request));
    }
}
```

Service de traitement

Le service `FileService` gère la logique d'upload.

- S'il y a une seule image (`oneImg`), on utilise un utilitaire pour sauvegarder le fichier et stocker son URL.
- S'il y a plusieurs images (`imgs`), on parcourt la liste et on enregistre toutes les URLs.

Exemple :

```
@Service
public class FileService {

    public FileResponse upload(FileRequest request) {
        ExampleEntity entity = new ExampleEntity();

        // Upload d'une seule image
        try {
```

```
            if (request.getOneImg() != null) {
                String icon =
FileTransferUtil.handleFileUpload(request.getOneImg());
                entity.setIcon(icon);
            }
        } catch (IOException e) {
            throw new IllegalArgumentException("Une image unique est
obligatoire");
        }

        // Upload de plusieurs images
        try {
            if (request.getImgs() != null &&
!request.getImgs().isEmpty()) {
                List<String> urls =
FileTransferUtil.uploadPictures(request.getImgs());
                entity.getPictures().addAll(urls);
            }
        } catch (IOException e) {
            throw new RuntimeException("Erreur lors de l'upload des
images", e);
        }

        return new FileResponse(entity.getId(), entity.getTitle(),
entity.getIcon(), entity.getPictures());
    }
}
```

Lien vers les répertoires pour accéder aux fichiers (images, PDF, etc.) :
pour reseau medical :http://185.170.213.160/repertoire_rml/