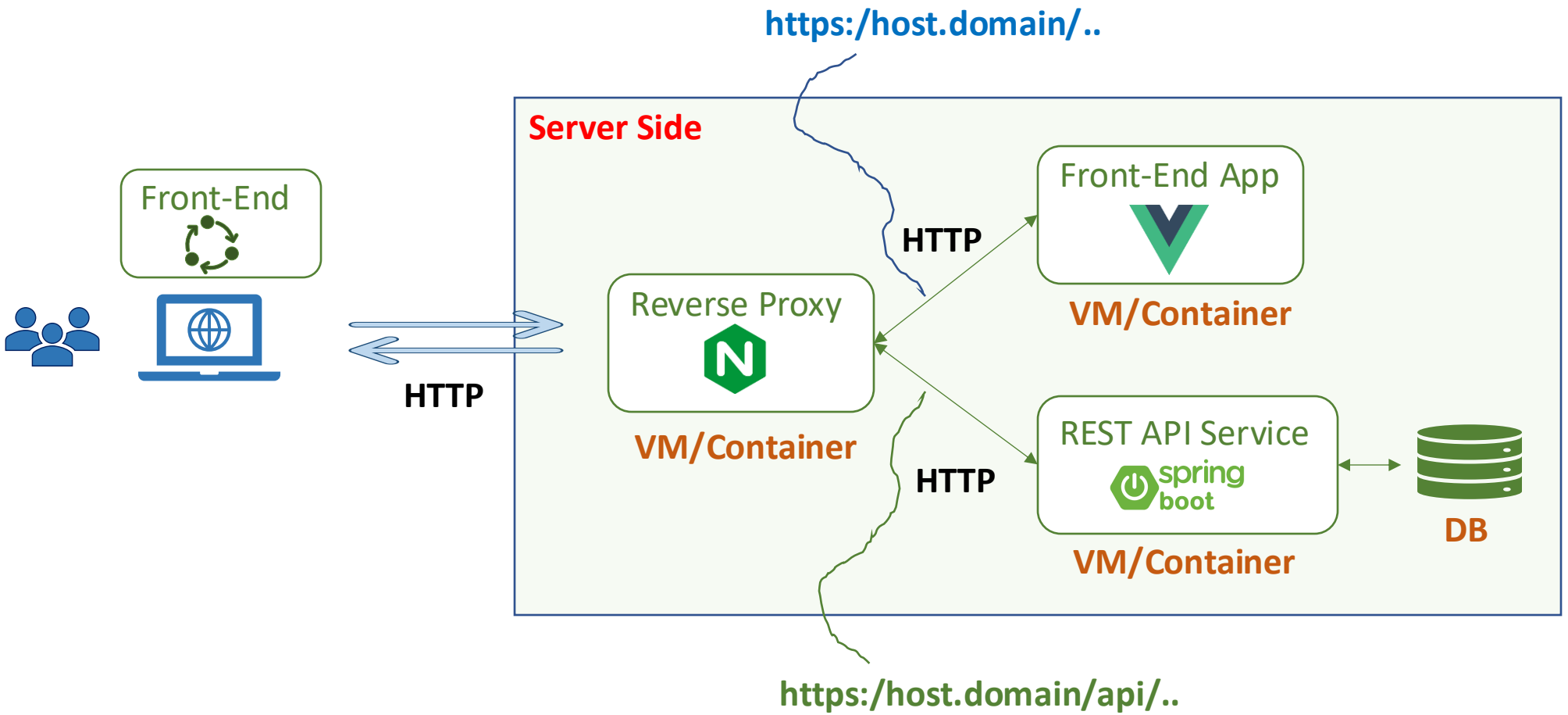
Spring RESTful API
File Services

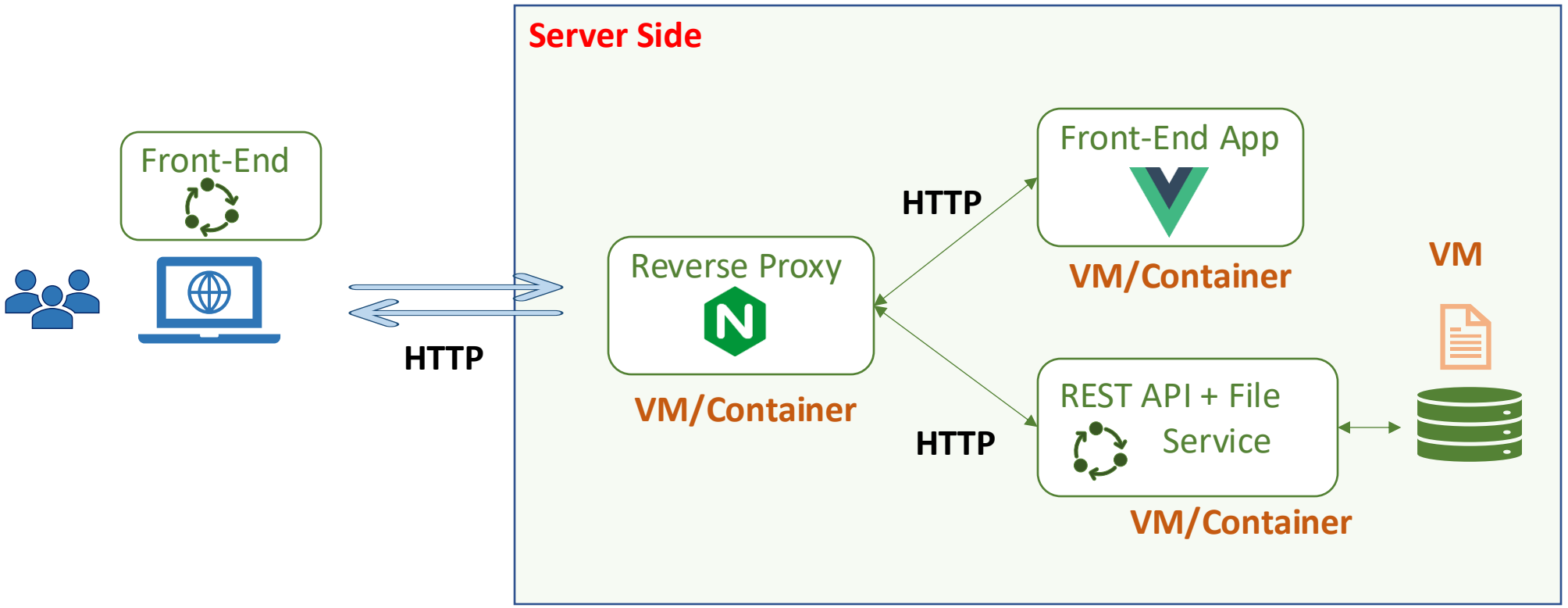
By

Pichet Limvajiranan

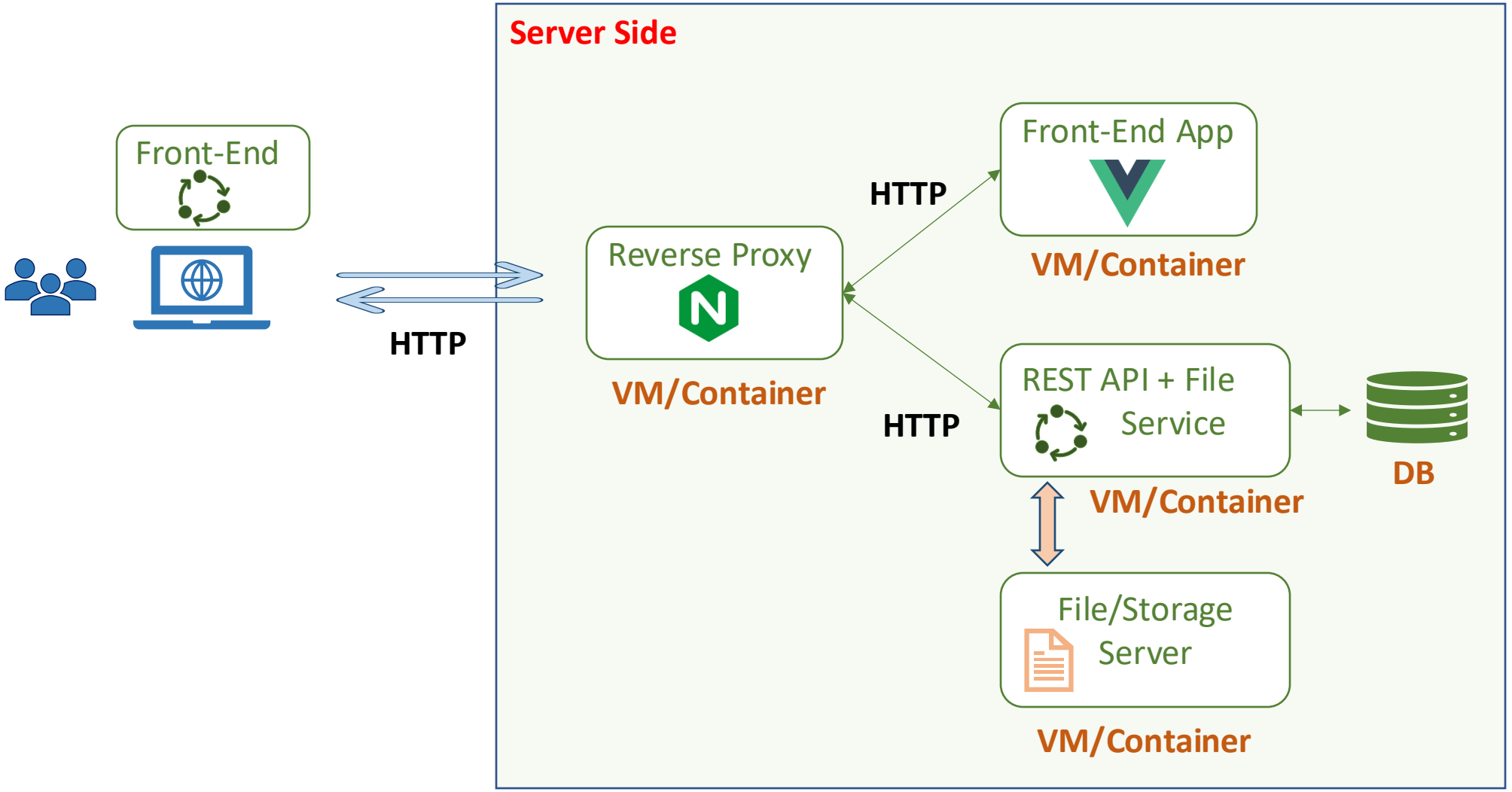
SPA Running Environment



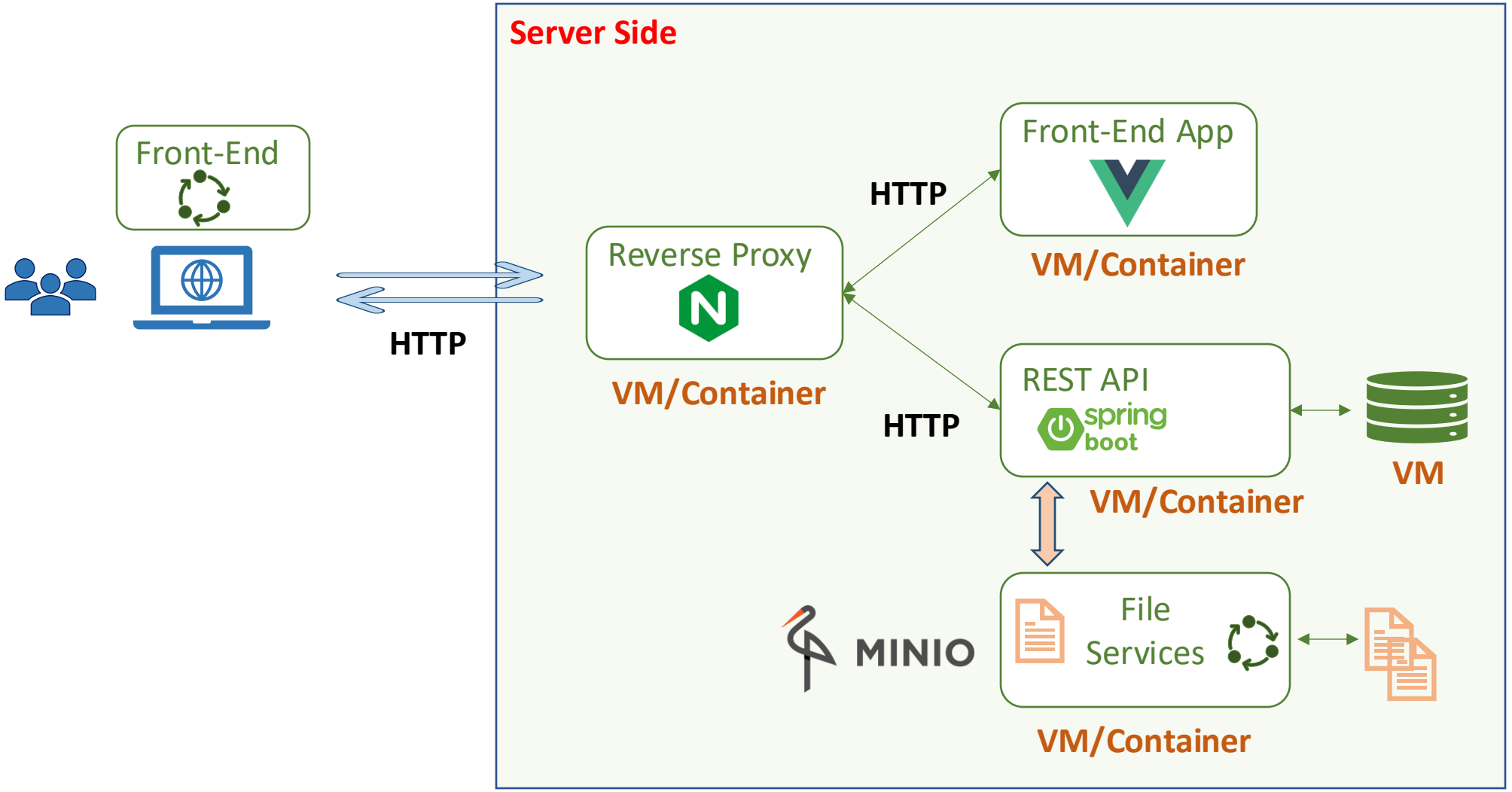
File Storage Architectures (1)



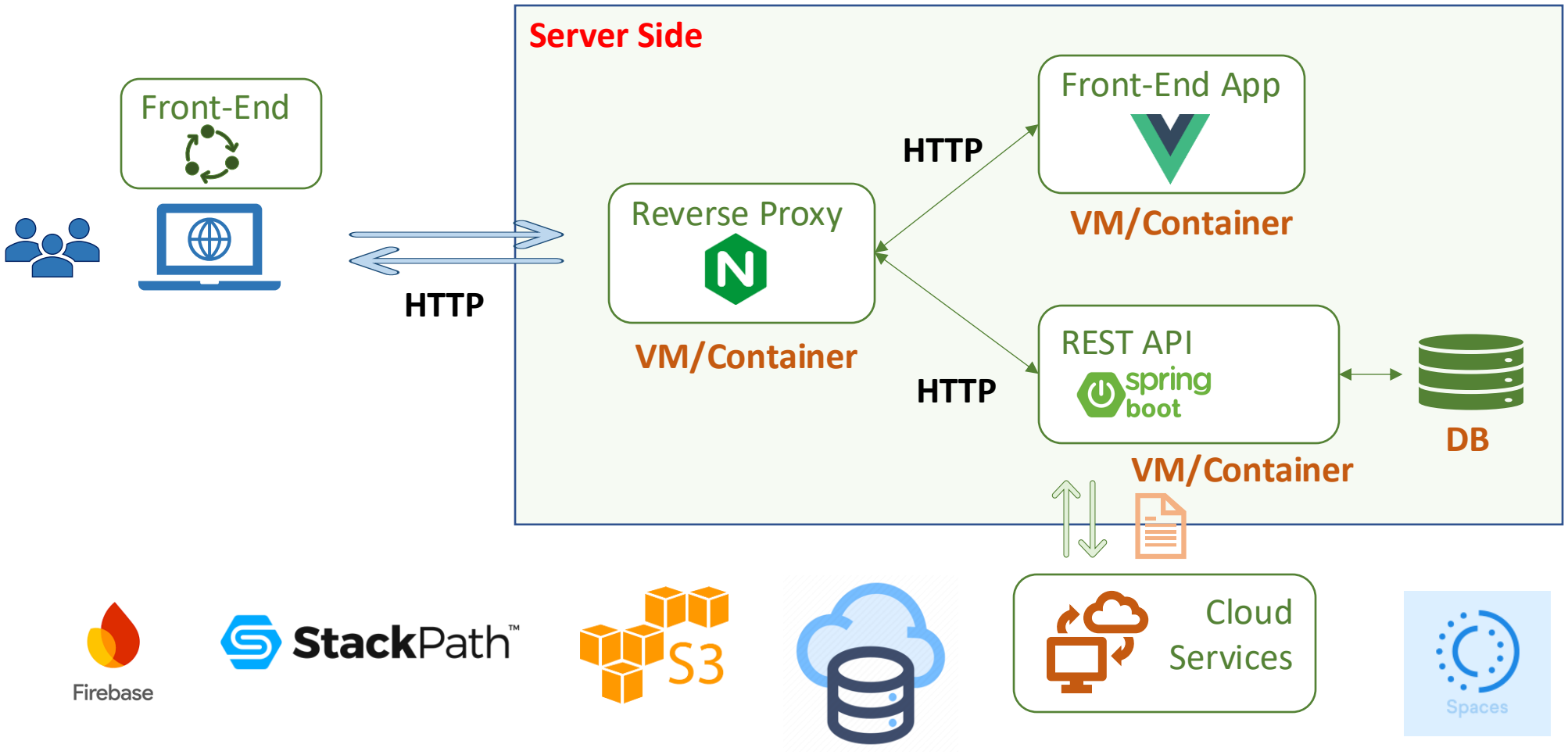
File Storage Architectures (2)



File Storage Architectures (3)



File Storage Architectures (4)



File Operation

- Upload
  - Send file – Form Data (Multipart) / File Only (FE)
  - **Read file data from Http Request (BE)**
  - **Write file to storage (BE)**
- Download
  - Return file URL / File content
- Delete
  - Read file name from Http Request
  - Remove file from storage

Configuring Server and File Storage Properties

```
## MULTIPART (MultipartProperties)
# Enable multipart uploads
spring.servlet.multipart.enabled=true

# Max file size.
spring.servlet.multipart.max-file-size=10MB

# Max Request Size
spring.servlet.multipart.max-request-size=80MB

# file-size-threshold specifies the size threshold after which files will
be written to disk. Default is 0, which means that the file will be written
to disk immediately.
spring.servlet.multipart.file-size-threshold=512KB

## File Storage Properties # All files uploaded through the REST API will
be stored in this directory
file.upload-dir=./product-images
```

Automatically binding properties to a POJO class

- Spring Boot has an awesome feature called @ConfigurationProperties using which you can automatically bind the properties defined in the application.properties file to a POJO class.
- Let's define a POJO class called FileStorageProperties inside based-package.properties package to bind all the file storage properties.

```
file.upload-dir=./product-images
```

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

```
@SpringBootApplication
@EnableConfigurationProperties({
      FileStorageProperties.class
})
public class DemoApplication {
   public static void main(String[] args) {
```

Binding properties to a POJO class

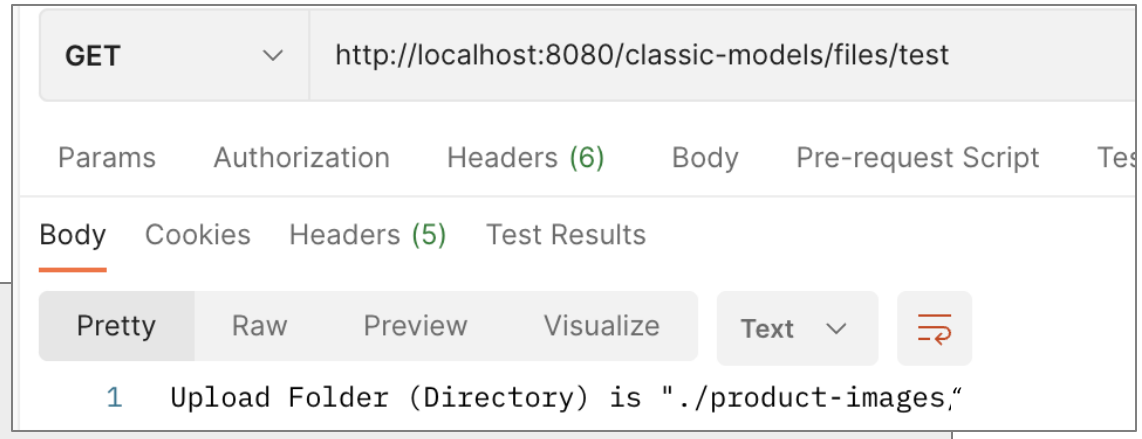
Create Configuration property class: FileStorageProperties.java (package based-package.properties

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

Register Configuration Property (into ApplicationConfig.java or ClassicmodelServiceApplication.java)

```
@EnableConfigurationProperties({
        FileStorageProperties.class
})
```

Test properties/POJO binding




```java
@RestController
@RequestMapping("/files")
public class FileController {
    @Autowired
    FileStorageProperties fileStorageProperties;

    @GetMapping("/test")
    public ResponseEntity<Object> testPropertiesMapping() {
        return ResponseEntity.ok("Upload Folder (Directory) is \""
                + fileStorageProperties.getUploadDir()+ "\"");
    }
}
```

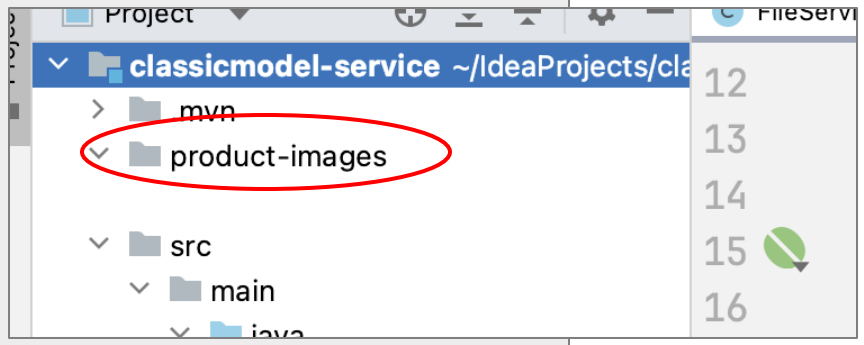
FileService : Initialize

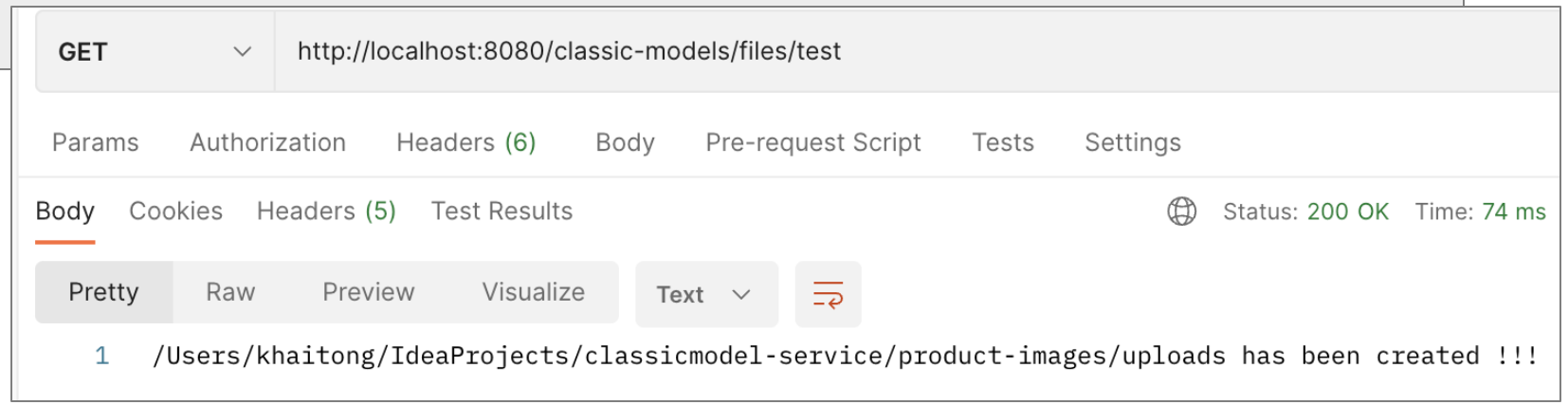
```
import java.io.IOException;
import java.net.MalformedURLException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
```

```
@Service
@Getter
public class FileService {
   private final Path fileStorageLocation;
   @Autowired
   public FileService(FileStorageProperties fileStorageProperties) {
     this.fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
     .toAbsolutePath().normalize();
     try {
       if (! Files.exists(this.fileStorageLocation)) {
           Files.createDirectories(this.fileStorageLocation);
       }
     } catch (IOException ex) {
       throw new RuntimeException(
         "Can't create the directory where the uploaded files will be stored.", ex);
     }
}
```

Modify FileController to Test FileService - Initialize

```java
@RestController
@RequestMapping("/files")
public class FileController {
    @Autowired
    FileService fileService;

    @GetMapping("/test")
    public ResponseEntity<Object> testPropertiesMapping() {
        return ResponseEntity.ok(fileService.getFileStorageLocation()+ " has been created !!!");
    }
}
```



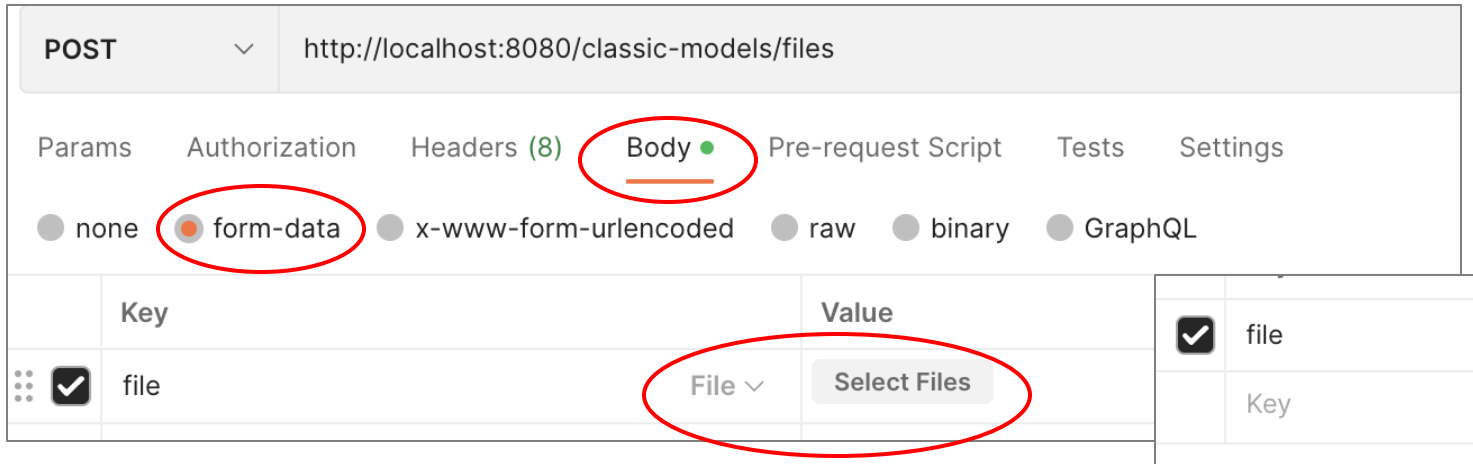


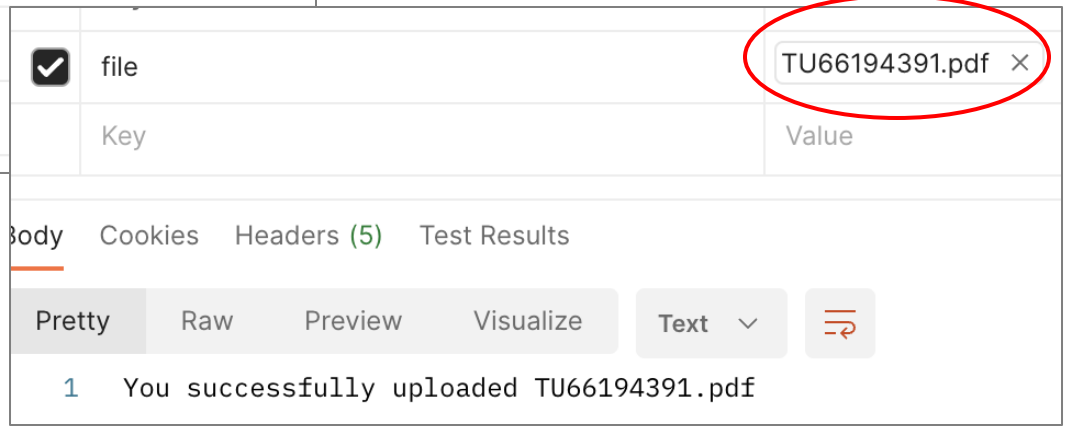
FileService: Storing File (add this method to FileService)

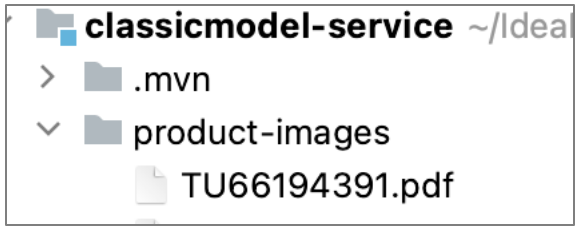
```
public String store(MultipartFile file) {
    // Normalize file name
    String fileName = StringUtils.cleanPath(file.getOriginalFilename());
    try {
        // Check if the file's name contains invalid characters
        if (fileName.contains("..")) {
            throw new RuntimeException("Sorry! Filename contains invalid path sequence " + fileName);
        }
        // Copy file to the target location (Replacing existing file with the same name)
        Path targetLocation = this.fileStorageLocation.resolve(fileName);
        Files.copy(file.getInputStream(), targetLocation, StandardCopyOption.REPLACE_EXISTING);
        return fileName;
    } catch (IOException ex) {
        throw new RuntimeException("Could not store file " + fileName + ". Please try again!", ex);
    }
}
```

Add this method to FileController - Test FileService – Storing File

```
@PostMapping("")
public ResponseEntity<Object> fileUpload(@RequestParam("file") MultipartFile file) {
   fileService.store(file);
   return ResponseEntity.ok("You successfully uploaded " + file.getOriginalFilename());
}
```







FileService: LoadFile

```
public Resource loadFileAsResource(String fileName) {
   try {
      Path filePath = this.fileStorageLocation.resolve(fileName).normalize();
      Resource resource = new UrlResource(filePath.toUri());
      if (resource.exists()) {
         return resource;
      } else {
         throw new RuntimeException("File not found " + fileName);
      }
   } catch (MalformedURLException ex) {
      throw new RuntimeException("File operation error: "
        + fileName, ex);
   }
}
```

File Controller

```
@GetMapping("/{filename:.+}")
@ResponseBody
public ResponseEntity<Resource> serveFile(
   @PathVariable String filename) {
    Resource file = fileService.loadFileAsResource(filename);
    return ResponseEntity.ok()
      .contentType(MediaType.IMAGE_JPEG).body(file);
}
```

Delete File

```
// Rest Controller
@DeleteMapping("/{filename:.+}")
public ResponseEntity<Object> removeFile(@PathVariable String filename) {
    fileService.removeFile(filename);
    return ResponseEntity.ok(filename+ " has been removed !");
}

// Service
public void removeFile(String fileName) {
    try {
        Path filePath = this.fileStorageLocation.resolve(fileName).normalize();
        if (Files.exists(filePath)) {
            Files.delete(filePath);
        } else {
            throw new ResourceNotFoundException("File not found " + fileName);
        }
    } catch (IOException ex) {
        throw new RuntimeException("File operation (DELETE) error: " + fileName, ex);
    }
}
```

Response Header: Content-Type

FileService.java

```
public String getFileType(Resource resource) {
    try {
        return Files.probeContentType(resource.getFile().toPath());
    } catch (IOException ex) {
        throw new RuntimeException("ProbeContentType error: " + resource,ex);
    }
}
```

FileController.java

```
@GetMapping("/{filename:.+}")
@ResponseBody
public ResponseEntity<Resource> serveFile(@PathVariable String filename) {
   Resource file = fileService.loadFileAsResource(filename);
   return ResponseEntity.ok().contentType(MediaType.valueOf(fileService.getFileType(file))).body(file);
}
```

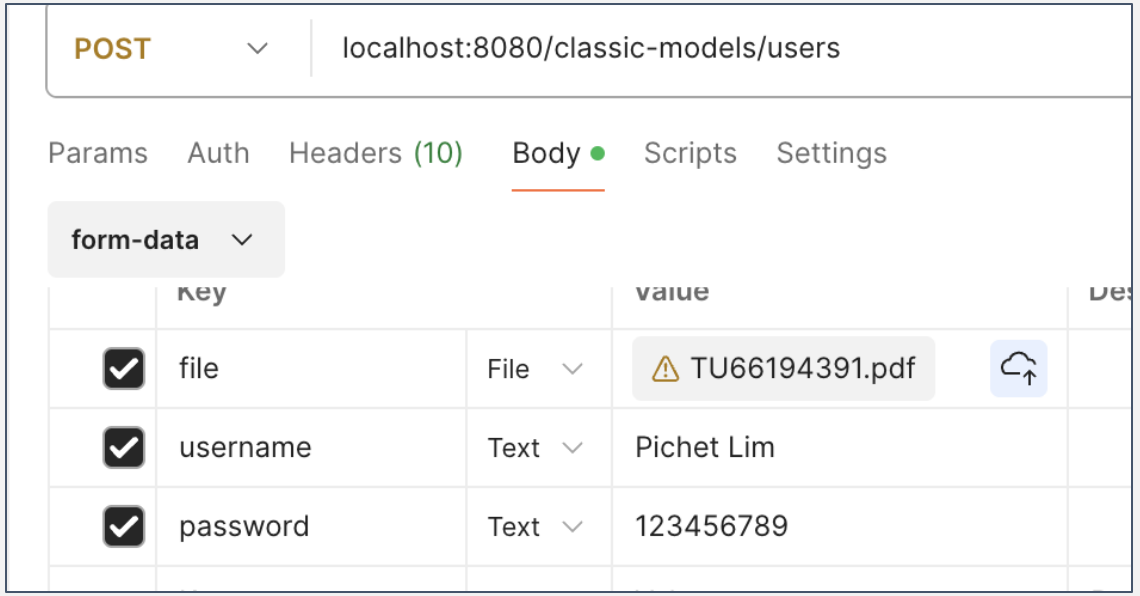
Validate Support File Type

```
private boolean isSupportedContentType(MultipartFile file) {
    String contentType = file.getContentType();
    return contentType.equals("application/vnd.openxmlformats-
officedocument.wordprocessingml.document")
            || contentType.equals("application/pdf")
            || contentType.equals("image/png")
            || contentType.equals("image/jpg")
            || contentType.equals("image/jpeg");
}

public String store(MultipartFile file) {    // Normalize file name
  if(! isSupportedContentType(file)) {
    throw new ResponseStatusException(HttpStatus.BAD_REQUEST, "Does not support content type: " + file.getContentType());
  }
```

Multipart Request Handling in Spring (1)

Using @ModelAttribute and (@RequestParam or @RequestPart)

```
@PostMapping("")
public ResponseEntity<Object> createUser(@ModelAttribute JwtRequestUser user
      , @RequestParam MultipartFile file) {
   fileService.store(file);
   return ResponseEntity.ok(user);
}

@Data
public class JwtRequestUser {
   private String username;
   @Size(min = 8).  @NotBlank
   private String password;
}
```

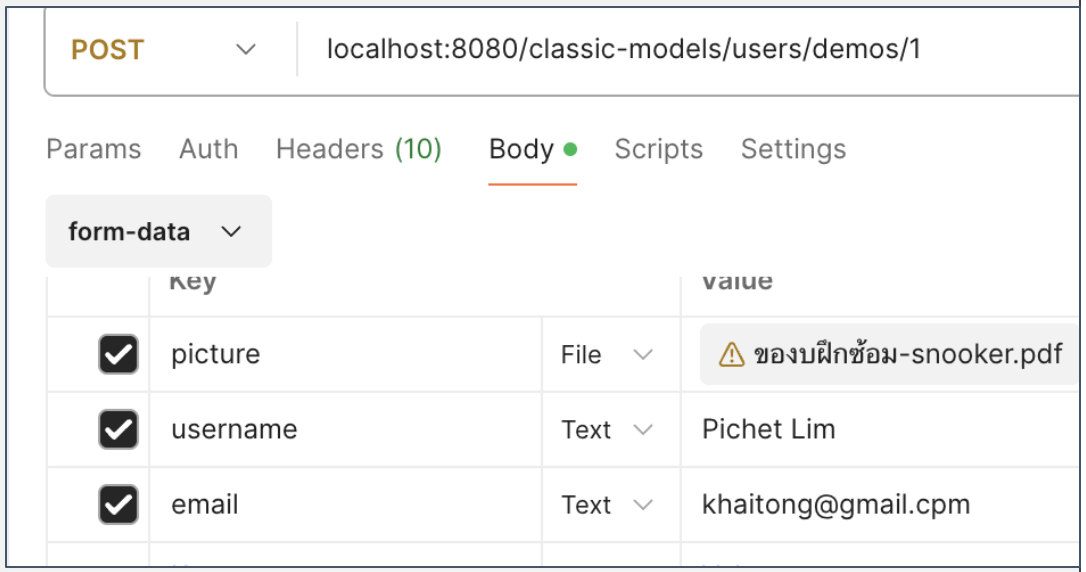


Multipart Request Handling in Spring (2)

Using @ModelAttribute

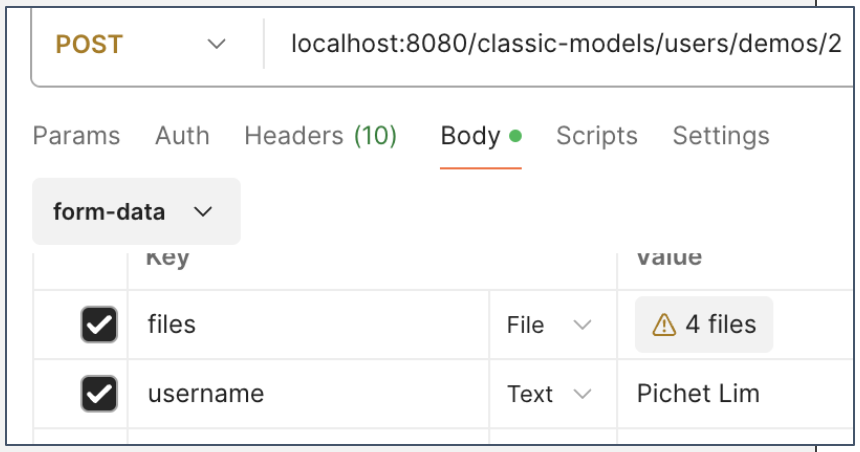
```
@PostMapping("/demos/1")
public ResponseEntity<Object> createUserDemo(
        @ModelAttribute ProfileDto user) {
    fileService.store(user.getPicture());
    return ResponseEntity.ok(user);
}
```

```
@Data
public class ProfileDto {
    private String username;
    private String email;
    @JsonIgnore
    private MultipartFile picture;
}
```



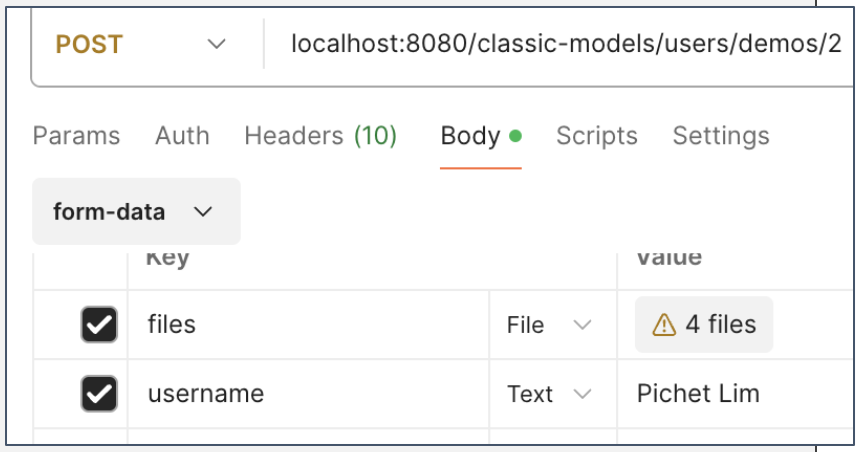
Multiple Files Uploading

```java
// Controller
@PostMapping("/demos/2")
public ResponseEntity<Object> createUserDemo2( @ModelAttribute JwtRequestUser user
        , @RequestPart List<MultipartFile> files) {
    fileService.store(files);
    return ResponseEntity.ok(user);
}

// Service
public List<String> store(List<MultipartFile> files) {
    List<String> fileNames = new ArrayList<>(files.size());
    files.forEach(file -> fileNames.add(store(file)));
    return fileNames;
}
```



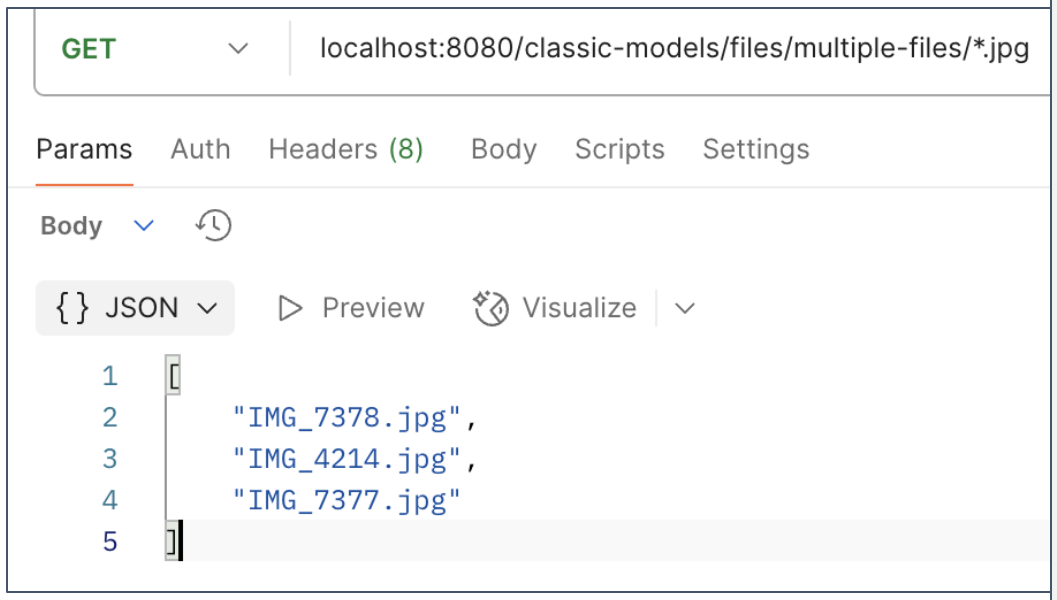
Multiple Files Uploading

```java
// Controller
@PostMapping("/demos/2")
public ResponseEntity<Object> createUserDemo2( @ModelAttribute JwtRequestUser user
        , @RequestPart List<MultipartFile> files) {
    fileService.store(files);
    return ResponseEntity.ok(user);
}

// Service
public List<String> store(List<MultipartFile> files) {
    List<String> fileNames = new ArrayList<>(files.size());
    files.forEach(file -> fileNames.add(store(file)));
    return fileNames;
}
```



Multiple Files Downloading

```
// Controller
@GetMapping("/multiple-files/{pattern:.+}")
@ResponseBody
public ResponseEntity<List<String>>
getFiles(@PathVariable String pattern) {
    return ResponseEntity.ok(
   fileService.getMatchedFiles(pattern));
}
```




```
// Service
public List<String> getMatchedFiles(String pattern) {
  List<String> matchesList = new ArrayList<String>();
  FileVisitor<Path> matcherVisitor = new SimpleFileVisitor<Path>() {
    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attribs)
     throws IOException {
      FileSystem fs = FileSystems.getDefault();
      PathMatcher matcher = fs.getPathMatcher("glob:"+pattern);
      Path name = file.getFileName();
      if (matcher.matches(name)) {
        matchesList.add(name.toString());
      }
      return FileVisitResult.CONTINUE;
    }
  };
  try {
    Files.walkFileTree(this.fileStorageLocation, matcherVisitor);
  } catch (IOException ex) {
    throw new RuntimeException(ex.getMessage());
  }
  return matchesList;
}
```