Uploading Files with Server Application

Description: Spring Boot is an open source, micro service-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its own codebase.

Project: Create a simple webpage to upload a file to a server application that receives HTTP multi-part file uploads. This project is based on Uploading Files and just expands on the screenshots and step by step instructions. This project also sequences building the application in attempt to reduce dependency errors. Finally it appears the website expected the user to download or branch out of the Git Hub repository since a few class files are omitted from the website, this document has the information to create the files.

Technology: This project uses the following technology:

- Integrated Development Environment (IDE):
    - Spring Tool Suite 4 (Version: 4.11.0.RELEASE)
- Java Development Kit (JDK):
    - Oracle's JDK 8 (1.8)

Table of Contents

Glossary of Terminology 3

Generate Spring Boot Download 3

Import the Spring Boot Download 3

  Import the project: “uploading-files” 3

Project Discussion 5

Create the Project from the Import 5

  Main Application Class: UploadingFilesApplication 5

  Create View Component: uploadForm.html 5

  Add Properties to Property File: 6

  Create Interface: StorageService 6

  Create Exception Class: StorageException 6

  Create Exception Class: StorageFileNotFoundException 7

  Create Class: StorageProperties 7

  Update File: pom.xml 8

  Create Class: FileSystemStorageService 8

  Create File Uploader Class: FileUploadController 10

  Update Main Application Class: UploadingFilesApplication 11

Initial Test of the Application 12

Other Testing of the Application 12

Project Conclusion 12

Glossary of Terminology

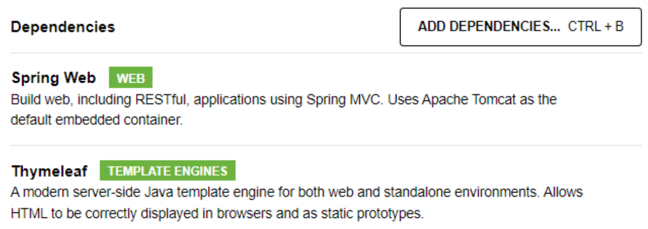
For a list of key terms and definitions used throughout this and various Spring Boot demo documents see the document titled “Appendix 01 Glossary”.

Generate Spring Boot Download

Follow the instructions in the document title “Appendix 02 Spring Initializr” to generate a spring boot download for this project.

When the document talks about adding dependencies add only these:

Spring Web
Thymeleaf



Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

When the document talks about the items in the “**Project Metadata**” use the values shown below:

“**Group**” use “**com.example**”
“**Artifact**” use “**uploading-files**”
“**Name**” use “**uploading-files**”
“**Package name**” use “**com.example.uploading-files**”

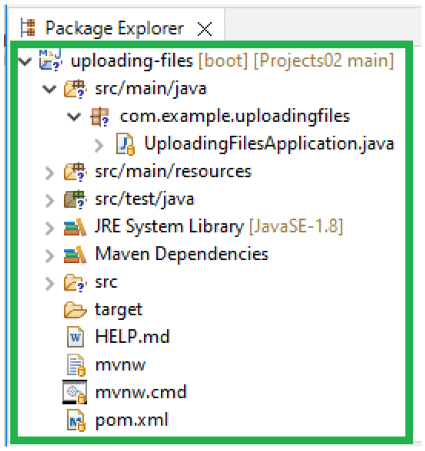
For other items in the “**Project Metadata**” use the defaults. Follow the instructions to extract the files from the zip file into the Spring Tool Suite (STS) 4 workspace.

Import the Spring Boot Download

Open the Spring Tool Suite 4 IDE.

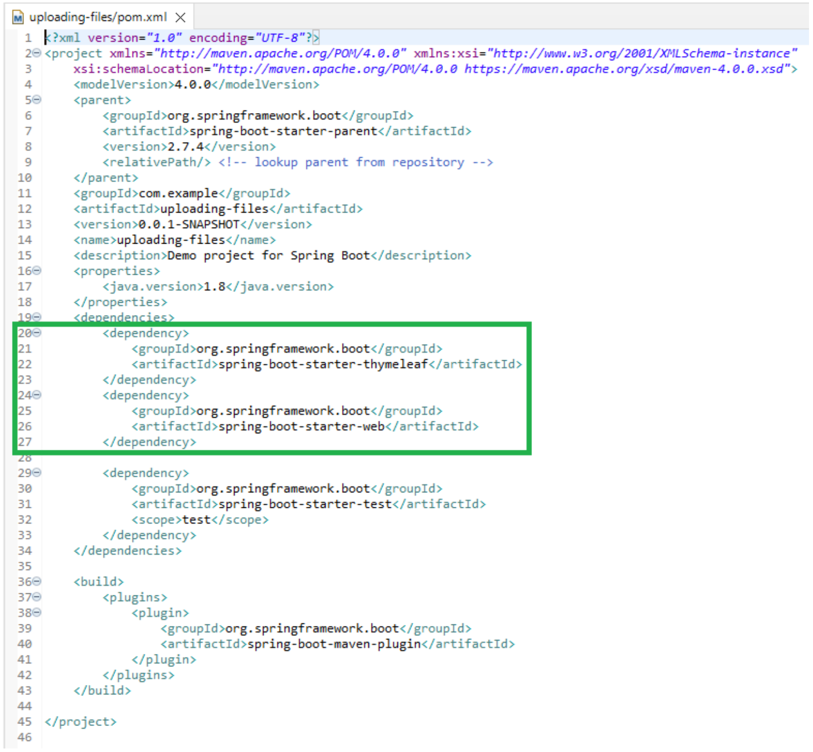
Import the project: “uploading-files”

Follow the instructions in the document title “Appendix 03 Import Spring Tool Suite Project” and import the “**uploading-files**” project that was created with Spring Initializr. After importing the project, it should look like the following using the IDE “Package Explorer”.



```
uploading-files [boot] [Projects02 main]
    src/main/java
        com.example.uploadingfiles
            UploadingFilesApplication.java
    src/main/resources
    src/test/java
    JRE System Library [JavaSE-1.8]
    Maven Dependencies
    src
    target
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
```

Look at the pom.xml and find the added dependencies for this project.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.4</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.example</groupId>
    <artifactId>uploading-files</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>uploading-files</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-thymeleaf</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

Project Discussion

This project will create a file upload controller and a simple HTML page. The project will be tested within the IDE using the Spring Boot built in Tomcat server.

As mentioned at the beginning of this document this project sequences building the application in a manner to reduce dependency errors. This project pulls missing code from the project website "Uploading Files" from the GIT Hub repository specifically code under the "complete/src" area. One file of concern is the FileUploadController class. The version from the website threw a startup error. Replacing the code from the repository corrected the error and at the section in this document "Initial Test of the Application" the program ran successfully.

Create the Project from the Import

Use the STS 4 IDE to modify the code after importing the project.

Main Application Class: UploadingFilesApplication

The class file "UploadingFilesApplication" containing the startup method generated by Spring Initializr tool is used with modification. The update will be performed later in this document after reference classes are defined.

Create View Component: uploadForm.html

This project uses a simple HTML page as a view component within the STS 4 IDE. The view component is created in the resources folder "src/main/resources/templates". The view components will incorporate Thymeleaf the dependency added in the pom.xml.

Create a file under src/main/resources/templates named: "uploadForm.html"
Right click on "templates" folder ➔ "New" ➔ "File"
Add the following code to the file:

```
<html xmlns:th="https://www.thymeleaf.org">
<body>

        <div th:if="${message}">
                <h2 th:text="${message}"/>
        </div>

        <div>
                <form method="POST" enctype="multipart/form-data" action="/">
                        <table>
                                <tr><td>File to upload:</td><td><input type="file" name="file" /></td></tr>
                                <tr><td></td><td><input type="submit" value="Upload" /></td></tr>
                        </table>
                </form>
        </div>

        <div>
                <ul>
                        <li th:each="file : ${files}">
                                <a th:href="${file}" th:text="${file}" />
                        </li>
                </ul>
        </div>

</body>
</html>
```

Add Properties to Property File:

Related to uploading a file we add properties to configure file upload size.

Tuning File Upload Limits

When configuring file uploads, it is often useful to set limits on the size of files. Imagine trying to handle a 5GB file upload! With Spring Boot, we can tune its auto-configured `MultipartConfigElement` with some property settings. – "Uploading Files"

Open existing file under src/main/resources: "application.properties".
This file initially contains no properties.
Add the following to the property file:

```
spring.servlet.multipart.max-file-size=128KB
spring.servlet.multipart.max-request-size=128KB
```

Create Interface: StorageService

In a production scenario, you more likely would store the files in a temporary location, a database, or perhaps a NoSQL store (such as Mongo's GridFS). It is best to NOT load up the file system of your application with content.

You will need to provide a `StorageService` so that the controller can interact with a storage layer (such as a file system). The following listing (from `src/main/java/com/example/uploadingfiles/storage/StorageService.java`) shows that interface: – "Uploading Files"

Create a package: "com.example.uploadingfiles.storage"
Create an interface under the package just created named: "StorageService".

Copy and replace the code shell with the following code:

```
package com.example.uploadingfiles.storage;

import org.springframework.core.io.Resource;
import org.springframework.web.multipart.MultipartFile;

import java.nio.file.Path;
import java.util.stream.Stream;

public interface StorageService {

          void init();

          void store(MultipartFile file);

          Stream<Path> loadAll();

          Path load(String filename);

          Resource loadAsResource(String filename);

          void deleteAll();

}
```

Create Exception Class: StorageException

This class was not defined on the website and was adapted from the GitHub Repro.

Create an exception class under package "com.example.uploadingfiles.storage" named: "StorageException" that extends RuntimeException.

Copy and replace the code shell with the following code:

```
package com.example.uploadingfiles.storage;

public class StorageException extends RuntimeException {

	private static final long serialVersionUID = 1L;

	public StorageException(String message) {
		super(message);
	}

	public StorageException(String message, Throwable cause) {
		super(message, cause);
	}
}
```

Create Exception Class: StorageFileNotFoundException

This class was not defined on the website and was adapted from the GitHub Repro.

Create an exception class under package “com.example.uploadingfiles.storage” named: “StorageFileNotFoundException” that extends StorageException.

Copy and replace the code shell with the following code:

```
package com.example.uploadingfiles.storage;

public class StorageFileNotFoundException extends StorageException {

	private static final long serialVersionUID = 1L;

	public StorageFileNotFoundException(String message) {
		super(message);
	}

	public StorageFileNotFoundException(String message, Throwable cause) {
		super(message, cause);
	}
}
```

Create Class: StorageProperties

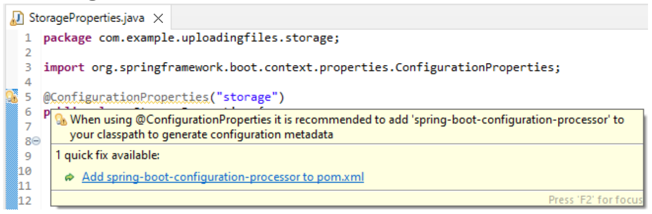
This class was not defined on the website and was taken from the GitHub Repro.

Create a class under package “com.example.uploadingfiles.storage” named: “StorageProperties”.

Copy and replace the code shell with the following code:

```
package com.example.uploadingfiles.storage;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties("storage")
public class StorageProperties {

	/**
	 * Folder location for storing files
	 */
	private String location = "upload-dir";

	public String getLocation() {
		return location;
	}

	public void setLocation(String location) {
		this.location = location;
	}

}
```

Add this class produced a warning. This might be due to the age of the project from the website. The Git Hub repository was last updated 3 years ago. In the next section the pom.xml will be updated as the warning indicates.



StorageProperties.java

```
package com.example.uploadingfiles.storage;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties("storage")
```

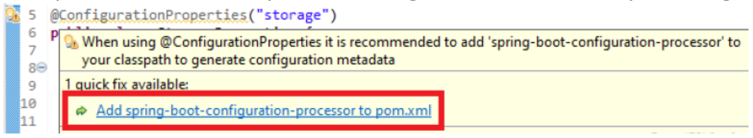
When using @ConfigurationProperties it is recommended to add 'spring-boot-configuration-processor' to your classpath to generate configuration metadata

1 quick fix available:

Add spring-boot-configuration-processor to pom.xml

Update File: pom.xml

The pom.xml can be updated using the IDE Wizard by clicking the link.



```
@ConfigurationProperties("storage")
```

When using @ConfigurationProperties it is recommended to add 'spring-boot-configuration-processor' to your classpath to generate configuration metadata

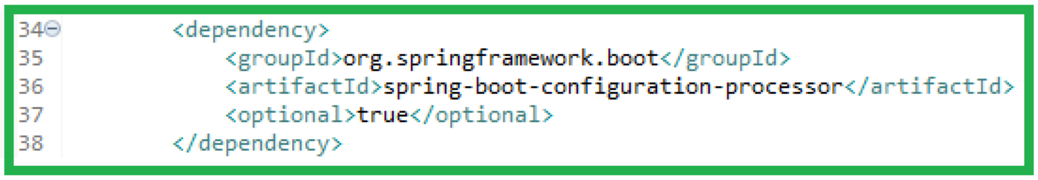
1 quick fix available:

Add spring-boot-configuration-processor to pom.xml

Alternatively you can add the following to the pom.xml

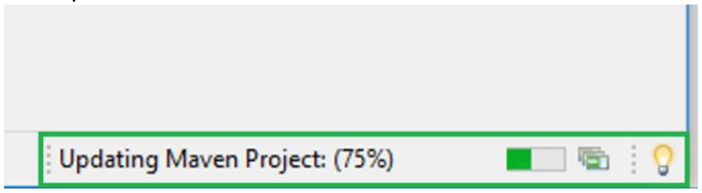
```
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-configuration-processor</artifactId>
                        <optional>true</optional>
                </dependency>
```

Either way the dependency should look like the following in the file:



```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
            <optional>true</optional>
        </dependency>
```

The project might need to be updated.
Right click on project ➔ “Maven” ➔ “Update Project…” ➔ “OK”
The update can take a while. The status is shown in the bottom right corner of the IDE.



Updating Maven Project: (75%)

Create Class: FileSystemStorageService

This class was not defined on the website and was taken from the GitHub Repro. This class is a major omission from the website since it implements the StorageService interface.

Create a class under package “com.example.uploadingfiles.storage” named: “FileSystemStorageService” which implement the StorageService interface.

Copy and replace the code shell with the following code:

```
package com.example.uploadingfiles.storage;

import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.stream.Stream;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.core.io.UrlResource;
import org.springframework.stereotype.Service;
import org.springframework.util.FileSystemUtils;
import org.springframework.web.multipart.MultipartFile;

@Service
public class FileSystemStorageService implements StorageService {

	private final Path rootLocation;

	@Autowired
	public FileSystemStorageService(StorageProperties properties) {
		this.rootLocation = Paths.get(properties.getLocation());
	}

	@Override
	public void store(MultipartFile file) {
		try {
			if (file.isEmpty()) {
				throw new StorageException("Failed to store empty file.");
			}
			Path destinationFile = this.rootLocation.resolve(
					Paths.get(file.getOriginalFilename()))
					.normalize().toAbsolutePath();
			if (!destinationFile.getParent().equals(this.rootLocation.toAbsolutePath())) {
				// This is a security check
				throw new StorageException(
						"Cannot store file outside current directory.");
			}
			try (InputStream inputStream = file.getInputStream()) {
				Files.copy(inputStream, destinationFile,
					StandardCopyOption.REPLACE_EXISTING);
			}
		}
		catch (IOException e) {
			throw new StorageException("Failed to store file.", e);
		}
	}

	@Override
	public Stream<Path> loadAll() {
		try {
			return Files.walk(this.rootLocation, 1)
				.filter(path -> !path.equals(this.rootLocation))
				.map(this.rootLocation::relativize);
		}
		catch (IOException e) {
			throw new StorageException("Failed to read stored files", e);
		}

	}

	@Override
	public Path load(String filename) {
		return rootLocation.resolve(filename);
	}

	@Override
	public Resource loadAsResource(String filename) {
		try {
			Path file = load(filename);
			Resource resource = new UrlResource(file.toUri());
			if (resource.exists() || resource.isReadable()) {
				return resource;
			}
			else {
				throw new StorageFileNotFoundException(
						"Could not read file: " + filename);

			}
		}
		catch (MalformedURLException e) {
			throw new StorageFileNotFoundException("Could not read file: " + filename, e);
		}
	}
	@Override
	public void deleteAll() {
		FileSystemUtils.deleteRecursively(rootLocation.toFile());
	}

	@Override
	public void init() {
		try {
			Files.createDirectories(rootLocation);
		}
		catch (IOException e) {
			throw new StorageException("Could not initialize storage", e);
		}
	}
}
```

Create File Uploader Class: FileUploadController

As mentioned in the Project Discussion the FileUploadController class from the website threw a startup error. Replacing the code from the repository corrected the error and after completing the next section "Update Main Application Class: UploadingFilesApplication" and this section with the code provided here, the program initially ran successfully.

Under package "com.example.uploadingfiles" create class: "FileUploadController".

Copy and replace the code shell with the following code:

```
package com.example.uploadingfiles;

import java.io.IOException;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.ResponseBody;
import org.springframework.web.multipart.MultipartFile;
import org.springframework.web.servlet.mvc.method.annotation.MvcUriComponentsBuilder;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.example.uploadingfiles.storage.StorageFileNotFoundException;
import com.example.uploadingfiles.storage.StorageService;

@Controller
public class FileUploadController {

	private final StorageService storageService;

	@Autowired
	public FileUploadController(StorageService storageService) {
		this.storageService = storageService;
	}

	@GetMapping("/")
	public String listUploadedFiles(Model model) throws IOException {

		model.addAttribute("files", storageService.loadAll().map(
				path -> MvcUriComponentsBuilder.fromMethodName(FileUploadController.class,
						"serveFile",
path.getFileName().toString()).build().toUri().toString())
				.collect(Collectors.toList()));

		return "uploadForm";
	}

	@GetMapping("/files/{filename:.+}")
	@ResponseBody
	public ResponseEntity<Resource> serveFile(@PathVariable String filename) {

		Resource file = storageService.loadAsResource(filename);
		return ResponseEntity.ok().header(HttpHeaders.CONTENT_DISPOSITION,
				"attachment; filename=\"" + file.getFilename() + "\"").body(file);
	}

	@PostMapping("/")
	public String handleFileUpload(@RequestParam("file") MultipartFile file,
			RedirectAttributes redirectAttributes) {

		storageService.store(file);
		redirectAttributes.addFlashAttribute("message",
				"You successfully uploaded " + file.getOriginalFilename() + "!");

		return "redirect:/";
	}

	@ExceptionHandler(StorageFileNotFoundException.class)
	public ResponseEntity<?> handleStorageFileNotFound(StorageFileNotFoundException exc) {
		return ResponseEntity.notFound().build();
	}

}
```

Update Main Application Class: UploadingFilesApplication

As mentioned previously we waited to update the main class until any reference dependencies are created. Here we add imports and a method to the class. Further details from the website describe these changes as follows:

> You want a target folder to which to upload files, so you need to enhance the basic `UploadingFilesApplication` class that Spring Initializr created and add a Boot `CommandLineRunner` to delete and re-create that folder at startup. The following listing (from `src/main/java/com/example/uploadingfiles/UploadingFilesApplication.java`) shows how to do so: – "Uploading Files"

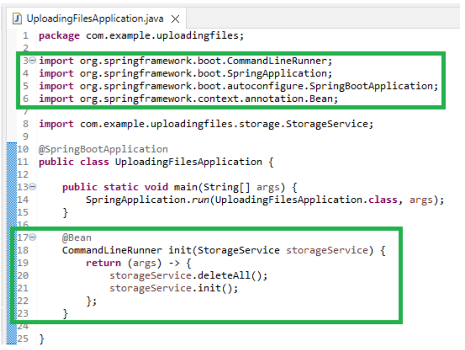
Add these imports:

```
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
```

Add the method:

```
	@Bean
	CommandLineRunner init(StorageService storageService) {
		return (args) -> {
			storageService.deleteAll();
			storageService.init();
		};
	}
```

The class should now look similar to the following:



UploadingFilesApplication.java

```
package com.example.uploadingfiles;

import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import com.example.uploadingfiles.storage.StorageService;

@SpringBootApplication
public class UploadingFilesApplication {

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

    @Bean
    CommandLineRunner init(StorageService storageService) {
        return (args) -> {
            storageService.deleteAll();
            storageService.init();
        };
    }

}
```

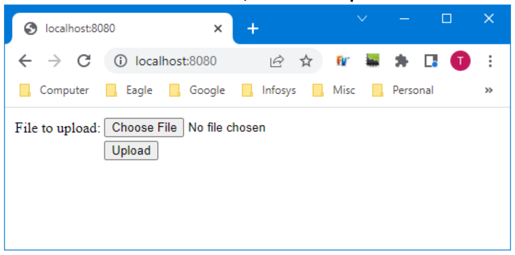
Initial Test of the Application

Start the application inside the STS 4 IDE as a spring boot application.
Right click inside UploadingFilesApplication class → "Run As" → "Spring Boot App"

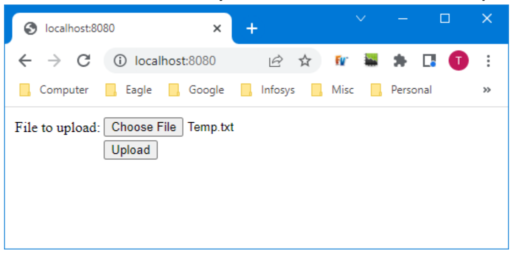
Enter the URL in a browser window:

```
http://localhost:8080
```

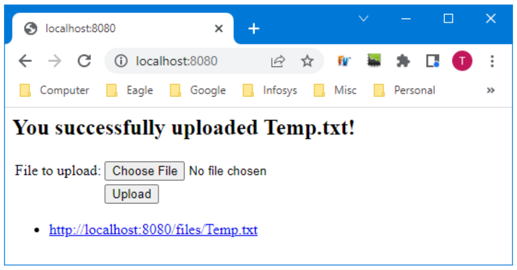
Select "Choose File", in the open file window choose a text file on your machine. Click Upload



File to upload: No file chosen



File to upload: Temp.txt



You successfully uploaded Temp.txt!

File to upload: No file chosen

- http://localhost:8080/files/Temp.txt

Other Testing of the Application

The website "Uploading Files" has a FileUploadTests class and the "GitHub Repro" has another class FileUploadIntegrationTests that introduces JUnit and Mockito concepts and dependencies. The website and repository does not provide the required details to correctly implement JUnit and Mockito concepts for testing. These concepts are important for developing and testing applications and will be covered in another project.

Project Conclusion

This concludes the project to upload a file from local machine to the application created. The project based on the "Uploading Files" website had various issues that this document resolved. The two testing concepts JUnit and Mockito are left for another project and document.