# Datei Up- und Download

**Beschreibung**

Dieses Dokument ergänzt einen Artikel, indem das Datei Up- und Download beschrieben ist. Ziel ist eine kleine Applikation, die das Grundkonzept des Datei Up- bzw. Downloads mit seinen Elementen zeigt.

**Inhaltsverzeichnis**

# 1 File Up- und Download

## 1.1 Einführung

Der folgende Artikel beschreibt Schritt für Schritt die Elemente, die für den File Up- und Download braucht. Die Implementierung basiert auf einer SOA-Architektur mit React im Frontend- und Spring Boot als Backend-Applikation.
https://rieckpil.de/howto-up-and-download-files-with-react-and-spring-boot

Für die folgenden Aufgaben verzichten wir auf die Implementierung einer Datenbank und lesen die Dateien direkt aus einem Ordner.

## 1.2 Frontend mit React

State Variablen:
```
const [error, setError] = useState("")
const [message, setMessage] = useState("")
const [file, setFile] = useState("")
```

Formular:
```
<div>
  <h2>Upload a file</h2>
  <h4 style={{color: 'red'}}>{error}</h4>
  <h4 style={{color: 'green'}}>{message}</h4>
  <input onChange={onFileChange} type="file"></input>
  <button onClick={uploadFile}>Upload</button>
</div>
```

Funktionen:
```
const onFileChange = (event) => {
  setFile(event.target.files[0])
}

const uploadFile = (event) => {
  setError('')
  setMessage('');
  ...
  let data = new FormData();
  data.append('file', file);
  data.append('name', file.name);

  fetch('http://localhost:8080/api/files', {
    method: 'POST',
    body: data
  }).then(response => {
    setError('')
    setMessage('Sucessfully uploaded file');
  }).catch(err => {
    setError(err);
  });
}
```

## 1.3 Backend mit Spring Boot

Datenstruktur für eine Datei:

```
@Data
@AllArgsConstructor
@ToString
public class MyFile {
    private String name;
    private long size;
    private String contentType;
    private byte[] data;
}
```

Rest-Controller (Endpoints):

```
@RestController
@CrossOrigin("http://localhost:3000")
@RequestMapping("api/files")
public class FilesRestController {

  @GetMapping()
  public ResponseEntity<List<MyFile>> getFiles() {
    return ResponseEntity.ok(new ArrayList());
  }

  @PostMapping()
  public ResponseEntity<Void> uploadNewFile(
         @NotNull @RequestParam("file") MultipartFile multipartFile)
         throws IOException {
    ...
  }
}
```

Einstellungen application.properties:

```
spring.servlet.multipart.max-file-size=2MB
spring.servlet.multipart.max-request-size=10MB
```

### *1.3.1 Ordner lesen und Dateien auflisten*

Hier wird die Klasse File verwendet, um alle Dateien und Ordner im Quellverzeichnis zu sammeln und dann die Methode isDirectory(), um zu prüfen, ob es sich um eine Datei oder einen Ordner handelt.

```
File folder = new File("/home/...");
for (File file : folder.listFiles()) {
  if (!file.isDirectory()) {
    System.out.println(file.getName());
  }
}
```

# 2 Weiterführende Aufgaben

## 2.1 Upload grosser Dateien

Multipart Upload

## 2.2 Drag and Drop Feld für Datei-Upload

## 2.3 Streaming von Videodateien