

Spring Boot Generic File Service

Es ist schon erstaunlich wie man auch mit Spring Boot REST Services generisch und allgemein basierend auf der Servlet Schnittstelle programmieren kann. Das folgende Programmscript zeigt den REST Controller GenericFileService mit dem wir anhand des Path URL Parameters eine lokale Datei laden und Buffered an den HTTP Client transferieren.

```
package ch.std.fileservice.rest;<br>import java.io.FileInputStream;<br>import java.io.OutputStream;<br>import java.net.URLConnection;<br>import javax.activation.MimetypesFileTypeMap;<br>import javax.servlet.http.HttpServletRequest;<br>import javax.servlet.http.HttpServletResponse;<br>import org.springframework.web.bind.annotation.GetMapping;<br>import org.springframework.web.bind.annotation.RequestMapping;<br>import org.springframework.web.bind.annotation.RestController;<br>@RestController<br>@RequestMapping(value = "/rest/file")<br>public class GenericFileService {<br>    public GenericFileService() {<br>        super();<br>    }<br>    @GetMapping<br>    public void get(HttpServletRequest httpServletRequest, HttpServletResponse httpServletResponse) throws<br>        Exception {<br>        String path = httpServletRequest.getParameter("path");<br>        if (path == null) {<br>            httpServletResponse.setContentType("text/html;<br>        charset=UTF-8");<br>            httpServletResponse.getWriter().print("path is<br>        missing");<br>            return;<br>        }<br>        String mimeType =<br>        URLConnection.guessContentTypeFromName(path);<br>        if (mimeType == null) {<br>            MimetypesFileTypeMap mimeTypeMap = new MimetypesFileTypeMap();<br>            mimeType =<br>            mimeTypeMap.getContentType(path);<br>            httpServletResponse.setContentType(mimeType);<br>            httpServletResponse.setHeader("Transfer-Encoding", "chunked");<br>            OutputStream os = httpServletResponse.getOutputStream();<br>            byte[] buffer = new byte[1<br>            &lt;&lt; 12];<br>            int bytesRead = -1;<br>            try (FileInputStream fis = new<br>            FileInputStream(path)) {<br>                while ((bytesRead = fis.read(buffer)) > 0) {<br>                    os.write(buffer, 0, bytesRead);<br>                    os.flush();<br>                }<br>            } catch (Exception<br>            e) {<br>                httpServletResponse.setContentType("text/html; charset=UTF-8");<br>                os.write(("path " + path + " does not<br>            exist").getBytes(StandardCharsets.UTF_8));<br>                os.flush();<br>            }<br>        }<br>        return;<br>    }<br>}<br>Das folgende Listing zeigt die Spring Boot Application:<br>package ch.std.fileservice;<br>import org.springframework.boot.SpringApplication;<br>import org.springframework.boot.autoconfigure.SpringBootApplication;<br>@SpringBootApplication<br>public class GenericFileServiceApplication {<br>    public static void main(String[] args) {<br>        SpringApplication.run(GenericFileServiceApplication.class, args);<br>    }<br>}<br>
```

Nach dem Start der Spring Boot Applikation ist der File Service aktiv und kann je nach Konfiguration wie folgt adressiert werden: `http://localhost:8080/rest/file?path=/mypath/myfile.txt` Sofern das File existiert wird es zum Client gestreamt.

Das folgende Listing zeigt den dazu passenden Spring Boot File Service Client entwickelt als Command Line Applikation:

```
package ch.std.fileservice.client;<br>import java.io.FileOutputStream;<br>import java.net.URL;<br>import java.util.Arrays;<br>import org.springframework.boot.ApplicationArguments;<br>import org.springframework.boot.ApplicationRunner;<br>import org.springframework.boot.SpringApplication;<br>import org.springframework.boot.WebApplicationType;<br>import org.springframework.boot.autoconfigure.SpringBootApplication;<br>import org.springframework.core.io.buffer.DataBufferUtils;<br>import org.springframework.http.HttpEntity;<br>import org.springframework.http.HttpHeaders;<br>import org.springframework.http.HttpMethod;<br>import org.springframework.http.MediaType;<br>import org.springframework.http.ResponseEntity;<br>
```

```
org.springframework.http.converter.ByteArrayHttpMessageConverter;import
org.springframework.web.client.RestTemplate;@SpringBootApplication;public class
GenericFileServiceClient implements ApplicationRunner {    public static void main(String[]
args) throws Exception {        SpringApplication app = new
SpringApplication(GenericFileServiceClient.class);
app.setWebApplicationType(WebApplicationType.NONE);    app.run(args);    }
@Override    public void run(ApplicationArguments args) throws Exception {        URL url =
null;        try {            url = new URL(args.getOptionValues("&#34;url&#34;).get(0));
        } catch (Exception e) {            System.err.println("&#34;missing --url option
argument&#34;);            this.help();            return;        }        String out =
null;        try {            out = args.getOptionValues("&#34;out&#34;).get(0);        } catch
(Exception e) {            RestTemplate restTemplate = new RestTemplate();
restTemplate.getMessageConverters().add(new ByteArrayHttpMessageConverter());
HttpHeaders headers = new HttpHeaders();        HttpEntity entity = new
HttpEntity(headers);        ResponseEntity response = restTemplate.exchange(url.toString(),
HttpMethod.GET, entity, byte[].class, "&#34;1&#34;");        if (out != null) {            try
(FileOutputStream fos = new FileOutputStream(out)) {                fos.write(response.getBody());
            }            System.out.println("&#34;result written to file
&#34; + out);        }        private void help() {            System.out.println("&#34;usage java -jar
genericfileclient-0.0.1-SNAPSHOT.jar --url= --out=&#34;);
System.out.println("&#34;example:&#34;);            System.out.println("&#34;java -jar
genericfileclient-0.0.1-SNAPSHOT.jar --url=http://localhost:8080/rest/file?path=in/bigimage.jpg
--out=out/bigimage.jpg&#34;);        }    }Das File bigimage.jpg kann ersetzt werden durch
eine real existierende Datei analog kann der Output Pfad angepasst werden.
```

Client und Service sind am besten in 2 separaten Spring Boot Projekten zu programmieren z.B. mit der Eclipse IDE.

Feedback

War dieser Blog für Sie wertvoll. Wir danken für jede Anregung und Feedback

Kontakt

Simtech AG
Finkenweg 23
3110 Münsingen
Schweiz

Impressum

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsausschluss.
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

<https://www.simtech-ag.ch/setheader>