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

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.txtSofern 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;
import java.io.FileOutputStream;
import java.net.URL;
import java.util.Arrays;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.WebApplicationType;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.core.io.buffer.DataBufferUtils;
import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import

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("url").get(0));
  } catch (Exception e) {
  System.err.println("missing --url option argument");
  this.help();
  return;
  }
  String out = null;
  try {
  out = args.getOptionValues("out").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, "1");
  if (out != null) {
  try (FileOutputStream fos = new FileOutputStream(out)) {
 fos.write(response.getBody());
  }
  System.out.println("result written to file " + out);
  }
  private void help() {
  System.out.println("usage java -jar genericfileclient-0.0.1-SNAPSHOT.jar --url= --out=");
 System.out.println("example:");
  System.out.println("java -jar genericfileclient-0.0.1-SNAPSHOT.jar --url=http://localhost:8080/rest/file?path=in/bigimage.jpg --out=out/bigimage.jpg");
  }
}
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

Copyright

https://www.simtech-ag.ch/getMessageConverters