Übung 4: SOAP Implementierung

# Grundidee

In den vorherigen Aufgaben habe ich feststellen müssen, dass ein objektorientierter und serialisierbarer Ansatz in vielen Fällen hilfreich ist, nicht aber service- oder ressourcenorientiert Technologien wie SOAP und REST sauber abdecken kann (Ausser man investiert viel Zeit in eine CRUD Logik).

Für die vierte Aufgabe habe ich deshalb einfach einen Bankentreiber samt Ressourcenverwaltung neu implementiert (Überraschenderweise war der Zeitaufwand recht gering).

# Implementierung

Der clientseitige Driver verwendet eine eigenständige Bankenimplementierung:

|  |
| --- |
| package ch.fhnw.vesys.clientrest;  import ch.fhnw.vesys.shared.api.\*;  import ch.fhnw.vesys.shared.local.LocalDriver;  import javax.ws.rs.client.Client;  import javax.ws.rs.client.ClientBuilder;  import javax.ws.rs.client.Entity;  import javax.ws.rs.client.WebTarget;  import javax.ws.rs.core.Form;  import javax.ws.rs.core.MediaType;  import javax.ws.rs.core.Response;  import java.io.IOException;  import java.net.URI;  import java.security.InvalidParameterException;  import java.util.Arrays;  import java.util.HashSet;  import java.util.Set;  public class RestDriver implements BankDriver {  private RestBank bank;  @Override  public void connect(String[] args) throws IOException {  try {  URI uri = new URI("http://" + args[0] + ":" + Integer.parseInt(args[1]) + "/accounts");  Client client = ClientBuilder.newClient();  WebTarget target = client.target(uri);  bank = new RestBank(target);  System.out.println("Rest driver connected");  } catch (Exception exception) {  throw new IOException("Unable to parse the given parameters");  }  }  @Override  public void disconnect() throws IOException {  System.out.println("Rest driver disconnected");  }  @Override  public Bank getBank() {  return bank;  }  private class RestBank implements Bank {  private final WebTarget target;  RestBank(WebTarget target) {  this.target = target;  }  @Override  public String createAccount(String owner) throws IOException {  Form form = new Form();  form.param("owner", owner);  Response response = target.request().post(Entity.form(form));  if (response.getStatus() == Response.Status.CREATED.getStatusCode()) {  return response.readEntity(String.class);  } else {  throw new IOException("Unable to create the account");  }  }  @Override  public boolean closeAccount(String number) throws IOException {  if (number == null || number.isEmpty()) {  return false;  }  Response response = target.path("/" + number).request().accept(MediaType.APPLICATION\_JSON).delete();  return response.getStatus() == Response.Status.OK.getStatusCode();  }  @Override  public Set<String> getAccountNumbers() throws IOException {  Response response = target.request().accept(MediaType.APPLICATION\_JSON).get();  if (response.getStatus() == Response.Status.OK.getStatusCode()) {  return new HashSet<>(Arrays.asList(response.readEntity(String[].class)));  } else {  throw new IOException("Unable to read the accounts");  }  }  @Override  public Account getAccount(String number) throws IOException {  if (number == null || number.isEmpty()) {  return null;  }  Response response = target.path("/" + number).request().accept(MediaType.APPLICATION\_JSON).get();  if (response.getStatus() == Response.Status.OK.getStatusCode()) {  LocalDriver.LocalAccount account = response.readEntity(LocalDriver.LocalAccount.class);  return new RestAccount(target, account.getNumber(), account.getOwner());  } else {  return null;  }  /\*LocalDriver.LocalAccount account = target.path("/" + number).request().accept(MediaType.APPLICATION\_JSON).get(LocalDriver.LocalAccount.class);  if (account != null) {  return new RestAccount(target, account.getNumber(), account.getOwner());  } else {  return null;  }\*/  }  @Override  public void transfer(Account a, Account b, double amount) throws IOException, IllegalArgumentException, OverdrawException, InactiveException {  RestAccount resta = new RestAccount(target, a.getNumber(), a.getOwner());  RestAccount restb = new RestAccount(target, b.getNumber(), b.getOwner());  if (!resta.isActive() || !restb.isActive()) {  throw new InactiveException();  }  resta.withdraw(amount);  restb.deposit(amount);  }  }  private class RestAccount implements Account {  private final String number;  private final String owner;  private final WebTarget target;  RestAccount(WebTarget target, String number, String owner) {  this.target = target;  this.number = number;  this.owner = owner;  }  @Override  public String getNumber() throws IOException {  return number;  }  @Override  public String getOwner() throws IOException {  return owner;  }  @Override  public boolean isActive() throws IOException {  Response response = target.path("/" + number).request().head();  return response.getStatus() != Response.Status.NOT\_FOUND.getStatusCode() && response.getStatus() == Response.Status.OK.getStatusCode();  }  @Override  public void deposit(double amount) throws IOException, IllegalArgumentException, InactiveException {  if (amount < 0) {  throw new IllegalArgumentException();  }  Form form = new Form();  form.param("amount", Double.toString(amount));  Response response = target.path("/" + number).request().put(Entity.form(form));  if (response.getStatus() == Response.Status.NOT\_FOUND.getStatusCode()) {  throw new InactiveException();  } else if (response.getStatus() == Response.Status.INTERNAL\_SERVER\_ERROR.getStatusCode()) {  throw new InactiveException();  }  }  @Override  public void withdraw(double amount) throws IOException, IllegalArgumentException, OverdrawException, InactiveException {  if (amount < 0) {  throw new IllegalArgumentException();  }  if (getBalance() - amount < 0) {  throw new OverdrawException();  }  Form form = new Form();  form.param("amount", Double.toString(-amount));  Response response = target.path("/" + number).request().put(Entity.form(form));  if (response.getStatus() == Response.Status.NOT\_FOUND.getStatusCode()) {  throw new InactiveException();  } else if (response.getStatus() == Response.Status.INTERNAL\_SERVER\_ERROR.getStatusCode()) {  throw new OverdrawException();  }  }  @Override  public double getBalance() throws IOException {  LocalDriver.LocalAccount account = target.path("/" + number).request().accept(MediaType.APPLICATION\_JSON).get(LocalDriver.LocalAccount.class);  return account.getBalance();  }  private void setBalance(double newamout) throws InactiveException {  Form form = new Form();  form.param("amount", Double.toString(newamout));  Response response = target.path("/" + number).request().post(Entity.form(form));  if (response.getStatus() == Response.Status.NOT\_FOUND.getStatusCode()) {  throw new InvalidParameterException();  } else if (response.getStatus() == Response.Status.GONE.getStatusCode()) {  throw new InactiveException();  }  }  }  } |

Serverseitig wird Jetty als Servletserver verwendet (Dieser stellt nicht alle EE Features zur Verfügung, da wir diese schlichtwegs nicht brauchen).

Der Server wird durch folgende Klasse gestartet:

|  |
| --- |
| package ch.fhnw.vesys.serverrest;  import org.eclipse.jetty.server.Server;  import org.eclipse.jetty.servlet.ServletContextHandler;  import org.eclipse.jetty.servlet.ServletHolder;  import org.glassfish.jersey.servlet.ServletContainer;  class RestServer {  private RestServer(int port) throws Exception {  RestConfig config = new RestConfig();  ServletHolder servletholder = new ServletHolder(new ServletContainer(config));  Server server = new Server(port);  ServletContextHandler context = new ServletContextHandler(server, "/\*");  context.addServlet(servletholder, "/\*");  try {  server.start();  server.join();  } finally {  server.destroy();  }  }  public static void main(String[] args) throws Exception {  new RestServer(1234);  }  } |

Die Ressourcen werden durch folgende Klasse registriert:

|  |
| --- |
| package ch.fhnw.vesys.serverrest;  import ch.fhnw.vesys.shared.api.BankDriver;  import ch.fhnw.vesys.shared.local.LocalDriver;  import org.glassfish.jersey.server.ResourceConfig;  class RestConfig extends ResourceConfig {  RestConfig() {  BankDriver bankdriver = new LocalDriver();  register(new RestController(bankdriver));  }  } |

Die Ressource Accounts unterstützt die vom Dozenten vorgeschlagenen Operationen, weshalb ich nicht noch einmal darauf eingehen möchte:

|  |
| --- |
| package ch.fhnw.vesys.serverrest;  import ch.fhnw.vesys.shared.api.Account;  import ch.fhnw.vesys.shared.api.BankDriver;  import javax.inject.Singleton;  import javax.ws.rs.\*;  import javax.ws.rs.core.\*;  @Singleton  @Path("/accounts")  @Produces({MediaType.APPLICATION\_JSON})  public class RestController {  private final BankDriver bankdriver;  RestController(BankDriver bankdriver) {  this.bankdriver = bankdriver;  }  @GET  @Consumes({MediaType.APPLICATION\_JSON})  public Response getAccounts() throws Exception {  String[] numbers = bankdriver.getBank().getAccountNumbers().toArray(new String[]{});  return Response.ok(numbers).build();  }  @POST  @Consumes({MediaType.APPLICATION\_FORM\_URLENCODED})  public Response createAccount(@Context UriInfo uriinfo, @FormParam("owner") String owner) throws Exception {  String number = bankdriver.getBank().createAccount(owner);  UriBuilder builder = uriinfo.getAbsolutePathBuilder();  builder.path(number);  return Response.created(builder.build()).entity(number).build();  }  @GET  @Path("{number}")  @Consumes({MediaType.APPLICATION\_JSON})  public Response getAccount(@PathParam("number") String number) throws Exception {  Account account = bankdriver.getBank().getAccount(number);  if (account == null) {  throw new NotFoundException("Unable to find the account");  }  return Response.ok(account).build();  }  @PUT  @Path("{number}")  @Consumes({MediaType.APPLICATION\_FORM\_URLENCODED})  public Response updateAccount(@PathParam("number") String number, @FormParam("amount") double amount) throws Exception {  Account account = bankdriver.getBank().getAccount(number);  if (account == null) {  throw new NotFoundException("Unable to find the account");  }  if (amount < 0 && account.getBalance() - amount < 0) {  return Response.serverError().build();  }  try {  if (amount > 0) {  account.deposit(amount);  } else if (amount < 0) {  account.withdraw(Math.abs(amount));  }  } catch (Exception exception) {  return Response.serverError().build();  }  return Response.ok().build();  }  @DELETE  @Path("{number}")  @Consumes({MediaType.APPLICATION\_JSON})  public Response deleteAccount(@PathParam("number") String number) throws Exception {  Account account = bankdriver.getBank().getAccount(number);  if (account == null) {  throw new NotFoundException("Unable to find the account");  }  if (bankdriver.getBank().closeAccount(number)) {  return Response.ok().build();  } else {  return Response.serverError().build();  }  }  @HEAD  @Path("{number}")  @Consumes({MediaType.APPLICATION\_JSON})  public Response checkAccount(@PathParam("number") String number) throws Exception {  Account account = bankdriver.getBank().getAccount(number);  if (account == null) {  throw new NotFoundException("Unable to find the account");  }  if (account.isActive()) {  return Response.ok().build();  } else {  return Response.noContent().build();  }  }  } |

Der restliche Code ist dem Projekt zu entnehmen.

# Verbesserungsmöglichkeiten

Da ich beruflich mit Spring Boot arbeite, war für mich der Vergleich mit JAX-RS und Jersey sehr interessant. Leider überzeugt mich das Framework in der jetzigen Konfiguration nicht wirklich, da ich Features Dependency Injection, besseres Marshalling und Exceptionhandling vermisse, sprich nicht out of the box anwenden kann.

Für die Zukunft wäre auf jeden Fall ein sauberes Exceptionhandling und das Erstellen einer Transfer Ressource wichtig, da momentan viel clientseitige Logik existiert, welche eigentlich vom Server gehandhabt werden sollte (Es muss im Client geprüft haben, ob der Absender bei einem Transfer genügend Geld hat).

# Source Code

Der Source Code ist hier zu finden: <https://github.com/swaechter/fhnw/tree/master/Module/vesys/bank>