package info.smartkit.blockchain.bigchaindb;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.builder.SpringApplicationBuilder;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.context.annotation.PropertySource;

import org.springframework.context.annotation.PropertySources;

//@PropertySources({ @PropertySource(value = "classpath:application-${spring.profiles.active}.properties") })

//@Configuration

//@EnableAutoConfiguration

//@ComponentScan

@SpringBootApplication

public class Application {

private static Logger LOG = LogManager.getLogger(Application.class);

//

private static Class<Application> applicationClass = Application.class;

//

protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {

//

return application.sources(applicationClass);

}

public static void main(String[] args) {

SpringApplication.run(Application.class, args);

}

}

package crispy\_octo\_moo.aop;

import info.smartkit.blockchain.bigchaindb.consts.Constants;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.AfterThrowing;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Pointcut;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.core.env.Environment;

import java.util.Arrays;

/\*\*

\* Aspect for logging execution of service and repository Spring components.

\*/

@Aspect

public class LoggingAspect {

private final Logger log = LoggerFactory.getLogger(this.getClass());

@Autowired

private Environment env;

@Pointcut("within(info.smartkit.blockchain.bigchaindb.services..\*) || within(info.smartkit.blockchain.bigchaindb.controllers..\*)")

public void loggingPointcut() {}

@AfterThrowing(pointcut = "loggingPointcut()", throwing = "e")

public void logAfterThrowing(JoinPoint joinPoint, Throwable e) {

if (env.acceptsProfiles(Constants.SPRING\_PROFILE\_DEVELOPMENT)) {

log.error("Exception in {}.{}() with cause = {}", joinPoint.getSignature().getDeclaringTypeName(),

joinPoint.getSignature().getName(), e.getCause(), e);

} else {

log.error("Exception in {}.{}() with cause = {}", joinPoint.getSignature().getDeclaringTypeName(),

joinPoint.getSignature().getName(), e.getCause());

}

}

@Around("loggingPointcut()")

public Object logAround(ProceedingJoinPoint joinPoint) throws Throwable {

if (log.isDebugEnabled()) {

log.debug("Enter: {}.{}() with argument[s] = {}", joinPoint.getSignature().getDeclaringTypeName(),

joinPoint.getSignature().getName(), Arrays.toString(joinPoint.getArgs()));

}

try {

Object result = joinPoint.proceed();

if (log.isDebugEnabled()) {

log.debug("Exit: {}.{}() with result = {}", joinPoint.getSignature().getDeclaringTypeName(),

joinPoint.getSignature().getName(), result);

}

return result;

} catch (IllegalArgumentException e) {

log.error("Illegal argument: {} in {}.{}()", Arrays.toString(joinPoint.getArgs()),

joinPoint.getSignature().getDeclaringTypeName(), joinPoint.getSignature().getName());

throw e;

}

}

}

package info.smartkit.blockchain.bigchaindb.configs;

import java.io.IOException;

import org.apache.commons.lang.StringUtils;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import org.springframework.context.ApplicationContextInitializer;

import org.springframework.context.ConfigurableApplicationContext;

import org.springframework.core.env.ConfigurableEnvironment;

import org.springframework.core.io.support.ResourcePropertySource;

/\*\*

\* Register this with the DispatcherServlet in a ServletInitializer class like:

\* dispatcherServlet.setContextInitializers(new PropertiesInitializer());

\*/

public class PropertiesInitializer implements ApplicationContextInitializer<ConfigurableApplicationContext> {

private static final Logger LOG = LogManager.getLogger(PropertiesInitializer.class);

/\*\*

\* Runs as appInitializer so properties are wired before spring beans

\*/

@Override

public void initialize(ConfigurableApplicationContext applicationContext) {

ConfigurableEnvironment env = applicationContext.getEnvironment();

String[] activeProfiles = getActiveProfiles(env);

for (String profileName : activeProfiles) {

LOG.info("Loading properties for Spring Active Profile: {}", profileName);

try {

ResourcePropertySource propertySource =

new ResourcePropertySource(profileName + "EnvProperties", "classpath:application-" + profileName

+ ".properties");

env.getPropertySources().addLast(propertySource);

LOG.debug("propertySource:" + propertySource.toString());

// Work-flow setting initialization here.

// TODO: @see https://github.com/EsotericSoftware/yamlbeans to replace this staff.

// MqttSettings.setUri((String) propertySource.getProperty("mqtt.uri"));

// //

// ThriftSettings.setIp((String) propertySource.getProperty("thrift.ip"));

// ThriftSettings.setPort(Integer.valueOf((String) propertySource.getProperty("thrift.port")));

// //

// ServerSetting.setPort(Integer.valueOf((String) propertySource.getProperty("server.port")));

// ServerSetting.setContextPath((String) propertySource.getProperty("server.contextPath"));

// //

//// System.out.println((String) propertySource.getProperty("imageStore.local"));

//// System.out.println((String) propertySource.getProperty("imageStore.remote"));

// ImageSettings.setStoreLocalPath((String) propertySource.getProperty("image.store.local"));

// ImageSettings.setStoreRemoteUrl((String) propertySource.getProperty("image.store.remote"));

} catch (IOException e) {

LOG.error("ERROR during environment properties setup - TRYING TO LOAD: " + profileName, e);

// Okay to silently fail here, as we might have profiles that do

// not have properties files (like dev1, dev2, etc)

}

}

}

/\*\*

\* Returns either the ActiveProfiles, or if empty, then the DefaultProfiles from Spring

\*/

protected String[] getActiveProfiles(ConfigurableEnvironment env) {

String[] activeProfiles = env.getActiveProfiles();

if (activeProfiles.length > 0) {

LOG.info("Using registered Spring Active Profiles: {}", StringUtils.join(activeProfiles, ", "));

return activeProfiles;

}

String[] defaultProfiles = env.getDefaultProfiles();

LOG.info("No Active Profiles found, using Spring Default Profiles: {}", StringUtils.join(defaultProfiles, ", "));

return defaultProfiles;

}

}

package info.smartkit.blockchain.bigchaindb.configs;

import java.io.IOException;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Component;

/\*\*

\* @author yangboz

\* @see https://spring.io/guides/gs/rest-service-cors/

\*/

@Component

public class SimpleCORSFilter implements Filter {

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException {

HttpServletResponse response = (HttpServletResponse) res;

response.setHeader("Access-Control-Allow-Origin", "\*");

response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS, PUT, DELETE, PATCH");

response.setHeader("Access-Control-Max-Age", "3600");

response.setHeader("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");

chain.doFilter(req, res);

}

public void init(FilterConfig filterConfig) {}

public void destroy() {}

}

package info.smartkit.blockchain.bigchaindb.configs;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Scope;

import org.springframework.context.annotation.ScopedProxyMode;

import org.springframework.core.env.Environment;

import org.springframework.social.config.annotation.EnableSocial;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionFactoryLocator;

import org.springframework.social.connect.ConnectionRepository;

import org.springframework.social.connect.support.ConnectionFactoryRegistry;

import org.springframework.social.linkedin.api.LinkedIn;

import org.springframework.social.linkedin.api.impl.LinkedInTemplate;

import org.springframework.social.linkedin.connect.LinkedInConnectionFactory;

//import org.springframework.social.facebook.api.impl.FacebookTemplate;

/\*\*

\* Created by yangboz on 15/9/20.

\*/

@Configuration

@EnableSocial

public class SocialApiConfig {

private static String sqootApiKey;

private static String sqootApiSecret;

//

@Autowired

Environment environment;

@Bean

public ConnectionFactoryLocator connectionFactoryLocator() {

ConnectionFactoryRegistry registry = new ConnectionFactoryRegistry();

registry.addConnectionFactory(new LinkedInConnectionFactory(

environment.getProperty("spring.social.linkedin.consumerKey"),

environment.getProperty("spring.social.linkedin.consumerSecret")));

return registry;

}

@Bean

@Scope(value = "request", proxyMode = ScopedProxyMode.INTERFACES)

public LinkedIn LinkedIn(ConnectionRepository connectionRepository) {

Connection<LinkedIn> linkedIn = connectionRepository.findPrimaryConnection(LinkedIn.class);

return linkedIn != null ? linkedIn.getApi() : new LinkedInTemplate("");//EJVLzdPSPeTqy2r2fVQapO7BSEzFg65MQaIF

}

}

package info.smartkit.blockchain.bigchaindb.configs;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration

public class StaticResourceConfiguration extends WebMvcConfigurerAdapter {

private static final String[] CLASSPATH\_RESOURCE\_LOCATIONS = {

"classpath:/META-INF/resources/", "classpath:/resources/",

"classpath:/static/", "classpath:/public/"};

/\*\*

\* Add our static resources folder mapping.

\*/

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

//

registry.addResourceHandler("/\*\*").addResourceLocations(

CLASSPATH\_RESOURCE\_LOCATIONS);

//

// Activiti repository resources(diagram picture,process BPM files).

// registry.addResourceHandler("/repository/\*\*").addResourceLocations("classpath:/repository/");

// Jasper report

registry.addResourceHandler("/static/\*\*").addResourceLocations(

"classpath:/static/");

// registry.addResourceHandler("/reports/\*\*").addResourceLocations("classpath:/reports/");

//

super.addResourceHandlers(registry);

}

}

package info.smartkit.blockchain.bigchaindb.configs;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import springfox.documentation.builders.PathSelectors;

import springfox.documentation.builders.RequestHandlerSelectors;

import springfox.documentation.service.ApiInfo;

import springfox.documentation.spi.DocumentationType;

import springfox.documentation.spring.web.plugins.Docket;

import springfox.documentation.swagger2.annotations.EnableSwagger2;

import java.util.ArrayList;

/\*\*

\* The Class SwaggerConfig.

\*/

@Configuration

@EnableSwagger2

public class SwaggerConfig {

@Bean

public Docket api() {

return new Docket(DocumentationType.SWAGGER\_2)

.select()

.apis(RequestHandlerSelectors.basePackage("info.smartkit.blockchain.bigchaindb.controllers"))

// .paths(PathSelectors.ant("bigchaindb/\*"))

.paths(PathSelectors.any())

.build()

.apiInfo(apiInfo());

// .useDefaultResponseMessages(false)

// .globalResponseMessage(RequestMethod.GET,

// new ArrayList(new ResponseMessageBuilder()

// .code(500)

// .message("500 message")

// .responseModel(new ModelRef("Error"))

// .build(),

// new ResponseMessageBuilder()

// .code(403)

// .message("Forbidden!")

// .build()));

}

private ApiInfo apiInfo() {

ApiInfo apiInfo = new ApiInfo(

"BLOCKCHAIN.SMARTKIT.INFO REST API",

"SMARTKIT.INFO description of API.",

"API TOS",

"Terms of service",

"contact@smartkit.info",

"License of API",

"API license URL");

return apiInfo;

}

}

package info.smartkit.blockchain.bigchaindb.consts;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Application constants.

\*/

public final class Constants {

private Constants() {

}

public static final String SPRING\_PROFILE\_DEVELOPMENT = "dev";

public static final String SPRING\_PROFILE\_PRODUCTION = "prod";

public static final String SPRING\_PROFILE\_FAST = "fast";

public static final String SPRING\_PROFILE\_CLOUD = "cloud";

public static final String SYSTEM\_ACCOUNT = "system";

//Facebook related

// public static final String FB\_APP\_ID = "844627638977678";

// public static final String FB\_APP\_SECRET = "40c4065408a4878eaff77882947ee3ef";

// public static final String FB\_APP\_PERMISSIONS = "ads\_management,ads\_read,bookmarked,email,manage\_notifications,manage\_pages,publish\_actions,publish\_pages,read\_insights,read\_mailbox,read\_page\_mailboxes,read\_stream,rsvp\_event,tab\_added,user\_about\_me,user\_birthday,user\_education\_history,user\_events,user\_friends,user\_games\_activity,user\_groups,user\_hometown,user\_likes,user\_location,user\_managed\_groups,user\_photos,user\_posts,user\_relationship\_details,user\_relationships,user\_religion\_politics,user\_status,user\_tagged\_places,user\_videos,user\_website,user\_work\_history";

//Global variables;

}

package info.smartkit.blockchain.bigchaindb.controllers;

import com.bigchaindb.builders.BigchainDbConfigBuilder;

import com.bigchaindb.builders.BigchainDbTransactionBuilder;

import com.bigchaindb.constants.Operations;

import com.bigchaindb.model.FulFill;

import com.bigchaindb.model.GenericCallback;

import com.bigchaindb.model.MetaData;

import com.bigchaindb.model.Transaction;

import com.bigchaindb.util.Base58;

import info.smartkit.blockchain.bigchaindb.dto.JsonObject;

import info.smartkit.blockchain.bigchaindb.dto.JsonString;

import io.swagger.annotations.ApiOperation;

import net.i2p.crypto.eddsa.EdDSAPrivateKey;

import net.i2p.crypto.eddsa.EdDSAPublicKey;

import okhttp3.Response;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import org.springframework.web.bind.annotation.\*;

import java.io.IOException;

import java.security.KeyPair;

import java.util.Map;

import java.util.TreeMap;

/\*\*

\* The Class BigchainDbController.

\*/

@RestController

// @see: https://github.com/bigchaindb/java-bigchaindb-driver#api-wrappers

// @see: https://gist.github.com/innoprenuer/d4c6798fe5c0581c05a7e676e175e515

@RequestMapping(value = "v1/bigchaindb")

public class BigchainDbController {

//

private static Logger LOG = LogManager.getLogger(BigchainDbController.class);

@RequestMapping(method = RequestMethod.GET, value = "/config")

@ApiOperation(value = "Response a string describing bigchaindb info.")

// @ApiImplicitParams({@ApiImplicitParam(name="Authorization", value="Authorization DESCRIPTION")})

public @ResponseBody JsonString info() {

this.setConfig();

return new JsonString("v0.0.0");

}

//Transactions

@RequestMapping(method = RequestMethod.GET, value = "/transaction")

@ApiOperation(value = "Create new assets.")

// @ApiImplicitParams({@ApiImplicitParam(name="Authorization", value="Authorization DESCRIPTION")})

public @ResponseBody JsonString createTransaction() throws Exception {

Map<String, String> assetData = getAssetMap();

MetaData metaData = getMetaData();

KeyPair keys = getKeys();

String txId = doCreate(assetData, metaData, keys);

//create transfer metadata

MetaData transferMetadata = new MetaData();

transferMetadata.setMetaData("where is he now?", "Japan");

System.out.println("(\*) Transfer Metadata Prepared..");

//execute TRANSFER transaction on the CREATED asset

doTransfer(txId, transferMetadata, keys);

return new JsonString(txId);

}

//Outputs

//Assets

@RequestMapping(method = RequestMethod.POST, value = "/assets")

@ApiOperation(value = "Create new assets.")

// @ApiImplicitParams({@ApiImplicitParam(name="Authorization", value="Authorization DESCRIPTION")})

public @ResponseBody JsonObject createAssets() {

Map<String, String> assetData = getAssetMap();

return new JsonObject(assetData);

}

private Map<String, String> getAssetMap() {

Map<String, String> assetData = new TreeMap<String, String>() {{

put("name", "James Bond");

put("age", "doesn't matter");

put("purpose", "saving the world");

}};

System.out.println("(\*) Assets Prepared..");

return assetData;

}

//Blocks

//MetaData

@RequestMapping(method = RequestMethod.GET, value = "/metadata")

@ApiOperation(value = "Send a Transaction.")

// @ApiImplicitParams({@ApiImplicitParam(name="Authorization", value="Authorization DESCRIPTION")})

public @ResponseBody JsonObject createMetadata() {

MetaData metaData = getMetaData();

return new JsonObject(metaData);

}

private MetaData getMetaData() {

// create metadata

MetaData metaData = new MetaData();

metaData.setMetaData("where is he now?", "Thailand");

System.out.println("(\*) Metadata Prepared..");

return metaData;

}

//Validators

//Keys

@RequestMapping(method = RequestMethod.GET, value = "/keys")

@ApiOperation(value = "Get Keys.")

// @ApiImplicitParams({@ApiImplicitParam(name="Authorization", value="Authorization DESCRIPTION")})

public @ResponseBody JsonObject keys() {

KeyPair keys = getKeys();

System.out.println(Base58.encode(keys.getPublic().getEncoded()));

System.out.println(Base58.encode(keys.getPrivate().getEncoded()));

return new JsonObject(keys);

}

/\*\*

\* configures connection url and credentials

\*/

public void setConfig() {

BigchainDbConfigBuilder

.baseUrl("http://testnet.bigchaindb.com") //or use http://testnet.bigchaindb.com

.addToken("app\_id", "6f30487a")

.addToken("app\_key", "0b84ac35ef5efbda162b9db3d1fce3f8").setup();

}

/\*\*

\* generates EdDSA keypair to sign and verify transactions

\* @return KeyPair

\*/

public KeyPair getKeys() {

// prepare your keys

net.i2p.crypto.eddsa.KeyPairGenerator edDsaKpg = new net.i2p.crypto.eddsa.KeyPairGenerator();

KeyPair keyPair = edDsaKpg.generateKeyPair();

System.out.println("(\*) Keys Generated..");

return keyPair;

}

/\*\*

\* performs CREATE transactions on BigchainDB network

\* @param assetData data to store as asset

\* @param metaData data to store as metadata

\* @param keys keys to sign and verify transaction

\* @return id of CREATED asset

\*/

public String doCreate(Map<String, String> assetData, MetaData metaData, KeyPair keys) throws Exception {

try {

//build and send CREATE transaction

Transaction transaction = null;

transaction = BigchainDbTransactionBuilder

.init()

.addAssets(assetData, TreeMap.class)

.addMetaData(metaData)

.operation(Operations.CREATE)

.buildAndSign((EdDSAPublicKey) keys.getPublic(), (EdDSAPrivateKey) keys.getPrivate())

.sendTransaction(handleServerResponse());

System.out.println("(\*) CREATE Transaction sent.. - " + transaction.getId());

return transaction.getId();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

return null;

}

/\*\*

\* performs TRANSFER operations on CREATED assets

\* @param txId id of transaction/asset

\* @param metaData data to append for this transaction

\* @param keys keys to sign and verify transactions

\*/

public void doTransfer(String txId, MetaData metaData, KeyPair keys) throws Exception {

Map<String, String> assetData = new TreeMap<String, String>();

assetData.put("id", txId);

try {

//which transaction you want to fulfill?

FulFill fulfill = new FulFill();

fulfill.setOutputIndex(0);

fulfill.setTransactionId(txId);

//build and send TRANSFER transaction

Transaction transaction = BigchainDbTransactionBuilder

.init()

.addInput(null, fulfill, (EdDSAPublicKey) keys.getPublic())

.addOutput("1", (EdDSAPublicKey) keys.getPublic())

.addAssets(txId, String.class)

.addMetaData(metaData)

.operation(Operations.TRANSFER)

.buildAndSign((EdDSAPublicKey) keys.getPublic(), (EdDSAPrivateKey) keys.getPrivate())

.sendTransaction(handleServerResponse());

System.out.println("(\*) TRANSFER Transaction sent.. - " + transaction.getId());

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

private GenericCallback handleServerResponse() {

//define callback methods to verify response from BigchainDBServer

GenericCallback callback = new GenericCallback() {

@Override

public void transactionMalformed(Response response) {

System.out.println("malformed " + response.message());

onFailure();

}

@Override

public void pushedSuccessfully(Response response) {

System.out.println("pushedSuccessfully");

onSuccess(response);

}

@Override

public void otherError(Response response) {

System.out.println("otherError" + response.message());

onFailure();

}

};

return callback;

}

private void onSuccess(Response response) {

//TODO : Add your logic here with response from server

System.out.println("Transaction posted successfully");

}

private void onFailure() {

//TODO : Add your logic here

System.out.println("Transaction failed");

}

}

package info.smartkit.blockchain.bigchaindb.controllers.exception;

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT\_FOUND)

public class NotFoundException extends RuntimeException {

public NotFoundException() {

super("The resource you requested does not exist");

}

}

package info.smartkit.blockchain.bigchaindb.controllers;

import info.smartkit.blockchain.bigchaindb.dto.JsonObject;

import info.smartkit.blockchain.bigchaindb.dto.JsonString;

import info.smartkit.blockchain.bigchaindb.services.IpfsService;

import io.swagger.annotations.ApiOperation;

import org.apache.logging.log4j.LogManager;

import org.apache.logging.log4j.Logger;

import org.hibernate.validator.constraints.NotBlank;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import org.springframework.web.multipart.MultipartFile;

import javax.validation.Valid;

import javax.validation.constraints.NotNull;

import javax.ws.rs.core.MediaType;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

/\*\*

\* The Class IpfsController.

\*/

@RestController

// @see: https://github.com/ipfs/java-ipfs-api

@RequestMapping(value = "v1/ipfs")

public class IpfsController {

//

private static Logger LOG = LogManager.getLogger(IpfsController.class);

@Autowired

IpfsService ipfsService;

//get ipfs file

@RequestMapping(method = RequestMethod.GET, value = "/{hashID}")

@ApiOperation(value = "Response an IPFS file.")

// @ApiImplicitParams({@ApiImplicitParam(name="Authorization", value="Authorization DESCRIPTION")})

public @ResponseBody

JsonObject getOne(@PathVariable("hashID") String hashID) throws IOException {

byte[] fileContents = ipfsService.get(hashID);

return new JsonObject(fileContents);

}

//put ipfs file

@RequestMapping(method = RequestMethod.POST, value = "/",consumes = MediaType.MULTIPART\_FORM\_DATA)

@ApiOperation(value = "Create new IPFS file.")

// @ApiImplicitParams({@ApiImplicitParam(name="Authorization", value="Authorization DESCRIPTION")})

public @ResponseBody JsonString putOne(@RequestPart(value = "file") @Valid @NotNull @NotBlank MultipartFile multipartFile) throws Exception {

File rawFile = convert(multipartFile);

String ipfsHashID = ipfsService.putFile(rawFile);

return new JsonString(ipfsHashID);

}

//@see: https://stackoverflow.com/questions/24339990/how-to-convert-a-multipart-file-to-file

public File convert(MultipartFile file) throws IOException {

File convFile = new File(file.getOriginalFilename());

convFile.createNewFile();

FileOutputStream fos = new FileOutputStream(convFile);

fos.write(file.getBytes());

fos.close();

return convFile;

}

}

package info.smartkit.blockchain.bigchaindb.controllers;

import info.smartkit.blockchain.bigchaindb.domain.LiUserProfile;

import info.smartkit.blockchain.bigchaindb.dto.DknToken;

import info.smartkit.blockchain.bigchaindb.dto.JsonObject;

import info.smartkit.blockchain.bigchaindb.dto.LiUserConnection;

import info.smartkit.blockchain.bigchaindb.repository.LinkedInUserRepository;

import info.smartkit.blockchain.bigchaindb.services.LinkedInUserService;

import io.swagger.annotations.ApiOperation;

import org.scribe.model.\*;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionRepository;

import org.springframework.social.linkedin.api.LinkedIn;

import org.springframework.social.linkedin.api.LinkedInProfile;

import org.springframework.social.linkedin.api.NetworkStatistics;

import org.springframework.social.linkedin.api.impl.LinkedInTemplate;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

import javax.inject.Inject;

import javax.validation.Valid;

/\*\*

\* The Class LinkedInConnectController.

\*

\* @author yangboz

\*/

@RestController

@RequestMapping("/v1/connect/linkedin/")

public class LinkedInConnectController {

// ==============

// PRIVATE FIELDS

// ==============

private final Logger LOG = LoggerFactory.getLogger(LinkedInConnectController.class);

// Autowire an object of type UserDao

@Autowired

private LinkedInUserRepository \_liUserDao;

//@Autowired

private LinkedIn linkedIn;

@Inject

private ConnectionRepository connectionRepository;

//

@Inject

public LinkedInConnectController(LinkedIn linkedIn, ConnectionRepository connectionRepository) {

this.linkedIn = linkedIn;

this.connectionRepository = connectionRepository;

}

@Autowired

private LinkedInUserService \_linkedInUserService;

//

// @Inject

// public LinkedInConnectController(LinkedIn linkedIn) {

// this.linkedIn = linkedIn;

// }

// private LinkedInRequestToken sessionRequestToken = null;

private Token sessionRequestToken = null;

//

//

@RequestMapping(value = "/profile", method = RequestMethod.POST)

@ApiOperation(httpMethod = "POST", value = "Response a string describing if the access\_token related user profile is successfully received.")

public JsonObject getAccessToken(@RequestBody @Valid DknToken dknToken) {

//

return new JsonObject(this.\_linkedInUserService.getUserProfile(dknToken));

}

@RequestMapping(value = "/access", method = RequestMethod.POST)

@ApiOperation(httpMethod = "POST", value = "Response a string describing if the access\_token related user profile is successfully received.")

public JsonObject getUserProfile(@RequestBody @Valid DknToken dknToken) {

/\*\*

\* Programmatically signs in the user with the given the user ID.

\* @see: spring-social-showcase-boot(SignInUtil)

\*/

LOG.info("dknToken:" + dknToken.toString());

SecurityContextHolder.getContext().setAuthentication(new UsernamePasswordAuthenticationToken(dknToken.getId(), null, null));

//@see: https://developer-programs.linkedin.com/documents/exchange-jsapi-tokens-rest-api-oauth-tokens

// String accessToken = "f8FX29g..."; // access token received from Facebook after OAuth authorization

// Facebook facebook = new FacebookTemplate(accessToken);

LOG.info("connectionRepository.findAllConnections():" + connectionRepository.findAllConnections().toString());

Connection<LinkedIn> connection = connectionRepository.findPrimaryConnection(LinkedIn.class);

// Connection<LinkedIn> connection = connectionRepository.

// "77nayor82qqip3", "UJOUycxP5UgdD3da"

LOG.info("Connection<LinkedIn>:" + connection);

LinkedIn linkedIn = connection != null ? connection.getApi() : new LinkedInTemplate(dknToken.getToken());

LOG.info("linkedIn,isAuthorized():" + linkedIn.isAuthorized() + "," + linkedIn.toString());

//Retrieving a user's profile data.

//@see: http://docs.spring.io/spring-social-facebook/docs/2.0.1.RELEASE/reference/htmlsingle/

LOG.info("linkedIn getProfileById:" + linkedIn.profileOperations().getProfileById(dknToken.getId()));

System.out.println("linkedIn.profileOperations():" + linkedIn.profileOperations().toString());

LinkedInProfile profile = linkedIn.profileOperations().getUserProfile();

LOG.info("LinkedInProfile: id:" + profile.getId());

//Synchronize the FB user profile to DB.

LiUserProfile liUser = new LiUserProfile(profile.getId(), profile.getLastName(), profile.getLastName(), profile.getHeadline(), profile.getIndustry(), profile.getPublicProfileUrl(), profile.getSiteStandardProfileRequest(), profile.getPublicProfileUrl());

//

this.\_liUserDao.save(liUser);

return new JsonObject(profile);

}

@RequestMapping(value = "/connections", method = RequestMethod.GET)

@ApiOperation(httpMethod = "GET", value = "Response a string describing if the user connnection is successfully received.")

public JsonObject getConnections(@RequestBody @Valid DknToken dknToken) {

NetworkStatistics statistics = linkedIn.connectionOperations().getNetworkStatistics();

LiUserConnection connection = new LiUserConnection();

connection.setFirstDegreeCount(statistics.getFirstDegreeCount());

connection.setSecondDegreeCount(statistics.getSecondDegreeCount());

connection.setConnections(linkedIn.connectionOperations().getConnections());

return new JsonObject(connection);

}

}