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);

}

}

//@see: http://www.infoq.com/cn/articles/spring-data-intro

package info.smartkit.blockchain.bigchaindb.domain;

import java.io.Serializable;

import java.util.Date;

import java.util.UUID;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import org.springframework.data.mongodb.core.mapping.Field;

import org.springframework.data.mongodb.core.mapping.Version;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

@Document

@JsonIgnoreProperties(ignoreUnknown = true)

@SuppressWarnings("serial")

public abstract class BaseEntity implements Serializable {

@Id

@Field("uuid")

private String id;

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

@Override

public int hashCode() {

return (id == null) ? 0 : id.hashCode();

}

@Override

public boolean equals(Object obj) {

if (this == obj) return true;

if (obj == null) return false;

if (getClass() != obj.getClass()) return false;

BaseEntity other = (BaseEntity) obj;

if (id == null) return other.id == null;

return id.equals(other.id);

}

/\*\*

\* All objects will have a unique UUID which allows for the decoupling from DB generated ids

\*/

// @Column(length=36)

private String uuid;

private Date timeCreated;

public BaseEntity() {

this(UUID.randomUUID());

}

public BaseEntity(UUID guid) {

// Assert.notNull(guid, "UUID is required");

setUuid(guid.toString());

this.timeCreated = new Date();

}

public UUID getUuid() {

return UUID.fromString(uuid);

}

public void setUuid(String uuid) {

this.uuid = uuid;

}

// public int hashCode() {

// return getUuid().hashCode();

// }

/\*\*

\* In most instances we can rely on the UUID to identify the object.

\* Subclasses may want a user friendly identifier for constructing easy to read urls

\* <p/>

\* <code>

\* /user/1883c578-76be-47fb-a5c1-7bbea3bf7fd0 using uuid as the identifier

\* <p/>

\* /user/jsmith using the username as the identifier

\* <p/>

\* </code>

\*

\* @return Object unique identifier for the object

\*/

public Object getIdentifier() {

return getUuid().toString();

}

public Date getTimeCreated() {

return timeCreated;

}

}

package info.smartkit.blockchain.bigchaindb.domain;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

import org.springframework.data.mongodb.core.mapping.Field;

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

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

/\*\*

\* Created by yangboz on 15/9/19.

\*/

@JsonIgnoreProperties(ignoreUnknown = true)

@SuppressWarnings("serial")

@Document(collection = "dkn\_li\_user\_profile")

public class LiUserProfile extends LinkedInProfile {

@Id

@Field("uuid")

private String id;

public LiUserProfile(String id, String firstName, String lastName, String headline, String industry, String publicProfileUrl, UrlResource siteStandardProfileRequest, String profilePictureUrl) {

super(id, firstName, lastName, headline, industry, publicProfileUrl, siteStandardProfileRequest, profilePictureUrl);

}

//Introducing the dummy constructor for JSON Mapper

public LiUserProfile() {

super(null, null, null, null, null, null, null, null);

}

}

package info.smartkit.blockchain.bigchaindb.dto;

public class DknToken {

private String id;//LinkedIn is member\_id;

private String token;//LinkedIn is bearer token;

private String provider;//linkedin...

public DknToken() {

}

public DknToken(String id, String token, String provider) {

this.id = id;

this.token = token;

this.provider = provider;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public String getToken() {

return token;

}

public void setToken(String token) {

this.token = token;

}

@Override

public String toString() {

return "DknToken{" +

"id='" + id + '\'' +

", token='" + token + '\'' +

", provider='" + provider + '\'' +

'}';

}

public String getProvider() {

return provider;

}

public void setProvider(String provider) {

this.provider = provider;

}

public String getIdentifier() {

return this.getProvider() + "\_" + this.getId();

}

}

package info.smartkit.blockchain.bigchaindb.dto;

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

import java.util.List;

/\*\*

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

\*/

public class LiUserConnection {

private int firstDegreeCount;

private int secondDegreeCount;

private List<LinkedInProfile> connections;

public int getFirstDegreeCount() {

return firstDegreeCount;

}

public void setFirstDegreeCount(int firstDegreeCount) {

this.firstDegreeCount = firstDegreeCount;

}

public int getSecondDegreeCount() {

return secondDegreeCount;

}

public void setSecondDegreeCount(int secondDegreeCount) {

this.secondDegreeCount = secondDegreeCount;

}

public List<LinkedInProfile> getConnections() {

return connections;

}

public void setConnections(List<LinkedInProfile> connections) {

this.connections = connections;

}

}

package info.smartkit.blockchain.bigchaindb.repository;

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

import org.springframework.data.mongodb.repository.MongoRepository;

/\*\*

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

\*/

public interface LinkedInUserRepository extends MongoRepository<LiUserProfile, String> {

}

/\*

\* Copyright (c) 2018. SMARTKIT.INFO.

\*/

package info.smartkit.blockchain.bigchaindb.services;

import io.ipfs.api.IPFS;

import io.ipfs.api.MerkleNode;

import io.ipfs.api.NamedStreamable;

import io.ipfs.multihash.Multihash;

import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;

import java.io.File;

import java.io.IOException;

import java.util.logging.Logger;

@Service

public class IpfsService {

public IPFS getIpfs() {

return ipfs;

}

IPFS ipfs = null;

protected Logger logger = Logger.getLogger(IpfsService.class

.getName());

@PostConstruct

public void main() throws IOException {

// Can't do this in the constructor because the RestTemplate injection

// happens afterwards.

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

IPFS ipfs = new IPFS("/ip4/127.0.0.1/tcp/5001");

logger.warning("hard coded IPFS info: "+ipfs.toString());

ipfs.refs.local();

}

public String putFile(File file) throws IOException {

// logger.info("IpfsService put() invoked: for " + file);

NamedStreamable.FileWrapper fileWrapper = new NamedStreamable.FileWrapper(file);

MerkleNode addResult = ipfs.add(fileWrapper).get(0);

logger.info("IpfsService put() result: " + addResult.toJSONString());

return addResult.toJSONString();

}

public String putByte(byte[] file) throws IOException {

NamedStreamable.ByteArrayWrapper fileWrapper = new NamedStreamable.ByteArrayWrapper(file);//"hello.txt", "G'day world! IPFS rocks!".getBytes()

MerkleNode addResult = ipfs.add(fileWrapper).get(0);

logger.info("IpfsService put() result: " + addResult.toJSONString());

return addResult.toJSONString();

}

public byte[] get(String hashStr) throws IOException {

Multihash filePointer = Multihash.fromBase58(hashStr);//"QmPZ9gcCEpqKTo6aq61g2nXGUhM4iCL3ewB6LDXZCtioEB"

byte[] fileContents = ipfs.cat(filePointer);

logger.info("IpfsService get() for " + hashStr);

return fileContents;

}

}

package info.smartkit.blockchain.bigchaindb.services;

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

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

public interface LinkedInUserService {

LiUserProfile getUserProfile(DknToken token);

}

package info.smartkit.blockchain.bigchaindb.services.impls;

import com.fasterxml.jackson.core.JsonGenerationException;

import com.fasterxml.jackson.core.JsonParser;

import com.fasterxml.jackson.databind.JsonMappingException;

import com.fasterxml.jackson.databind.ObjectMapper;

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

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

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

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

import info.smartkit.blockchain.bigchaindb.utils.OAuthCookie;

import org.scribe.builder.ServiceBuilder;

import org.scribe.builder.api.LinkedInApi;

import org.scribe.model.OAuthRequest;

import org.scribe.model.Response;

import org.scribe.model.Token;

import org.scribe.model.Verb;

import org.scribe.oauth.OAuthService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.core.env.Environment;

import org.springframework.social.connect.ConnectionRepository;

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

import org.springframework.stereotype.Service;

import javax.crypto.Mac;

import javax.crypto.spec.SecretKeySpec;

import javax.inject.Inject;

import java.io.IOException;

/\*\*

\* Created by yangboz on 9/24/15.

\*/

@Service

public class LinkedInUserServiceImpl implements LinkedInUserService {

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

// PRIVATE FIELDS

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

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

// Autowire an object of type UserDao

@Autowired

private LinkedInUserRepository \_liUserDao;

// @Autowired

private LinkedIn linkedIn;

@Inject

private ConnectionRepository connectionRepository;

//

@Inject

public LinkedInUserServiceImpl(LinkedIn linkedIn, ConnectionRepository connectionRepository) {

this.linkedIn = linkedIn;

this.connectionRepository = connectionRepository;

}

@Autowired

Environment environment;

public static final String OAUTH2\_ACCESS\_TOKEN = "xoauth\_oauth2\_access\_token";

public static final String X\_LI\_FORMAT = "x-li-format";

//

public final String KEY() {

return environment.getProperty("spring.social.linkedin.consumerKey");

}

public final String SECRET() {

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

}

public static final String ACCESS\_TOKEN\_ENDPOINT = "https://api.linkedin.com/uas/oauth/accessToken";

public static final String PROFILE\_URL = "http://api.linkedin.com/v1/people/~:(id,first-name,last-name,headline)";

@Override

public LiUserProfile getUserProfile(DknToken token) {

//@see: https://github.com/fernandezpablo85/scribe-java/wiki/getting-started

//@see: https://github.com/fernandezpablo85/TokenExchangeSample/blob/master/src/main/java/com/linkedin/oauth/ExchangeService.java

//@see: https://github.com/fernandezpablo85/scribe-java/tree/master/src/test/java/org/scribe/examples

//1

OAuthService service = new ServiceBuilder()

.apiKey(KEY())

.apiSecret(SECRET())

.provider(LinkedInApi.withScopes("r\_basicprofile r\_network r\_emailaddress rw\_company\_admin"))

.build();

// Exchange 2.0 token for 1.0a (long lived)

OAuthRequest oAuthRequest = new OAuthRequest(Verb.POST, ACCESS\_TOKEN\_ENDPOINT);

// Add the 2.0 token as a parameter

oAuthRequest.addHeader(X\_LI\_FORMAT, "json");

oAuthRequest.addBodyParameter(OAUTH2\_ACCESS\_TOKEN, token.getToken());

// Use an empty 1.0a access\_token

Token emptyToken = new Token("", "");

// Sign and then send the request

service.signRequest(emptyToken, oAuthRequest);

Response oAuthRequestResp = oAuthRequest.send();

LOG.info("Raw oAuthRequestResp.getBody():" + oAuthRequestResp.getBody());

OAuthCookie oAuthCookie = this.plainTextToOAuthCookie(oAuthRequestResp.getBody());

LOG.info("Parsed OAuthCookie:" + oAuthCookie);

//Profile fetch

Token fetchRequestToken = new Token(oAuthCookie.getOauth\_token(), oAuthCookie.getOauth\_token\_secret());

//

OAuthRequest anotherOauthRequest = new OAuthRequest(Verb.GET, PROFILE\_URL);

anotherOauthRequest.addHeader(X\_LI\_FORMAT, "json");//format as JSON

service.signRequest(fetchRequestToken, anotherOauthRequest);

Response response = anotherOauthRequest.send();

LOG.info("FetchReqeustResp.getBody():" + response.getBody());

//

LiUserProfile liUserProfile = null;

ObjectMapper mapper = new ObjectMapper();

// mapper.configure(JsonParser.Feature.ALLOW\_SINGLE\_QUOTES, true);

try {

liUserProfile = mapper.readValue(response.getBody(), LiUserProfile.class);

} catch (IOException e) {

e.printStackTrace();

}

LOG.info("Final mapped LiUserProfile:" + liUserProfile);

return liUserProfile;

}

private byte[] shaSign(String baseString, String secret) {

try {

Mac mac = Mac.getInstance("HmacSHA1");

SecretKeySpec secretKey = new SecretKeySpec(secret.getBytes(), "HmacSHA1");

mac.init(secretKey);

return mac.doFinal(baseString.getBytes());

} catch (Exception e) {

throw new IllegalStateException("Error while generating the HMAC-SHA1 signature", e);

}

}

private OAuthCookie plainTextToOAuthCookie(String plainText) {

//oauth\_token=78--b367d0e9-a312-4fc8-814f-c20444c9f4bd&oauth\_token\_secret=b89bd23c-459f-4698-a6ba-5e76cd5d8d07&oauth\_expires\_in=4742507&oauth\_authorization\_expires\_in=4742507

String jsonStr = "{";

String[] plainTexts = plainText.split("&");

for (int i = 0; i < plainTexts.length; i++) {

String[] textElements = plainTexts[i].split("=");

if (i > 0) {

jsonStr += ",'" + textElements[0] + "'";

} else {

jsonStr += "'" + textElements[0] + "'";

}

//

jsonStr += ":";

jsonStr += "'" + textElements[1] + "'";

}

jsonStr += "}";

OAuthCookie oAuthCookie = null;

ObjectMapper mapper = new ObjectMapper();

mapper.configure(JsonParser.Feature.ALLOW\_SINGLE\_QUOTES, true);

try {

oAuthCookie = mapper.readValue(jsonStr, OAuthCookie.class);

} catch (JsonGenerationException e) {

e.printStackTrace();

} catch (JsonMappingException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

System.out.println("Translated oAuthCookie:" + oAuthCookie.toString());

return oAuthCookie;

}

}

package info.smartkit.blockchain.bigchaindb.utils;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.http.HttpRequest;

import org.springframework.http.client.ClientHttpRequestExecution;

import org.springframework.http.client.ClientHttpRequestInterceptor;

import org.springframework.http.client.ClientHttpResponse;

import java.io.IOException;

/\*\*

\* Created by yangboz on 10/1/15.

\*

\* @see: http://stackoverflow.com/questions/7952154/spring-resttemplate-how-to-enable-full-debugging-logging-of-requests-responses

\*/

public class LoggingRequestInterceptor implements ClientHttpRequestInterceptor {

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

@Override

public ClientHttpResponse intercept(HttpRequest request, byte[] body, ClientHttpRequestExecution execution) throws IOException {

ClientHttpResponse response = execution.execute(request, body);

log(request, body, response);

return response;

}

private void log(HttpRequest request, byte[] body, ClientHttpResponse response) throws IOException {

//do logging

LOG.info("request:" + request.getURI().toString() + ",response:" + response.getStatusCode() + ",body:" + body.toString());

}

}

package info.smartkit.blockchain.bigchaindb.utils;

/\*\*

\* Created by yangboz on 9/24/15.

\*

\* @see https://github.com/fernandezpablo85/TokenExchangeSample/blob/master/src/main/java/com/linkedin/oauth/ExchangeService.java

\*/

public class OAuthCookie {

public OAuthCookie() {

}

// public String signature\_method;

// public String signature\_order;

// public String access\_token;

// public String signature;

// public String member\_id;

// public String oauth\_one\_token;

private String oauth\_token;

private String oauth\_token\_secret;

private String oauth\_expires\_in;

private String oauth\_authorization\_expires\_in;

private String oauth\_problem;//Esp for problem parsing.

public String getOauth\_token() {

return oauth\_token;

}

public void setOauth\_token(String oauth\_token) {

this.oauth\_token = oauth\_token;

}

public String getOauth\_token\_secret() {

return oauth\_token\_secret;

}

public void setOauth\_token\_secret(String oauth\_token\_secret) {

this.oauth\_token\_secret = oauth\_token\_secret;

}

public String getOauth\_expires\_in() {

return oauth\_expires\_in;

}

public void setOauth\_expires\_in(String oauth\_expires\_in) {

this.oauth\_expires\_in = oauth\_expires\_in;

}

public String getOauth\_authorization\_expires\_in() {

return oauth\_authorization\_expires\_in;

}

public void setOauth\_authorization\_expires\_in(String oauth\_authorization\_expires\_in) {

this.oauth\_authorization\_expires\_in = oauth\_authorization\_expires\_in;

}

public String toString() {

return "oauth\_token:" + getOauth\_token() + ",oauth\_token\_secret:" + getOauth\_token\_secret()

+ ",oauth\_expires\_in:" + getOauth\_expires\_in() + ",oauth\_authorization\_expires\_in:" + getOauth\_authorization\_expires\_in();

}

public String getOauth\_problem() {

return oauth\_problem;

}

public void setOauth\_problem(String oauth\_problem) {

this.oauth\_problem = oauth\_problem;

}

}

package net.exacode.spring.social.connect;

import java.util.ArrayList;

import java.util.Collections;

import java.util.LinkedList;

import java.util.List;

import java.util.Set;

import org.springframework.dao.DuplicateKeyException;

import org.springframework.dao.EmptyResultDataAccessException;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionData;

import org.springframework.social.connect.ConnectionFactoryLocator;

import org.springframework.social.connect.ConnectionKey;

import org.springframework.social.connect.ConnectionRepository;

import org.springframework.social.connect.DuplicateConnectionException;

import org.springframework.social.connect.NoSuchConnectionException;

import org.springframework.social.connect.NotConnectedException;

import org.springframework.util.LinkedMultiValueMap;

import org.springframework.util.MultiValueMap;

/\*\*

\* {@link ConnectionRepository} that abstracts from persistence layer.

\*

\* @author mendlik

\*

\*/

public class GenericConnectionRepository implements ConnectionRepository {

private final String userId;

private final SocialConnectionDao connectionService;

private final ConnectionFactoryLocator connectionFactoryLocator;

public GenericConnectionRepository(String userId,

SocialConnectionDao connectionService,

ConnectionFactoryLocator connectionFactoryLocator) {

this.userId = userId;

this.connectionService = connectionService;

this.connectionFactoryLocator = connectionFactoryLocator;

}

/\*\*

\* Add a new connection to this repository for the current user.

\*/

@Override

public void addConnection(Connection<?> connection) {

try {

ConnectionData data = connection.createData();

int rank = connectionService.getMaxRank(userId,

data.getProviderId());

connectionService.create(userId, connection, rank);

} catch (DuplicateKeyException e) {

throw new DuplicateConnectionException(connection.getKey());

}

}

/\*\*

\* Find all connections the current user has across all providers

\*/

@Override

public MultiValueMap<String, Connection<?>> findAllConnections() {

List<Connection<?>> resultList = connectionService

.getConnections(this.userId);

MultiValueMap<String, Connection<?>> connections = new LinkedMultiValueMap<String, Connection<?>>();

Set<String> registeredProviderIds = this.connectionFactoryLocator

.registeredProviderIds();

for (String registeredProviderId : registeredProviderIds) {

connections.put(registeredProviderId,

Collections.<Connection<?>> emptyList());

}

for (Connection<?> connection : resultList) {

String providerId = connection.getKey().getProviderId();

if (connections.get(providerId).size() == 0) {

connections.put(providerId, new LinkedList<Connection<?>>());

}

connections.add(providerId, connection);

}

return connections;

}

/\*\*

\* Find the connections the current user has to the provider of the given

\* API

\*/

@SuppressWarnings("unchecked")

@Override

public <A> List<Connection<A>> findConnections(Class<A> apiType) {

List<?> connections = findConnections(getProviderId(apiType));

return (List<Connection<A>>) connections;

}

/\*\*

\* Find the connections the current user has to the provider registered by

\* the given id

\*/

@Override

public List<Connection<?>> findConnections(String providerId) {

return connectionService.getConnections(this.userId, providerId);

}

/\*\*

\* Find the connections the current user has to the given provider users.

\*/

@Override

public MultiValueMap<String, Connection<?>> findConnectionsToUsers(

MultiValueMap<String, String> providerUsers) {

if (providerUsers == null || providerUsers.isEmpty()) {

throw new IllegalArgumentException(

"Unable to execute find: no providerUsers provided");

}

List<Connection<?>> resultList = connectionService.getConnections(

userId, providerUsers);

MultiValueMap<String, Connection<?>> connectionsForUsers = new LinkedMultiValueMap<String, Connection<?>>();

for (Connection<?> connection : resultList) {

String providerId = connection.getKey().getProviderId();

List<String> userIds = providerUsers.get(providerId);

List<Connection<?>> connections = connectionsForUsers

.get(providerId);

if (connections == null) {

connections = new ArrayList<Connection<?>>(userIds.size());

for (int i = 0; i < userIds.size(); i++) {

connections.add(null);

}

connectionsForUsers.put(providerId, connections);

}

String providerUserId = connection.getKey().getProviderUserId();

int connectionIndex = userIds.indexOf(providerUserId);

connections.set(connectionIndex, connection);

}

return connectionsForUsers;

}

/\*\*

\* Get a connection for the current user by its key

\*/

@Override

public Connection<?> getConnection(ConnectionKey connectionKey) {

try {

return connectionService.getConnection(userId,

connectionKey.getProviderId(),

connectionKey.getProviderUserId());

} catch (EmptyResultDataAccessException e) {

throw new NoSuchConnectionException(connectionKey);

}

}

/\*\*

\* Get a connection between the current user and the given provider user.

\*/

@Override

@SuppressWarnings("unchecked")

public <A> Connection<A> getConnection(Class<A> apiType,

String providerUserId) {

String providerId = getProviderId(apiType);

return (Connection<A>) getConnection(new ConnectionKey(providerId,

providerUserId));

}

/\*\*

\* Get the "primary" connection the current user has to the provider of the

\* given API.

\*/

@Override

@SuppressWarnings("unchecked")

public <A> Connection<A> getPrimaryConnection(Class<A> apiType) {

String providerId = getProviderId(apiType);

Connection<A> connection = (Connection<A>) findPrimaryConnection(providerId);

if (connection == null) {

throw new NotConnectedException(providerId);

}

return connection;

}

/\*\*

\* Find the "primary" connection the current user has to the provider of the

\* given API

\*/

@Override

@SuppressWarnings("unchecked")

public <A> Connection<A> findPrimaryConnection(Class<A> apiType) {

String providerId = getProviderId(apiType);

return (Connection<A>) findPrimaryConnection(providerId);

}

/\*\*

\* Update a Connection already added to this repository.

\*/

@Override

public void updateConnection(Connection<?> connection) {

connectionService.update(userId, connection);

}

/\*\*

\* Remove all Connections between the current user and the provider from

\* this repository.

\*/

@Override

public void removeConnections(String providerId) {

connectionService.remove(userId, providerId);

}

/\*\*

\* Remove a single Connection for the current user from this repository.

\*/

@Override

public void removeConnection(ConnectionKey connectionKey) {

connectionService.remove(userId, connectionKey);

}

private <A> String getProviderId(Class<A> apiType) {

return connectionFactoryLocator.getConnectionFactory(apiType)

.getProviderId();

}

private Connection<?> findPrimaryConnection(String providerId) {

// where userId = ? and providerId = ? and rank = 1

return connectionService.getPrimaryConnection(userId, providerId);

}

}

package net.exacode.spring.social.connect;

import java.util.Arrays;

import java.util.List;

import java.util.Set;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionFactoryLocator;

import org.springframework.social.connect.ConnectionKey;

import org.springframework.social.connect.ConnectionRepository;

import org.springframework.social.connect.ConnectionSignUp;

import org.springframework.social.connect.UsersConnectionRepository;

/\*\*

\* {@link UsersConnectionRepository} that abstracts from persistence layer.

\*

\* @author mendlik

\*/

public class GenericUsersConnectionRepository implements

UsersConnectionRepository {

private final SocialConnectionDao mongoService;

private final ConnectionFactoryLocator connectionFactoryLocator;

private ConnectionSignUp connectionSignUp;

public GenericUsersConnectionRepository(SocialConnectionDao mongoService,

ConnectionFactoryLocator connectionFactoryLocator) {

this.mongoService = mongoService;

this.connectionFactoryLocator = connectionFactoryLocator;

}

public void setConnectionSignUp(ConnectionSignUp connectionSignUp) {

this.connectionSignUp = connectionSignUp;

}

@Override

public List<String> findUserIdsWithConnection(Connection<?> connection) {

ConnectionKey key = connection.getKey();

List<String> localUserIds = mongoService.getUserIds(

key.getProviderId(), key.getProviderUserId());

if (localUserIds.size() == 0 && connectionSignUp != null) {

String newUserId = connectionSignUp.execute(connection);

if (newUserId != null) {

createConnectionRepository(newUserId).addConnection(connection);

return Arrays.asList(newUserId);

}

}

return localUserIds;

}

@Override

public Set<String> findUserIdsConnectedTo(String providerId,

Set<String> providerUserIds) {

return mongoService.getUserIds(providerId, providerUserIds);

}

@Override

public ConnectionRepository createConnectionRepository(String userId) {

if (userId == null) {

throw new IllegalArgumentException("userId cannot be null");

}

return new GenericConnectionRepository(userId, mongoService,

connectionFactoryLocator);

}

}

package net.exacode.spring.social.connect;

import java.util.List;

import java.util.Set;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionKey;

import org.springframework.util.MultiValueMap;

/\*\*

\* Data access layer that separates spring social logic from persistance layer.

\* <p>

\* Original spring supporting schema is defined in lib

\* {@code spring-social-core} in file

\* {@code /org/springframework/social/connect/jdbc/JdbcUsersConnectionRepository.sql}

\* .

\*

\* @author mendlik

\*

\*/

public interface SocialConnectionDao {

int getMaxRank(String userId, String providerId);

void create(String userId, Connection<?> userConn, int rank);

void update(String userId, Connection<?> userConn);

void remove(String userId, ConnectionKey connectionKey);

void remove(String userId, String providerId);

Connection<?> getPrimaryConnection(String userId, String providerId);

Connection<?> getConnection(String userId, String providerId,

String providerUserId);

List<Connection<?>> getConnections(String userId);

List<Connection<?>> getConnections(String userId, String providerId);

List<Connection<?>> getConnections(String userId,

MultiValueMap<String, String> providerUsers);

Set<String> getUserIds(String providerId, Set<String> providerUserIds);

List<String> getUserIds(String providerId, String providerUserId);

}

package net.exacode.spring.social.connect.mongo;

import org.bson.types.ObjectId;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.index.CompoundIndex;

import org.springframework.data.mongodb.core.index.CompoundIndexes;

import org.springframework.data.mongodb.core.mapping.Document;

/\*\*

\* The MongoDB collection for the spring social connections.

\* <p>

\* Note: After conversion to {@link MongoConnection} all OAuth tokens are

\* encrypted.

\*

\*

\* @author mendlik

\*/

@Document(collection = "connections")

@CompoundIndexes({

@CompoundIndex(name = "connectionsRankIdx", def = "{'userId': 1, 'providerId': 1, 'rank': 1}", unique = true),

@CompoundIndex(name = "connectionsPrimaryIdx", def = "{'userId': 1, 'providerId': 1, 'providerUserId': 1}", unique = true) })

public class MongoConnection {

@Id

private ObjectId id;

private String userId;

private String providerId;

private String providerUserId;

private int rank;

private String displayName;

private String profileUrl;

private String imageUrl;

private String accessToken;

private String secret;

private String refreshToken;

private Long expireTime;

public ObjectId getId() {

return id;

}

public String getUserId() {

return userId;

}

public void setUserId(String userId) {

this.userId = userId;

}

public String getProviderId() {

return providerId;

}

public void setProviderId(String providerId) {

this.providerId = providerId;

}

public String getProviderUserId() {

return providerUserId;

}

public void setProviderUserId(String providerUserId) {

this.providerUserId = providerUserId;

}

public int getRank() {

return rank;

}

public void setRank(int rank) {

this.rank = rank;

}

public String getDisplayName() {

return displayName;

}

public void setDisplayName(String displayName) {

this.displayName = displayName;

}

public String getProfileUrl() {

return profileUrl;

}

public void setProfileUrl(String profileUrl) {

this.profileUrl = profileUrl;

}

public String getImageUrl() {

return imageUrl;

}

public void setImageUrl(String imageUrl) {

this.imageUrl = imageUrl;

}

public String getAccessToken() {

return accessToken;

}

public void setAccessToken(String accessToken) {

this.accessToken = accessToken;

}

public String getSecret() {

return secret;

}

public void setSecret(String secret) {

this.secret = secret;

}

public String getRefreshToken() {

return refreshToken;

}

public void setRefreshToken(String refreshToken) {

this.refreshToken = refreshToken;

}

public Long getExpireTime() {

return expireTime;

}

public void setExpireTime(Long expireTime) {

this.expireTime = expireTime;

}

}

package net.exacode.spring.social.connect.mongo;

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

import org.springframework.security.crypto.encrypt.TextEncryptor;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionData;

import org.springframework.social.connect.ConnectionFactory;

import org.springframework.social.connect.ConnectionFactoryLocator;

import org.springframework.stereotype.Component;

/\*\*

\* Converts data between spring social connections {@link Connection} and

\* MongoDB documents {@link MongoConnection}.

\*

\* @author mendlik

\*/

@Component

public class MongoConnectionConverter {

private final ConnectionFactoryLocator connectionFactoryLocator;

private final TextEncryptor textEncryptor;

@Autowired

public MongoConnectionConverter(

ConnectionFactoryLocator connectionFactoryLocator,

TextEncryptor textEncryptor) {

this.connectionFactoryLocator = connectionFactoryLocator;

this.textEncryptor = textEncryptor;

}

public Connection<?> convert(MongoConnection cnn) {

if (cnn == null)

return null;

ConnectionData connectionData = fillConnectionData(cnn);

ConnectionFactory<?> connectionFactory = connectionFactoryLocator

.getConnectionFactory(connectionData.getProviderId());

return connectionFactory.createConnection(connectionData);

}

private ConnectionData fillConnectionData(MongoConnection uc) {

return new ConnectionData(uc.getProviderId(), uc.getProviderUserId(),

uc.getDisplayName(), uc.getProfileUrl(), uc.getImageUrl(),

decrypt(uc.getAccessToken()), decrypt(uc.getSecret()),

decrypt(uc.getRefreshToken()), uc.getExpireTime());

}

public MongoConnection convert(Connection<?> cnn) {

ConnectionData data = cnn.createData();

MongoConnection userConn = new MongoConnection();

userConn.setProviderId(data.getProviderId());

userConn.setProviderUserId(data.getProviderUserId());

userConn.setDisplayName(data.getDisplayName());

userConn.setProfileUrl(data.getProfileUrl());

userConn.setImageUrl(data.getImageUrl());

userConn.setAccessToken(encrypt(data.getAccessToken()));

userConn.setSecret(encrypt(data.getSecret()));

userConn.setRefreshToken(encrypt(data.getRefreshToken()));

userConn.setExpireTime(data.getExpireTime());

return userConn;

}

private String decrypt(String encryptedText) {

return encryptedText != null ? textEncryptor.decrypt(encryptedText)

: encryptedText;

}

private String encrypt(String text) {

return text != null ? textEncryptor.encrypt(text) : text;

}

}

package net.exacode.spring.social.connect.mongo;

import static org.springframework.data.mongodb.core.query.Criteria.where;

import static org.springframework.data.mongodb.core.query.Query.query;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Map.Entry;

import java.util.Set;

import net.exacode.spring.social.connect.SocialConnectionDao;

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

import org.springframework.dao.DuplicateKeyException;

import org.springframework.data.domain.Sort;

import org.springframework.data.domain.Sort.Direction;

import org.springframework.data.mongodb.core.MongoTemplate;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import org.springframework.data.mongodb.core.query.Update;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionKey;

import org.springframework.stereotype.Service;

import org.springframework.util.MultiValueMap;

import com.mongodb.WriteConcern;

/\*\*

\* A MongoDB data access object for spring social connections

\*

\* @author mendlik

\*/

@Service

public class MongoConnectionDao implements SocialConnectionDao {

private final MongoTemplate mongoTemplate;

private final MongoConnectionConverter converter;

@Autowired

public MongoConnectionDao(MongoTemplate mongoTemplate,

MongoConnectionConverter converter) {

this.mongoTemplate = mongoTemplate;

this.converter = converter;

}

/\*\*

\* Returns the max connection rank for the user and the provider.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#getMaxRank(String,

\* String)

\*/

@Override

public int getMaxRank(String userId, String providerId) {

// select coalesce(max(rank) + 1, 1) as rank from UserConnection where

// userId = ? and providerId = ?

Query q = query(where("userId").is(userId).and("providerId")

.is(providerId));

q.with(new Sort(Direction.DESC, "rank"));

MongoConnection cnn = mongoTemplate.findOne(q, MongoConnection.class);

if (cnn == null)

return 1;

return cnn.getRank() + 1;

}

/\*\*

\* Create a new connection for the user.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#create(String,

\* Connection, int)

\*/

@Override

public void create(String userId, Connection<?> userConn, int rank) {

MongoConnection mongoCnn = converter.convert(userConn);

mongoCnn.setUserId(userId);

mongoCnn.setRank(rank);

mongoTemplate.insert(mongoCnn);

}

/\*\*

\* Update a connection.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#update(String,

\* Connection)

\*/

@Override

public void update(String userId, Connection<?> userConn) {

MongoConnection mongoCnn = converter.convert(userConn);

mongoCnn.setUserId(userId);

try {

mongoTemplate.setWriteConcern(WriteConcern.SAFE);

mongoTemplate.save(mongoCnn);

} catch (DuplicateKeyException e) {

Query q = query(where("userId").is(userId).and("providerId")

.is(mongoCnn.getProviderId()).and("providerUserId")

.is(mongoCnn.getProviderUserId()));

Update update = Update

.update("expireTime", mongoCnn.getExpireTime())

.set("accessToken", mongoCnn.getAccessToken())

.set("profileUrl", mongoCnn.getProfileUrl())

.set("imageUrl", mongoCnn.getImageUrl())

.set("displayName", mongoCnn.getDisplayName());

mongoTemplate.findAndModify(q, update, MongoConnection.class);

}

}

/\*\*

\* Remove a connection.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#remove(String,

\* ConnectionKey)

\*/

@Override

public void remove(String userId, ConnectionKey connectionKey) {

// delete where userId = ? and providerId = ? and providerUserId = ?

Query q = query(where("userId").is(userId).and("providerId")

.is(connectionKey.getProviderId()).and("providerUserId")

.is(connectionKey.getProviderUserId()));

mongoTemplate.remove(q, MongoConnection.class);

}

/\*\*

\* Remove all the connections for a user on a provider.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#remove(String,

\* String)

\*/

@Override

public void remove(String userId, String providerId) {

// delete where userId = ? and providerId = ?

Query q = query(where("userId").is(userId).and("providerId")

.is(providerId));

mongoTemplate.remove(q, MongoConnection.class);

}

/\*\*

\* Return the primary connection.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#getPrimaryConnection(String,

\* String)

\*/

@Override

public Connection<?> getPrimaryConnection(String userId, String providerId) {

// where userId = ? and providerId = ? and rank = 1

Query q = query(where("userId").is(userId).and("providerId")

.is(providerId).and("rank").is(1));

MongoConnection mc = mongoTemplate.findOne(q, MongoConnection.class);

return converter.convert(mc);

}

/\*\*

\* Get the connection for user, provider and provider user id.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#getConnection(String,

\* String, String)

\*/

@Override

public Connection<?> getConnection(String userId, String providerId,

String providerUserId) {

// where userId = ? and providerId = ? and providerUserId = ?

Query q = query(where("userId").is(userId).and("providerId")

.is(providerId).and("providerUserId").is(providerUserId));

MongoConnection mc = mongoTemplate.findOne(q, MongoConnection.class);

return converter.convert(mc);

}

/\*\*

\* Get all the connections for an user id.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#getConnections(String)

\*/

@Override

public List<Connection<?>> getConnections(String userId) {

// select where userId = ? order by providerId, rank

Query q = query(where("userId").is(userId));

q.with(new Sort(Direction.ASC, "providerId"));

q.with(new Sort(Direction.ASC, "rank"));

return runQuery(q);

}

/\*\*

\* Get all the connections for an user id on a provider.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#getConnections(String,

\* String)

\*/

@Override

public List<Connection<?>> getConnections(String userId, String providerId) {

// where userId = ? and providerId = ? order by rank

Query q = new Query(where("userId").is(userId).and("providerId")

.is(providerId));

q.with(new Sort(Direction.ASC, "rank"));

return runQuery(q);

}

/\*\*

\* Get all the connections for an user.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#getConnections(String,

\* MultiValueMap)

\*/

@Override

public List<Connection<?>> getConnections(String userId,

MultiValueMap<String, String> providerUsers) {

// userId? and providerId = ? and providerUserId in (?, ?, ...) order by

// providerId, rank

if (providerUsers == null || providerUsers.isEmpty()) {

throw new IllegalArgumentException(

"Unable to execute find: no providerUsers provided");

}

List<Criteria> lc = new ArrayList<Criteria>();

for (Entry<String, List<String>> entry : providerUsers.entrySet()) {

String providerId = entry.getKey();

lc.add(where("providerId").is(providerId).and("providerUserId")

.in(entry.getValue()));

}

Criteria criteria = where("userId").is(userId);

criteria.orOperator(lc.toArray(new Criteria[lc.size()]));

Query q = new Query(criteria);

q.with(new Sort(Direction.ASC, "providerId"));

q.with(new Sort(Direction.ASC, "rank"));

return runQuery(q);

}

/\*\*

\* Get the user ids on the provider.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#getUserIds(String,

\* Set)

\*/

@Override

public Set<String> getUserIds(String providerId, Set<String> providerUserIds) {

// select userId from " + tablePrefix + "UserConnection where providerId

// = :providerId and providerUserId in (:providerUserIds)

Query q = query(where("providerId").is(providerId)

.and("providerUserId")

.in(new ArrayList<String>(providerUserIds)));

q.fields().include("userId");

List<MongoConnection> results = mongoTemplate.find(q,

MongoConnection.class);

Set<String> userIds = new HashSet<String>();

for (MongoConnection mc : results) {

userIds.add(mc.getUserId());

}

return userIds;

}

/\*\*

\* Get the user ids on the provider with a given provider user id.

\*

\* @see net.exacode.spring.social.connect.SocialConnectionDao#getUserIds(String,

\* String)

\*/

@Override

public List<String> getUserIds(String providerId, String providerUserId) {

// select userId where providerId = ? and providerUserId = ?",

Query q = query(where("providerId").is(providerId)

.and("providerUserId").is(providerUserId));

q.fields().include("userId");

List<MongoConnection> results = mongoTemplate.find(q,

MongoConnection.class);

List<String> userIds = new ArrayList<String>();

for (MongoConnection mc : results) {

userIds.add(mc.getUserId());

}

return userIds;

}

private List<Connection<?>> runQuery(Query query) {

List<MongoConnection> results = mongoTemplate.find(query,

MongoConnection.class);

List<Connection<?>> l = new ArrayList<Connection<?>>();

for (MongoConnection mc : results) {

l.add(converter.convert(mc));

}

return l;

}

}

package org.eluder.spring.social.mongodb;

import java.util.Date;

import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.index.CompoundIndex;

import org.springframework.data.mongodb.core.index.CompoundIndexes;

import org.springframework.data.mongodb.core.mapping.Document;

@Document(collection = "connections")

@CompoundIndexes({

@CompoundIndex(name = "primary", def = "{ 'userId' : 1, 'providerId' : 1, 'providerUserId' : 1 }", unique = true),

@CompoundIndex(name = "rank", def = "{ 'userId' : 1, 'providerId' : 1, 'created' : 1 }", unique = true)

})

public class MongoConnection {

@Id

private String id;

private Date created;

private String userId;

private String providerId;

private String providerUserId;

private String displayName;

private String profileUrl;

private String imageUrl;

private String accessToken;

private String secret;

private String refreshToken;

private Long expireTime;

public String getId() {

return id;

}

public void setId(final String id) {

this.id = id;

}

public Date getCreated() {

return created;

}

public void setCreated(final Date created) {

this.created = created;

}

public String getUserId() {

return userId;

}

public void setUserId(final String userId) {

this.userId = userId;

}

public String getProviderId() {

return providerId;

}

public void setProviderId(final String providerId) {

this.providerId = providerId;

}

public String getProviderUserId() {

return providerUserId;

}

public void setProviderUserId(final String providerUserId) {

this.providerUserId = providerUserId;

}

public String getDisplayName() {

return displayName;

}

public void setDisplayName(final String displayName) {

this.displayName = displayName;

}

public String getProfileUrl() {

return profileUrl;

}

public void setProfileUrl(final String profileUrl) {

this.profileUrl = profileUrl;

}

public String getImageUrl() {

return imageUrl;

}

public void setImageUrl(final String imageUrl) {

this.imageUrl = imageUrl;

}

public String getAccessToken() {

return accessToken;

}

public void setAccessToken(final String accessToken) {

this.accessToken = accessToken;

}

public String getSecret() {

return secret;

}

public void setSecret(final String secret) {

this.secret = secret;

}

public String getRefreshToken() {

return refreshToken;

}

public void setRefreshToken(final String refreshToken) {

this.refreshToken = refreshToken;

}

public Long getExpireTime() {

return expireTime;

}

public void setExpireTime(final Long expireTime) {

this.expireTime = expireTime;

}

}

package org.eluder.spring.social.mongodb;

import static com.google.common.collect.Lists.transform;

import static org.springframework.data.mongodb.core.query.Criteria.where;

import static org.springframework.data.mongodb.core.query.Query.query;

import static org.springframework.data.mongodb.core.query.Update.update;

import java.util.ArrayList;

import java.util.LinkedList;

import java.util.List;

import java.util.Map;

import com.google.common.collect.ImmutableList;

import org.springframework.dao.DuplicateKeyException;

import org.springframework.data.domain.Sort;

import org.springframework.data.mongodb.core.MongoOperations;

import org.springframework.data.mongodb.core.query.Criteria;

import org.springframework.data.mongodb.core.query.Query;

import org.springframework.data.mongodb.core.query.Update;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionFactoryLocator;

import org.springframework.social.connect.ConnectionKey;

import org.springframework.social.connect.ConnectionRepository;

import org.springframework.social.connect.DuplicateConnectionException;

import org.springframework.social.connect.NoSuchConnectionException;

import org.springframework.social.connect.NotConnectedException;

import org.springframework.util.LinkedMultiValueMap;

import org.springframework.util.MultiValueMap;

public class MongoConnectionRepository implements ConnectionRepository {

private final String userId;

private final MongoOperations mongo;

private final ConnectionFactoryLocator connectionFactoryLocator;

private final MongoConnectionTransformers mongoConnectionTransformers;

public MongoConnectionRepository(final String userId, final MongoOperations mongo, final ConnectionFactoryLocator connectionFactoryLocator, final MongoConnectionTransformers mongoConnectionTransformers) {

this.userId = userId;

this.mongo = mongo;

this.connectionFactoryLocator = connectionFactoryLocator;

this.mongoConnectionTransformers = mongoConnectionTransformers;

}

@Override

public MultiValueMap<String, Connection<?>> findAllConnections() {

final Query query = query(where("userId").is(userId)).with(sortByProviderId().and(sortByCreated()));

final List<Connection<?>> results = findConnections(query);

final MultiValueMap<String, Connection<?>> connections = new LinkedMultiValueMap<>();

for (String registeredProviderId : connectionFactoryLocator.registeredProviderIds()) {

connections.put(registeredProviderId, ImmutableList.<Connection<?>>of());

}

for (Connection<?> connection : results) {

final String providerId = connection.getKey().getProviderId();

if (connections.get(providerId).isEmpty()) {

connections.put(providerId, new LinkedList<Connection<?>>());

}

connections.add(providerId, connection);

}

return connections;

}

@Override

public List<Connection<?>> findConnections(final String providerId) {

final Query query = query(where("userId").is(userId).and("providerId").is(providerId)).with(sortByCreated());

return ImmutableList.copyOf(findConnections(query));

}

@Override

@SuppressWarnings("unchecked")

public <A> List<Connection<A>> findConnections(final Class<A> apiType) {

final List<?> connections = findConnections(getProviderId(apiType));

return (List<Connection<A>>) connections;

}

@Override

public MultiValueMap<String, Connection<?>> findConnectionsToUsers(final MultiValueMap<String, String> providerUserIds) {

if (providerUserIds == null || providerUserIds.isEmpty()) {

throw new IllegalArgumentException("providerUserIds must be defined");

}

final List<Criteria> filters = new ArrayList<>(providerUserIds.size());

for (Map.Entry<String, List<String>> entry : providerUserIds.entrySet()) {

final String providerId = entry.getKey();

filters.add(where("providerId").is(providerId).and("providerUserId").in(entry.getValue()));

}

final Criteria criteria = where("userId").is(userId);

criteria.orOperator(filters.toArray(new Criteria[filters.size()]));

final Query query = new Query(criteria).with(sortByProviderId().and(sortByCreated()));

final List<Connection<?>> results = transform(mongo.find(query, MongoConnection.class), mongoConnectionTransformers.toConnection());

MultiValueMap<String, Connection<?>> connectionsForUsers = new LinkedMultiValueMap<>();

for (Connection<?> connection : results) {

final String providerId = connection.getKey().getProviderId();

final String providerUserId = connection.getKey().getProviderUserId();

final List<String> userIds = providerUserIds.get(providerId);

List<Connection<?>> connections = connectionsForUsers.get(providerId);

if (connections == null) {

connections = new ArrayList<>(userIds.size());

for (int i = 0; i < userIds.size(); i++) {

connections.add(null);

}

connectionsForUsers.put(providerId, connections);

}

final int connectionIndex = userIds.indexOf(providerUserId);

connections.set(connectionIndex, connection);

}

return connectionsForUsers;

}

@Override

public Connection<?> getConnection(final ConnectionKey connectionKey) {

final Query query = query(where("userId").is(userId).and("providerId").is(connectionKey.getProviderId()).and("providerUserId").is(connectionKey.getProviderUserId()));

final Connection<?> connection = findOneConnection(query);

if (connection == null) {

throw new NoSuchConnectionException(connectionKey);

} else {

return connection;

}

}

@Override

@SuppressWarnings("unchecked")

public <A> Connection<A> getConnection(final Class<A> apiType, final String providerUserId) {

final String providerId = getProviderId(apiType);

return (Connection<A>) getConnection(new ConnectionKey(providerId, providerUserId));

}

@Override

@SuppressWarnings("unchecked")

public <A> Connection<A> getPrimaryConnection(final Class<A> apiType) {

final String providerId = getProviderId(apiType);

final Connection<A> connection = (Connection<A>) findPrimaryConnection(providerId);

if (connection == null) {

throw new NotConnectedException(providerId);

} else {

return connection;

}

}

@Override

@SuppressWarnings("unchecked")

public <A> Connection<A> findPrimaryConnection(final Class<A> apiType) {

final String providerId = getProviderId(apiType);

return (Connection<A>) findPrimaryConnection(providerId);

}

@Override

public void addConnection(final Connection<?> connection) {

try {

final MongoConnection mongoConnection = mongoConnectionTransformers.fromConnection(userId).apply(connection);

mongo.insert(mongoConnection);

} catch (DuplicateKeyException ex) {

throw new DuplicateConnectionException(connection.getKey());

}

}

@Override

public void updateConnection(final Connection<?> connection) {

final MongoConnection mongoConnection = mongoConnectionTransformers.fromConnection(userId).apply(connection);

final Query query = query(where("userId").is(userId).and("providerId").is(mongoConnection.getProviderId()).and("providerUserId").is(mongoConnection.getProviderUserId()));

final Update update =

update("displayName", mongoConnection.getDisplayName())

.set("profileUrl", mongoConnection.getProfileUrl())

.set("imageUrl", mongoConnection.getImageUrl())

.set("accessToken", mongoConnection.getAccessToken())

.set("secret", mongoConnection.getSecret())

.set("refreshToken", mongoConnection.getRefreshToken())

.set("expireTime", mongoConnection.getExpireTime());

mongo.updateFirst(query, update, MongoConnection.class);

}

@Override

public void removeConnections(final String providerId) {

final Query query = query(where("userId").is(userId).and("providerId").is(providerId));

mongo.remove(query, MongoConnection.class);

}

@Override

public void removeConnection(final ConnectionKey connectionKey) {

final Query query = query(where("userId").is(userId).and("providerId").is(connectionKey.getProviderId()).and("providerUserId").is(connectionKey.getProviderUserId()));

mongo.remove(query, MongoConnection.class);

}

private Connection<?> findPrimaryConnection(String providerId) {

final Query query = query(where("userId").is(userId).and("providerId").is(providerId)).with(sortByCreated());

return findOneConnection(query);

}

private List<Connection<?>> findConnections(Query query) {

return transform(mongo.find(query, MongoConnection.class), mongoConnectionTransformers.toConnection());

}

private Connection<?> findOneConnection(Query query) {

return mongoConnectionTransformers.toConnection().apply(mongo.findOne(query, MongoConnection.class));

}

private <A> String getProviderId(Class<A> apiType) {

return connectionFactoryLocator.getConnectionFactory(apiType).getProviderId();

}

private Sort sortByProviderId() {

return new Sort(Sort.Direction.ASC, "providerId");

}

private Sort sortByCreated() {

return new Sort(Sort.Direction.DESC, "created");

}

}

package org.eluder.spring.social.mongodb;

import java.util.Date;

import com.google.common.base.Function;

import org.springframework.security.crypto.encrypt.TextEncryptor;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionData;

import org.springframework.social.connect.ConnectionFactory;

import org.springframework.social.connect.ConnectionFactoryLocator;

public class MongoConnectionTransformers {

private final ConnectionFactoryLocator connectionFactoryLocator;

private final TextEncryptor textEncryptor;

public MongoConnectionTransformers(final ConnectionFactoryLocator connectionFactoryLocator, final TextEncryptor textEncryptor) {

this.connectionFactoryLocator = connectionFactoryLocator;

this.textEncryptor = textEncryptor;

}

public Function<MongoConnection, String> toUserId() {

return new Function<MongoConnection, String>() {

@Override

public String apply(final MongoConnection input) {

if (input == null) {

return null;

}

return input.getUserId();

}

};

}

public Function<MongoConnection, Connection<?>> toConnection() {

return new Function<MongoConnection, Connection<?>>() {

@Override

public Connection<?> apply(final MongoConnection input) {

if (input == null) {

return null;

}

final ConnectionData cd = new ConnectionData(

input.getProviderId(),

input.getProviderUserId(),

input.getDisplayName(),

input.getProfileUrl(),

input.getImageUrl(),

decrypt(input.getAccessToken()),

decrypt(input.getSecret()),

decrypt(input.getRefreshToken()),

input.getExpireTime()

);

final ConnectionFactory<?> connectionFactory = connectionFactoryLocator.getConnectionFactory(input.getProviderId());

return connectionFactory.createConnection(cd);

}

};

}

public Function<Connection<?>, MongoConnection> fromConnection(final String userId) {

return new Function<Connection<?>, MongoConnection>() {

@Override

public MongoConnection apply(final Connection<?> input) {

if (input == null) {

return null;

}

final ConnectionData cd = input.createData();

final MongoConnection mongoConnection = new MongoConnection();

mongoConnection.setCreated(new Date());

mongoConnection.setUserId(userId);

mongoConnection.setProviderId(cd.getProviderId());

mongoConnection.setProviderUserId(cd.getProviderUserId());

mongoConnection.setDisplayName(cd.getDisplayName());

mongoConnection.setProfileUrl(cd.getProfileUrl());

mongoConnection.setImageUrl(cd.getImageUrl());

mongoConnection.setAccessToken(encrypt(cd.getAccessToken()));

mongoConnection.setSecret(encrypt(cd.getSecret()));

mongoConnection.setRefreshToken(encrypt(cd.getRefreshToken()));

mongoConnection.setExpireTime(cd.getExpireTime());

return mongoConnection;

}

};

}

private String encrypt(final String decrypted) {

if (decrypted == null) {

return null;

}

return textEncryptor.encrypt(decrypted);

}

private String decrypt(final String encrypted) {

if (encrypted == null) {

return null;

}

return textEncryptor.decrypt(encrypted);

}

}

package org.eluder.spring.social.mongodb;

import static com.google.common.base.Preconditions.checkArgument;

import static com.google.common.collect.Lists.transform;

import static org.springframework.data.mongodb.core.query.Criteria.where;

import static org.springframework.data.mongodb.core.query.Query.query;

import java.util.List;

import java.util.Set;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.ImmutableSet;

import org.springframework.data.mongodb.core.MongoOperations;

import org.springframework.data.mongodb.core.query.Query;

import org.springframework.social.connect.Connection;

import org.springframework.social.connect.ConnectionFactoryLocator;

import org.springframework.social.connect.ConnectionKey;

import org.springframework.social.connect.ConnectionRepository;

import org.springframework.social.connect.ConnectionSignUp;

import org.springframework.social.connect.UsersConnectionRepository;

public class MongoUsersConnectionRepository implements UsersConnectionRepository {

private final MongoOperations mongo;

private final ConnectionFactoryLocator connectionFactoryLocator;

private final MongoConnectionTransformers mongoConnectionTransformers;

private ConnectionSignUp connectionSignUp;

public MongoUsersConnectionRepository(final MongoOperations mongo, final ConnectionFactoryLocator connectionFactoryLocator, final MongoConnectionTransformers mongoConnectionTransformers) {

this.mongo = mongo;

this.connectionFactoryLocator = connectionFactoryLocator;

this.mongoConnectionTransformers = mongoConnectionTransformers;

}

public void setConnectionSignUp(final ConnectionSignUp connectionSignUp) {

this.connectionSignUp = connectionSignUp;

}

@Override

public List<String> findUserIdsWithConnection(final Connection<?> connection) {

ConnectionKey key = connection.getKey();

Query query = query(where("providerId").is(key.getProviderId()).and("providerUserId").is(key.getProviderUserId()));

query.fields().include("userId");

List<String> localUserIds = ImmutableList.copyOf(transform(mongo.find(query, MongoConnection.class), mongoConnectionTransformers.toUserId()));

if (localUserIds.isEmpty() && connectionSignUp != null) {

String newUserId = connectionSignUp.execute(connection);

if (newUserId != null) {

createConnectionRepository(newUserId).addConnection(connection);

return ImmutableList.of(newUserId);

}

}

return localUserIds;

}

@Override

public Set<String> findUserIdsConnectedTo(final String providerId, final Set<String> providerUserIds) {

Query query = query(where("providerId").is(providerId).and("providerUserId").in(providerUserIds));

query.fields().include("userId");

return ImmutableSet.copyOf(transform(mongo.find(query, MongoConnection.class), mongoConnectionTransformers.toUserId()));

}

@Override

public ConnectionRepository createConnectionRepository(final String userId) {

checkArgument(userId != null, "userId must be defined");

return new MongoConnectionRepository(userId, mongo, connectionFactoryLocator, mongoConnectionTransformers);

}

}