Formation Micro-services avec Spring Boot

Support de cours : http://demo.dawan.biz/mohamed/spring-boot.pdf

Référence pour apprendre le HQL ou JP-QL :
https://docs.jboss.org/hibernate/orm/3.5/reference/fr-FR/html/queryhql.html
https://docs.oracle.com/javaee/7/tutorial/persistence-querylanguage.htm

dawan.org/files/feuilles-boot-amiens.pdf
dawan.org/files/feuilles-boot-nantes.pdf
dawan.org/files/feuilles-boot-paris.pdf

Web Service : application web sans IHM qui expose un ensemble d'opérations métier
les services web sont inter-opérables
le service web et les clients du WS ne sont pas forcément écrits dans le même langage de programmation
le flux échangé peut être du JSON, XML, Text, binaire

Il existe 2 catégories de services web :
SOAP : protocole basé sur du XML véhiculé sur Http(s), tcp, Middleware Orienté Messages
REST : architecture basée sur des URIs (bases du protocole HTTP), multi-formats : JSON, XML, Text, binaire

Les services web sont sans état (stateless) : entre 2 requêtes, on ne conserve de sessions.
Si on souhaite persister des choses, il faut définir un mécanisme de suivi (jeton ou autre)

Micro-service : service web avec une granularité plus fine
Il faut qu'il traite un périmètre plus petit et doit être indépendant sur ses données.

Construction d'applications web en Java :
- composants de base : Servlet/JSP
- Frameworks web : Spring Web, JSF, Struts

les composants web sont à héberger dans un serveur d'applications Java EE :
Apache Tomcat, Oracle GlassFish, Redhat WebFly, Redhat JBoss, IBM WebSphere,...

Empaquetage et déploiement d'une application Java :
- appli Desktop ou bibliothèques : .jar (Java ARchive)
- appli Web : .war (Web ARchive)
- appli d'entreprise : .ear (Entreprise ARchive) = ensemble de .war et de .jar

---

APIs de développement de services web :
- SOAP => JAX-WS : Java API for Xml Web Services
- REST => JAX-RS : Java API for Xml and Restful Services

Outils de build : Ant, Maven, Gradle
ils permettent de compiler le projet, gérer les dépendances, exécuter les tests, analyser le code, générer la doc,
empaqueter dans une archive et déployer le projet

Un projet Maven est paramétré avec un fichier XML (pom.xml)

Le fichier pom.xml hérite d'un super POM qui contient des configurations par défaut, notamment celle du repository Maven.

Spring : Galaxie de frameworks.

- Il contient plusieurs briques : https://spring.io/projects/spring-framework
- **- Spring Core (IoC) :**
- conteneur d'objets qui instancie des classes au démarrage de l'application et met à disposition à l'objet.
- les objets à instancier dans le conteneur doivent être déclarées soit en Xml (<bean>), soit avec une annotation :
- @Bean, @Component, @Service, @Repository,

- Spring Web : framework MVC pour la construction d'applications web (services web compris)

- Spring Data JPA : framework permettant d'offrir un ORM pour la gestion de la couche de persistance
JPA = Java Persistence API (ensemble d'interfaces)
Il existe plusieurs implémentations de JPA : Hibernate, EclipseLink, OpenJPA,...

- Spring Security : framework permettant de sécuriser une application Spring (client lourd ou léger ou API)

- Spring Boot : framework permettant de simplifier l'initialisation, le paramétrage et le déploiement d'un projet Spring
on peut empaqueter le livrable dans un FAT JAR (qui embarque l'environnement d'exécution)
Exemple :
si on construit une application web => .war déployé sur un serveur d'applications Java EE
on peut demander à Spring Boot d'empaqueter dans un FAT JAR (.war déployé sur le serveur)

Projet Spring Boot :
utilisation du starter pour générer un squelette
le fichier de conf peut être un .properties ou .yml

```
#server.port=8080

#configuration de la datasource
```

```
spring.datasource.initialization-mode=always
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
#org.postgresql.Driver
#com.mysql.cj.jdbc.Driver (Driver MySQL8)

spring.datasource.url=jdbc:mariadb://localhost:3306/myapi1bdd
#si MySQL : jdbc:mysql://localhost:3306/myapi1bdd?
useSSL=false&useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegac
yDatetimeCode=false&serverTimezone=UTC
#si PostgreSQL : jdbc:postgresql://localhost/myapi1bdd

spring.datasource.username=root
spring.datasource.password=

#config de JPA/Hibernate
#dialect : classe permettant de traduire les req vers le SQL spécifique
au SGBD utilisé
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDia
lect
#org.hibernate.dialect.PostgreSQL9Dialect
#org.hibernate.dialect.MySQL8Dialect

#Stratégie utilisée pour générer les tables en Bdd
spring.jpa.hibernate.ddl-auto=update


#config des logs
#TODO : mettre à false en production
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.generate_statistics=false


#Loggers
logging.level.org.springframework.web=info
logging.level.org.springframework.core=info
logging.level.org.springframework.beans=info
logging.level.org.springframework.context=info

logging.level.org.hibernate=info
logging.level.org.hibernate.SQL=info
logging.level.org.hibernate.type=info
logging.level.org.hibernate.tool.hbm2ddl=info
logging.level.org.hibernate.jdbc=info
logging.level.org.hibernate.transaction=info
logging.level.org.hibernate.cache=info
```

Contrôleur REST :

classe java annotée permettant de répondre à des requêtes HTTP : get, post, delete, put,...
chaque méthode indiquera le type MIME de la réponse et peut avoir des paramètres

le contrôleur REST peut être documenté et suivre l'Open API Specification (standard, amené par Swagger)
Mise en place :

- Ajout de la dépendance dans le pom.xml :

```
<dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-ui</artifactId>
        <version>1.5.2</version>
    </dependency>
```

- la doc au format json est accessible sans aucun autre paramétrage à l'url : /v3/api-docs
  on peut personnaliser cette URL en utilisant la propriété :
  springdoc.api-docs.path=/api-docs

- la représentation html du fichier json ou yaml sera accessible en utilisant l'URL :
  /swagger-ui.html

Spring Actuator : outil permettant d'offrir une interface REST pour monitorer le service web
Mise en place :

- copie de la dépendance :

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

- config dans le fichier application.properties

```
#Monitoring du service
```

- #en ajoutant Spring Actuator, voir dependency dans le pom.xml
- #management.server.port=8090
- management.endpoints.web.exposure.include=*
- #actuator vous définira les URLs accessibles ici : http://localhost:8080/actuator

---

Récupération des paramètres d'entête (header) : @RequestHeader
A utiliser pour un seul paramètre ou pour récupérer l'ensemble :

```
@GetMapping(produces = { "application/json", "application/xml" })
    public AgendaDto getAll(@RequestHeader("User-Agent") String userAgent) {
        Logger.getAnonymousLogger().info("get list...");
        System.out.println(userAgent);
```

```
        return new AgendaDto(contactService.getAllContacts());
    }

    @GetMapping(value="/list-headers", produces = "text/plain")
    public String getHeaders(@RequestHeader MultiValueMap<String, String> headers) {
        return headers.toString();
    }

    ou en utilisant la classe HttpHeaders :
    @GetMapping(value="/list-headers2", produces = "text/plain")
    public String getHeaders2(@RequestHeader HttpHeaders headers) {
        return headers.toString();
    }
```

Tout paramètre envoyé dans l'url
(peu importe le mode de passage : @PathVariable ou @RequestParam) est obligatoire par défaut.
Si le param n'est pas reçu, on aura un 405 (Method not allowed)

Path Variable : xxxxxxxxxx/{valeur}
Request Param : xxxxxxxxxxx?nomParam=valeur
Matrix Param : nomPar=valeur1;nomParam2=valeur2

Configuration MVC de l'application Spring Web :

Pour pouvoir personnaliser la configuration, il faut ajouter un "bean" qui retourne un objet de type WebMvcConfigurer

Ce dernier sert à :
- configurer des convertisseurs
- configurer des règles d'autorisation de requêtes multi-domaines (CROSS Origin)
- paramétrer des règles de paths
- ....

```
@Bean
    public WebMvcConfigurer myMvcConfigurer() {

        return new WebMvcConfigurer() {

            //CROSS ORIGIN
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                //registry.addMapping("/api/contacts").allowedMethods("GET").allowedOrigins("*");

//registry.addMapping("/api/contacts").allowedMethods("POST","PUT").allowedOrigins("jehann.fr");
                registry.addMapping("/**").allowedOrigins("*");
            }
```

```
            //CONVERTERS
            @Override
            public void addFormatters(FormatterRegistry registry) {
                registry.addConverter(new StringToContactDtoConverter());
            }

            // ...


            //MATRIX
        };
    }
```

Un Converter en Spring Web permet de traduire un type vers un autre type (alternative à ModelMapper)

Il faut créer une classe qui implémente l'interface Converter<S,D> et définir la méthode Convert :

```
import org.springframework.core.convert.converter.Converter;
import fr.dawan.myapi1.dto.ContactDto;

public class StringToContactDtoConverter implements Converter<String, ContactDto> {

    @Override
    public ContactDto convert(String source) {
        String[] myArray = source.split(",");
        ContactDto dto = new ContactDto();
        dto.setName(myArray[0]);
        dto.setEmail(myArray[1]);
        return dto;
    }

}
--------
le convertisseur peut être utilisé :
  - soit implicitement (par exemple, dans des méthodes du contrôleur) :
    // /test?contact=user2,user2@dawan.fr
    @GetMapping(value="/test", produces="application/json")
    public ContactDto test(@RequestParam("contact") ContactDto coDto) {
        //TODO traitement métier
        return coDto;
    }

  - soit explicitement en injectant (@Autowired) un "ConversionService"
    @Autowired ConversionService conversionService;
```

```
// /test2?contact=user2,user2@dawan.fr
@GetMapping(value="/test", produces="application/json")
public ContactDto test2(@RequestParam("contact") String contactStr) {
```

- ContactDto coDto = conversionService.convert(contactStr, ContactDto.class);
- //TODO traitement métier...

```
    return coDto;
}
```

Gestion des exceptions non capturées dans le contrôleur par le développeur

si une méthode génère des exceptions non capturées :

```
@GetMapping(value="/test-exception", produces = "application/json")
public ContactDto testException() throws Exception {
    //....
    throw new Exception("Erreur blablablabla");
}
```

On peut les traiter de manière plus globale à l'aide d'intercepteurs :

```
    import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.logging.Level;
import java.util.logging.Logger;

import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

import fr.dawan.myapi1.dto.APIError;

@ControllerAdvice
public class MyExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(value = { Exception.class })
    protected ResponseEntity<Object> handleConflict(Exception ex, WebRequest request) {
        HttpHeaders headers = new HttpHeaders();
        // .. ajouter des entêtes dans la rponse si besoin

        // On crée notre propre objet de réponse
        StringWriter sw = new StringWriter();
        PrintWriter pw = new PrintWriter(sw);
```

```
            ex.printStackTrace(pw);
            pw.close();
            Logger.getAnonymousLogger().log(Level.SEVERE, sw.toString());

            APIError myError = new APIError(HttpStatus.INTERNAL_SERVER_ERROR, ex.getMessage());

            return handleExceptionInternal(ex, myError, headers, HttpStatus.INTERNAL_SERVER_ERROR,
request);
      }
}
```

```
import org.springframework.http.HttpStatus;

public class APIError {

      private HttpStatus status;
      private String message;

      public APIError(HttpStatus internalServerError, String message) {
            this.status = internalServerError;
            this.message = message;
      }

      public HttpStatus getStatus() {
            return status;
      }

      public void setStatus(HttpStatus status) {
            this.status = status;
      }

      public String getMessage() {
            return message;
      }

      public void setMessage(String message) {
            this.message = message;
      }

}
```

Tests unitaires et MockMVC

```
package fr.dawan.myapi1;

import static org.assertj.core.api.Assertions.assertThat;
```

```
import static org.assertj.core.api.Assertions.fail;
import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertTrue;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.nio.charset.StandardCharsets;

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;

import com.fasterxml.jackson.databind.DeserializationFeature;
import com.fasterxml.jackson.databind.ObjectMapper;

import fr.dawan.myapi1.controllers.ContactController;
import fr.dawan.myapi1.dto.ContactDto;

@SpringBootTest
@AutoConfigureMockMvc
class Myapi1ApplicationTests {

	@Autowired
	private MockMvc mockMvc;

	@Autowired
	private ContactController contactController;

	@Autowired
	private ObjectMapper objectMapper; // objet permettant de convertir String <=> JSON/XML

	@Test
	void contextLoads() {
		assertThat(contactController).isNotNull();
	}

	@BeforeEach
	public void beforeEach() throws Exception{
```

```
objectMapper.configure(DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT, true);
        objectMapper.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
        SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd");
        objectMapper.setDateFormat(df);
    }

    @Test
    void testFindById() {
        // ici, on passe par la méthode du contrôleur et on ne vérifie pas l'URL en GET
        // avec le param
//          ContactDto cDto = contactController.getById(2);
//          assertEquals("user", cDto.getName());

        try {
            // Exemple 1 : on teste avec jsonPath les valeurs
//              mockMvc.perform(get("/api/contacts/2").accept(MediaType.APPLICATION_JSON))
//                          .andExpect(status().isOk())
//                          .andExpect(jsonPath("$.name", is("user")));

            // Exemple 2 : on récupère la réponse json et on la parse comme on le souhaite
            byte[] arr = mockMvc.perform(
                    get("/api/contacts/2").accept(MediaType.APPLICATION_JSON)
                    .characterEncoding("utf-8"))
                    .andExpect(status().isOk()).andReturn().getResponse()
                    .getContentAsByteArray();
            String jsonRep = new String (arr,StandardCharsets.UTF_8);

            ContactDto cDto = objectMapper.readValue(jsonRep, ContactDto.class);
            assertEquals("user", cDto.getName());
            assertEquals("user@dawan.fr", cDto.getEmail());

        } catch (Exception e) {
            fail(e.getMessage());
        }
    }

    @Test
    void testFindAll() {
    try {
        mockMvc.perform(get("/api/contacts").accept(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$[0].name", is("user")));
    } catch (Exception e) {
        fail(e.getMessage());
    }
```

```
	}

	@Test
	void testSave() {
		try {
			ContactDto coToInsert = new ContactDto();
			coToInsert.setName("myNewCo");
			coToInsert.setEmail("myNewCo@dawan.fr");
			coToInsert.setPassword("myNewPwd");


objectMapper.configure(DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT, true);
			String jsonReq = objectMapper.writeValueAsString(coToInsert);

			byte[] jsonReponseBytes = mockMvc.perform(
							post("/api/contacts")			//VERB + URI
								.contentType(MediaType.APPLICATION_JSON)
//consumes MIME TYPE
								.characterEncoding("utf-8")
								.accept(MediaType.APPLICATION_JSON)
//produces MIME TYPE
								.content(jsonReq)
						)
						.andExpect(status().isOk())
						.andReturn().getResponse().getContentAsByteArray();
			String jsonReponse = new String (jsonReponseBytes,StandardCharsets.UTF_8);

			ContactDto cDto = objectMapper.readValue(jsonReponse, ContactDto.class);
			assertTrue(cDto.getId()!=0);

		} catch (Exception e) {
			fail(e.getMessage());
		}
	}

	@Test
	void testUpdate() {

	}

	@Test
	void testDelete() {

	}

	@Test
```

```
    void testCount() {

    }

}
```

Soit une classe avec testM1 et testM2
L'ordre des exécutions de méthodes est le suivant :

@BeforeAll (méthode statique pour initialiser des variables utilisées dans les différents tests)

@BeforeEach
@Test : testM1
@AfterEach

@BeforeEach
@Test : testM2
@AfterEach

@AfterAll

Hash Password
le hachage génère une emprunte non décryptable
Méthodes (algorithmes) de hachage : MD5, SHA-1, SHA-256, SHA-512

```
import java.math.BigInteger;
import java.security.MessageDigest;

public class HashTools {

    public static String hashSHA512(String input) throws Exception{
        //getInstance récupère le singleton MessageDigest
        MessageDigest md = MessageDigest.getInstance("SHA-512");

        //méthode pour réinitialiser le contenu du messageDigest s'il a été utilisé auparavant
        md.reset();

        //application de l'algorithme de hachage à la chaine en entrée
        byte[] messageDigestArray = md.digest(input.getBytes("utf-8"));

        //conversion du messageDigestArray en une réprésentation numérique signée
        BigInteger bi = new BigInteger(1, messageDigestArray);

        return String.format("%0128x", bi);
```

```
    }
}
```

Méthode du service à modifier (ContactServiceImpl) :

```
   @Override
      public ContactDto saveOrUpdate(ContactDto cDto) {
            Contact c = DtoTools.convert(cDto, Contact.class);
            try {
                  c.setPassword(HashTools.hashSHA512(c.getPassword()));
            } catch (Exception e) {
                  e.printStackTrace();
            }
            c = contactRepository.saveAndFlush(c);
            return DtoTools.convert(c, ContactDto.class);
      }
```

Download de fichiers

- conserver le chemin vers le fichier qlq part. Ici, on utilise imagePath dans la table "contact"

- on peut si on le souhaite définir un répertoire pour stocker les pièces jointe :

```
#app properties
app.storagefolder=C:/tempBidon/
```

toute propriété peut être récupérée avec l'annotation ***@Value*** :

```
@Value("${app.storagefolder}")
private String storagefolder;
```

Le téléchargement d'un fichier génère un flux octets qui peut être très volumineux.
Toute requête générant une réponse conséquente peut être encapsulée dans un ResponseEntity<**StreamingReponseBody**> au lieu du ResponseEntity<**Resource**>.
Vu que le traitement de téléchargement peut prend du temps, on peut l'embarquer dans une tâche asynchrone (Executors, disponibles depuis Java 5) :

https://medium.com/swlh/streaming-data-with-spring-boot-restful-web-service-87522511c071

//download file :

```
//le retour peut être Resource ou InputStream.
//on peut également faire du StreamingResponseBody dans le cas d'un
fichier hyper //volumineux
//si le fichier est volumineux, il vaut mieux exécuter le téléchargement
```

```
en asynchrone
//TaskExecutor
@GetMapping(value="/download/{id}")
public ResponseEntity<Resource>
downloadImageByContactId(@PathVariable("id")long id) throws Exception{

ContactDto c = contactService.getById(id);

File f = new File(storagefolder + c.getImagePath());
Path path = Paths.get(f.getAbsolutePath());
ByteArrayResource resource = new
ByteArrayResource(Files.readAllBytes(path));

HttpHeaders headers = new HttpHeaders();
headers.add(HttpHeaders.CONTENT_DISPOSITION,
"attachment;filename=\""+c.getImagePath()+"\"");
headers.add("Cache-Control","no-cache, no-store, must-revalidate");
headers.add("Pragma","no-cache");
headers.add("Expires","0");

return ResponseEntity.ok()
.headers(headers)
.contentLength(f.length())
.contentType(MediaType.APPLICATION_OCTET_STREAM)
.body(resource);
}
```

Upload de pièces jointes :

On peut uploader 1 ou plusieurs pièces jointes dans une requête HTTP multipart-data

Il faut dans un premier temps configurer notre application pour accepter les pièces jointes.
Soit avec l'annotation ***@MultipartConfig*** dans le contrôleur concerné ou mieux (plus global), dans le fichier application.properties

```
## MULTIPART (MultipartProperties)
# Enable multipart uploads
spring.servlet.multipart.enabled=true
# Threshold after which files are written to disk.
spring.servlet.multipart.file-size-threshold=2KB
# Max file size.
spring.servlet.multipart.max-file-size=10MB
# Max Request Size
spring.servlet.multipart.max-request-size=215MB
```

- Le contrôleur va accepter des paramètres qui seront récupérés dans un MultipartFile :

```
@PostMapping(value="/save-file", produces = "text/plain",
consumes="multipart/form-data")
public String uploadFile(@RequestParam("file") MultipartFile file) {
```

- try {

```
File f = new File(storagefolder + file.getOriginalFilename());
```

- try(BufferedOutputStream bw = new BufferedOutputStream(new FileOutputStream(f))){
- bw.write(file.getBytes());

```
}
return "OK";
```

- }

```
catch (Exception e) {
```

- e.printStackTrace();
- return "KO";
- }

```
}
```

// POST WITH IMAGE

```
@PostMapping(value = "/save-with-image", consumes = "multipart/form-
data", produces = "application/json")
public ContactDto saveWithImage(@RequestParam("contact") String
contactStr, @RequestPart("file") MultipartFile file) throws Exception {
```

- ContactDto cDto = objectMapper.readValue(contactStr, ContactDto.class);

```
File f = new File(storagefolder + file.getOriginalFilename());
```

- try (BufferedOutputStream bw = new BufferedOutputStream(new FileOutputStream(f))) {
- bw.write(file.getBytes());

```
}
cDto.setImagePath(file.getOriginalFilename());
```

- return contactService.saveOrUpdate(cDto);

```
}
```

Une image peut être encodé en base64 et on aura une simple chaine de caractères qu'on pourra stocker directement
en base de données sans faire de FileUpload.

Jeton JWT

La première requête contient le login/password et va générer un token (valide un certain temps)
=> il existe un standard : JSON Web Token (https://jwt.io/) mais on peut définir notre propre stratégie
les tokens peuvent être stockés en Bdd si besion

L'intérêt est de véhiculer un token dans les requêtes afin de contrôler l'utilisateur et ses droits
Un intercepteur est développé pour vérifier le header Authorization et analyser le token.

Mise en place :

1) ajout des dépendances : Spring Security et JWT :

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>io.jsonwebtoken</groupId>
        <artifactId>jjwt</artifactId>
        <version>0.9.1</version>
    </dependency>
```

2) ajouter une classe de configuration pour Spring Security :
class XX extends WebSecurityConfigurerAdapter

Client Java (RestTemplate)

CSRF : Cross-Site Request Forgery (XSRF)
Faille de sécurité (authentification web) - usurpation d'identité pour exécuter une action malveillante
https://www.ionos.fr/digitalguide/serveur/securite/cross-site-request-forgery/

Gestion des logs en Java

On peut utiliser :

- soit l'api native : java.util.logging

- soit utiliser une API externe : Log4j, Logback, ...
  Avantages : multiples configurations de supports de logs

Vocabulaire :

Logger : objet permettant d'écrire les logs
Appender : support de logs (on peut associer plusieurs appenders au même logger)
logger : root => 2 appenders : console, file1, bdd, ...
ConsoleAppender, FileAppender, RollingFileAppender, ...

Layout : Type du message (Text, XML)
PatternLayout => Text
XmlLayout => Xml
Pattern : Format du message (%date %nomclasse %niveau du message %message %saut de ligne)
%d{format} %p %C %m %n
Level : Niveau du message (INFO, WARN, ERROR,...)

Client de web service :

- JavaScript :
pure : XmlHttpRequest
jQuery : $.ajax(.....)
Vue.js : Axios
Angular : HttpClient
React : méthode fetch(url)
- Java
- Autre langage

Java :
api native : HttpURLConnection
framework :
* commons http : HttpClient
* Spring web : classe RestTemplate ou WebClient

Exemple de client Java : (en utilisant RestTemplate)

```
RestTemplate restTemplate = new RestTemplate();

objectMapper.configure(DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT, true);

List<ContactDto> lstResult = null;
URI url = null;
try {
    url = new URI(AppConstants.MYAPI_BASE_URL+ "/api/contacts/" + page + "/" + size);
    ResponseEntity<String> rep = restTemplate.getForEntity(url, String.class);
    if (rep.getStatusCode().equals(HttpStatus.OK)) {
        String json = rep.getBody();
        ContactDto[] resArray = objectMapper.readValue(json, ContactDto[].class);
        lstResult = Arrays.asList(resArray);
    }

} catch (Exception e) {
    e.printStackTrace();
}

return lstResult;
```

```
}
```

Exemple de post :

```
   // Appel au service web
        RestTemplate restTemplate = new RestTemplate();

objectMapper.configure(DeserializationFeature.ACCEPT_EMPTY_STRING_AS_NULL_OBJECT, true);

        URI url = null;
        try {
            url = new URI(AppConstants.MYAPI_BASE_URL+ "/authenticate");
            String jsonReq = objectMapper.writeValueAsString(obj);

            ResponseEntity<String> rep = restTemplate.postForEntity(url, jsonReq, String.class);

            if (rep.getStatusCode().equals(HttpStatus.OK)) {
                String json = rep.getBody();
                LoginResponseDto repDto = objectMapper.readValue(json, LoginResponseDto.class);
                return repDto;
            }

        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;
}
```

Activation des matrixParam
(implements WebMvcConfigurer)

Applications CFA (Centre de formations pour apprentis)

(NANTES) SYDAPHASAVANH Kanha : Entities OK / Repositories / services / contrôleurs

(NANTES) BILLON Tanguy Entities OK (M) / Repositories / services / contrôleurs OK test en cours

(PARIS) ATIA Ali-Haïdar Entities OK (M) / Repositories / services / contrôleurs

(PARIS) POTIER Nicolas Entities OK(M) / Repositories / services / contrôleurs

Modèle métier :

Etudiant : id, nom, liste de cours où il est inscrit

Cours : id, titre, contenu, many Cours to One Formateur, étudiants qui suivent le cours

Formateur : id, nom, one To Many Cours

GET : les cours animés par un formateur byName
/Cours/search?formateur=Mourad

GET : /formateurs/{idFormateur}

GET : /cours/{idcours}/etudiants

GET : /formateurs/{idformateur}/cours

POST : insérer un cours

POST : insérer un formateur

POST : insertion d'un étudiant

POST : inscription d'un étudiant à un cours

Applications d'évaluations de connaissances (Évolution de : https://dwskills.dawan.fr/)

(NANTES) HOUIMLI Yosra

(NANTES) STEPHANT Erwan Entités : OK/ Repositories OK/ services OK/ controllers: bientôt terminé

(NANTES) BILLARD Vincent Entités : OK / Repositories OK / services /methodes controllers : 6/8 OK (M) /

(NANTES) ROGEZ Fredy Entités : OK / Repositories+services+controllers : 90%

Modèle métier :

Quiz : id, subject, plusieurs questions

Question (plusieurs questions pour plusieurs quiz) , multiple?, numOrder

Response: many réponse to one question

QuizTest : id, creationDate, score, Quiz, Contact, Map<Question,List<Response> historique des

réponses

WS :

GET /quizzes (liste des quiz)
POST /quizzes

GET /quizzes/{idquiz}/questions/{numOrder} ou /questions?quizId=xx&numOrder=yyy
GET /quizzes/{idquiz}/questions/count ou /questions/count?quizId=xx
GET : /responses/{idrep}

POST /questions
POST /responses
POST /quiztest

Scanner de CV avec IA pour proposer des formations qui conviennent au profil

(NANTES) LAUTRU Fabien Entities OK (M) / Repositories OK / services OK / controleur 80%
(NANTES) FAUDUET Maxime
(NANTES) GODINEZ ARANA Laura Daniela Entities OK / Repositories OK / services OK / controleur 40%
(NANTES) MALPEL Simon Entities OK / Repositories OK / Services OK / contrôleur ~en cours~

Modèle :

CV : id, scanDate, sections, keywords, many CVs to One Contact
Section : id, titre, contenu, keywords, many sections to One CV
Contact : id, email
Job : id, titre, contenu

GET : /cvs/{id}/sections
GET : /contacts/{idcontact}/cvs
GET : /sections/{id}/keywords

POST :
insertion contact
insertion section
insertion cv

Application de génération de planning (.NET/365) avec API externalisées en Java (formations/interventions/formateurs)

(PARIS) DA SILVA LIMA Jade Entities OK (M) / Repositories OK / Serices OK / Controllers Ok / Missing tests for Intervention
(PARIS) NGUYEN Valentin Entities OK (M) / Controllers en cours
(PARIS) MAURY Romain Entities OK (M)
(PARIS) MOKHTARI Ahmed Reda Entities OK (M)

Formation : id, titre
Employe (Formateur/Autre) : id, nom, type(Formateur/Commercial/Administratif/Indépendant)
Intervention (Cours/Conférence) : id, dateDebut, dateFin, many interv to one Employe,
many interv. to one Formation
OperationModif : id, ancienneValeur, nouvValeur (voir Design Pattern "Memento")

Get :
/formations
/formations/{id}
/interventions?formation=xxxx (titre ou id)

POST :
OperationModif ==>

Applications de gestion d'offres emploi et candidatures
(PARIS) DOUCOURE Alfoussein Entites ok Repo en cours et service en cours avec le controller
(PARIS) JANET Sylvain Entities OK (M) / Repositories OK / Services OK / Controllers OK
OK Pour GET OK Pour POST
Ajout des GETALL(paginé), GETBYID, DELETE et COUNT pour chaque classe OK
GenericService OK / GenericController OK
TESTS CONTROLLERS CRUD OK
TESTS CONTROLLERS GET spécifiques métier OK

- InitDb pour tests sql -> java OK

(PARIS) PEROU Aser Amagana Entities Ok(M), Contact & OffreEmploi (Service+Repo+Controller OK)
(PARIS) KUCUK Denis Entities OK (M) , OffreEmploiController (Ok) , CandidatureController (en cours,.. j'aurais une question)

Modèle Métier
Contact : id, nom, email
OffreEmploi : id, titre, contenu, dateCreation, datePrisePoste, lieu
Candidature : id, many candidatures to one OffreEmploi, many candidatures to one Contact
Entretien : id, creationDate, rapport, many entretiens to one candidature

GET :
/offres/{idoffre}/candidatures ou /candidatures?offreid=xxx
/entretiens?idcontact=xxx
/offres?lieu=xxx

POST :
Contact
Offre
Candidature

Entretien

Applications de gestion RH

(Amiens) Corentin HERDUIN **Entities OK (M)**

(Amiens) Victor ANDRÉ Entities OK (M) | Respositories OK | Services OK | Controllers OK

appli qui gère les entretiens annuels des salariés et leurs évolutions

Salarie : id, nom, many salariés pour un one Salarié manager
Poste : id, titre, dateDebut, dateFin, typeContrat (CDI, CDD, APPRENTISSAGE), many poste to one salarié,
Entretien :id, dateEntretien, many entretiens to one salarie, compteRendu

POST :
insertion d'un salarié OK
insertion d'un poste OK
insertion d'un entretien OK

GET :
/salaries/current : salariés qui sont en postes aujourd'hui OK
/salaries/{id} OK
/postes?idsalarie=xxxx OK
/entretiens?idsalarie=xxx OK

- Application de gestion RH : avec comme fonctionnalités la gestion des entretiens annuels du salarié et de son parcours de formations en interne/externe
- Des salariés ont une fiche avec id, nom, prenom, adresse, ..
- Plusieurs "postes" peuvent être associés au salarié depuis son embauche, chaque poste contient : date d'embauche, type de contrat (CDI, CDD, ...), date de départ, volume horaire hebdomadaire, salaire, pièce jointe pour le contrat, lieu de travail (centre)..
- Chaque poste peut avoir un manager hiérarchique qui est lui même un salarié
- Chaque salarié à des compétences (dév C#, gestion projets, etc..)
- On peut lui faire suivre des formations qui vont lui faire gagner des compétences

- *Un module pour créer les entretiens annuels : notification 2 semaines avant la date d'entrée au manager et salarié, formulaire de saisi pour le salarié, formulaire de saisi pour le manager, signature électronique, fonction d'envoi du compte-rendu, rappel si le compte-rendu n'est pas signé*

- *Un module pour créer les Plans annuels d'activités (selon format excel actuel)*
- *Un module pour demander des formations*

- Des fonctionnalités pour sauvegarder les entretiens annuels et les comptes rendus associés.
- Un tableau de bord pour les RHs pour visualiser les salariés formés, les futurs entretiens annuels,....

Modèle métier :
- Company : int? id, string name, Address address
- Address : int num, string street, string city, int zipCode, string country
- User : ajouter : string ProPhone, + une énumération UserType { ADMIN, EMPLOYEE, MANAGER, RH}, UserType type
- Employee hérite de User : DateTime BirthDate, Address PersonalAddress, string PersonalPhone, Company company, bool isManager, list<Compétence> compétences, list<formation> Formations, Poste poste
- Poste : int? id, DateTime DateEmbauche, enum typeContrat{CDI, CDD, APPRENTISSAGE, ALTERNANCE, CONTRAT_PRO}, DateTime dateDepart, double volumeHoraire, stringPJ, Company lieuTravail, Employee Manager, list<Compétence> compétencesRequired, List<Employee> lstEmployee
- Compétences : int? id, List<Employee> lstEmployee , string competence
- Formation : int? id, DateTime DateDepart, DateTime DateFin, double volumeHoraire, double Prix, List<User>employeeInscrits, list<Competence> trainedCompetence
- Entretien : int? id, CompteRendu compteRendu, List<Employee>? employee, Employee Manager, Employee RH
- CompteRendu :int? id, string Content

-

```
import java.io.Serializable;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "login")
@XmlAccessorType(XmlAccessType.FIELD)
public class LoginDto implements Serializable{

	private String email;
	private String password;
```

```
	public LoginDto(String email, String password) {
		super();
		this.email = email;
		this.password = password;
	}
	public LoginDto() {
		super();
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
}
```

```
package fr.dawan.myapi1.dto;

import java.io.Serializable;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "login-reponse")
@XmlAccessorType(XmlAccessType.FIELD)
public class LoginResponseDto implements Serializable{

	private String token;

	public LoginResponseDto() {

	}


	public LoginResponseDto(String token) {
		super();
		this.token = token;
	}
```

```
	public String getToken() {
		return token;
	}

	public void setToken(String token) {
		this.token = token;
	}


}
```

```
package fr.dawan.myapi1.services;

import java.util.ArrayList;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import fr.dawan.myapi1.entities.Contact;
import fr.dawan.myapi1.repositories.ContactRepository;

@Service
public class JwtUserDetailsService implements UserDetailsService {

	@Autowired
	private ContactRepository contactRepository;

	@Override
	public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
		Contact co = contactRepository.findByEmail(username);
		if(co==null)
			throw new UsernameNotFoundException("User not found with username : " + username);

		return new User(co.getEmail(), co.getPassword(), new ArrayList<>());
	}

}
```

```
package fr.dawan.myapi1.interceptors;
```

```
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import fr.dawan.myapi1.services.JwtUserDetailsService;
import fr.dawan.myapi1.tools.JwtTokenUtil;

@Component
public class LoginFilter extends OncePerRequestFilter {

	@Autowired
	private JwtUserDetailsService jwtUserDetailsService;

	@Autowired
	private JwtTokenUtil jwtTokenUtil;

	@Override
	protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain
filterChain)
			throws ServletException, IOException {
		String headerAuth = request.getHeader("Authorization");
		if (headerAuth == null || headerAuth.trim().equals("") || !headerAuth.startsWith("Bearer ")) {
			Logger.getAnonymousLogger().log(Level.INFO, "Erreur : jeton absent !");
		} else {
			// Découper le headerAuth pour retirer Bearer xxxxx
			String token = headerAuth.substring(7);
			System.out.println("token = " + token);
			// TODO vérification du token
			String username = null;
			try {
				username = jwtTokenUtil.getUsernameFromToken(token);
				if (username != null && SecurityContextHolder.getContext().getAuthentication() ==
null) {
```

```
                        UserDetails userDetails = jwtUserDetailsService.loadUserByUsername(username);
                        if (jwtTokenUtil.validateToken(token, userDetails)) {
                            UsernamePasswordAuthenticationToken usernamePwdAuthToken = new
UsernamePasswordAuthenticationToken(
                                    userDetails, null, userDetails.getAuthorities());
                            usernamePwdAuthToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                            // spécifier au contexte que l'utilisateur en cours est bien authentifié

SecurityContextHolder.getContext().setAuthentication(usernamePwdAuthToken);
                        }
                    }
                } catch (Exception e) {
                    Logger.getAnonymousLogger().log(Level.INFO, "JWT Token Invalid !");
                }
            }
            filterChain.doFilter(request, response);
        }

}
```

```
package fr.dawan.myapi1.tools;

import java.io.Serializable;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

@Component
public class JwtTokenUtil implements Serializable {

    private static final long serialVersionUID = -2550185165626007488L;

    public static final long JWT_TOKEN_VALIDITY = 5*60*60;

    @Value("${jwt.secret}")
```

```
	private String secret;

	public String getUsernameFromToken(String token) {
		return getClaimFromToken(token, Claims::getSubject);
	}

	public Date getIssuedAtDateFromToken(String token) {
		return getClaimFromToken(token, Claims::getIssuedAt);
	}

	public Date getExpirationDateFromToken(String token) {
		return getClaimFromToken(token, Claims::getExpiration);
	}

	public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {
		final Claims claims = getAllClaimsFromToken(token);
		return claimsResolver.apply(claims);
	}

	private Claims getAllClaimsFromToken(String token) {
		return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
	}

	private Boolean isTokenExpired(String token) {
		final Date expiration = getExpirationDateFromToken(token);
		return expiration.before(new Date());
	}

	private Boolean ignoreTokenExpiration(String token) {
		// here you specify tokens, for that the expiration is ignored
		return false;
	}

	public String generateToken(UserDetails userDetails) {
		Map<String, Object> claims = new HashMap<>();
		return doGenerateToken(claims, userDetails.getUsername());
	}

	private String doGenerateToken(Map<String, Object> claims, String subject) {

		return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new
Date(System.currentTimeMillis()))
				.setExpiration(new Date(System.currentTimeMillis() +
JWT_TOKEN_VALIDITY*1000)).signWith(SignatureAlgorithm.HS512, secret).compact();
	}
```

```
	public Boolean canTokenBeRefreshed(String token) {
		return (!isTokenExpired(token) || ignoreTokenExpiration(token));
	}

	public Boolean validateToken(String token, UserDetails userDetails) {
		final String username = getUsernameFromToken(token);
		return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
	}
}
```

_____application.properties_____

```
jwt.secret=myapisecretkey123
jwt.get.token.uri=/authenticate
```

_____

```
package fr.dawan.myapi1;

import java.io.IOException;
import java.io.Serializable;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

@Component
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint, Serializable {

	@Override
	public void commence(HttpServletRequest request, HttpServletResponse response,
			AuthenticationException authException) throws IOException, ServletException {

		response.sendError(HttpServletResponse.SC_UNAUTHORIZED,"Unauthorized");
	}
}
```

_____

```
package fr.dawan.myapi1;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import
```

```
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import fr.dawan.myapi1.interceptors.LoginFilter;
import fr.dawan.myapi1.services.JwtUserDetailsService;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter{

	@Autowired
	private LoginFilter loginFilter;

	@Autowired
	private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

	@Autowired
	private JwtUserDetailsService jwtUserDetailsService;

	@Autowired
	public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
		auth.userDetailsService(jwtUserDetailsService); //.passwordEncoder(passwordEncoder());
	}

	@Bean
	public PasswordEncoder passwordEncoder() {
		return new BCryptPasswordEncoder();
	}


	@Bean
	@Override
	public AuthenticationManager authenticationManagerBean() throws Exception {
		return super.authenticationManagerBean();
	}
```

```
	@Override
	protected void configure(HttpSecurity http) throws Exception {
		//désactiver le CSRF et ne pas demander d'authentification pour cette requête
		http
			.csrf().disable()
			.authorizeRequests().antMatchers("/authenticate").permitAll() //on ignore l'authentif pour
celle-là
			.anyRequest().authenticated() //toute requête doit être authentifié
			.and()
			.exceptionHandling().authenticationEntryPoint(jwtAuthenticationEntryPoint) //gestion des
exceptions (traitement à réaliser dans la classe JWTAuth...)
			.and()
			.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); //les
sessions ne seront pas utilisées

		//ajout d'un filtre pour valider les tokens sur chaque requête
		http.addFilterBefore(loginFilter, UsernamePasswordAuthenticationFilter.class);
	}

}
```

```
package fr.dawan.myapi1.controllers;

import java.util.Objects;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import fr.dawan.myapi1.dto.ContactDto;
import fr.dawan.myapi1.dto.LoginDto;
import fr.dawan.myapi1.dto.LoginResponseDto;
import fr.dawan.myapi1.services.ContactService;
import fr.dawan.myapi1.tools.HashTools;
import fr.dawan.myapi1.tools.JwtTokenUtil;
```

```
@RestController
public class LoginController {

//	@Autowired
//	private ContactService contactService;

	@Autowired
	private AuthenticationManager authenticationManager;

	@Autowired
	private UserDetailsService jwtInMemoryUserDetailsService;

	@Autowired
	private JwtTokenUtil jwtTokenUtil;

	@PostMapping(value="/authenticate", consumes = "application/json")
	public ResponseEntity<?> checkLogin(@RequestBody LoginDto loginObj) throws Exception{
//		ContactDto cDto = contactService.findByEmail(loginObj.getEmail());
//		//TODO crypter le mot de passe reçu pour comparer avec le hash en Bdd
//		if(cDto !=null && cDto.getPassword().contentEquals(loginObj.getPassword())) {
//			//TODO generate token (change to JWT)
//			String token = HashTools.hashSHA512(loginObj.getEmail()+"@dawanXE45");
//			//TODO stoker le token/email qlq part pour qu'on puisse le vérifier dans les contrôleur
//			return ResponseEntity.ok(new LoginResponseDto(token));
//		}else
//			throw new Exception("Erreur : identifiants incorrects !");


		//authentification
		authenticate(loginObj.getEmail(), loginObj.getPassword());

		UserDetails userDetails =
jwtInMemoryUserDetailsService.loadUserByUsername(loginObj.getEmail());

		//générer le token
		String token = jwtTokenUtil.generateToken(userDetails);

		//retourner la réponse
		return ResponseEntity.ok(new LoginResponseDto(token));

	}


	private void authenticate(String username, String password) throws Exception{
		Objects.requireNonNull(username);
		Objects.requireNonNull(password);
```

```
            try {
                authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username,
password));
            }catch(Exception e) {
                throw new Exception("Authentification incorrecte !",e);
            }

        }

}
```