Formation Spring BOOT

TP : Sécuriser un service web REST via JWT en utilisant Spring BOOT, JJWT, Spring Data JPA et MySQL

SOMMAIRE

I- Objectifs :

- ✓ Sécuriser une API REST avec Spring Security et JWT.

II- Outils utilisés :

Dans ce TP, nous allons utiliser les outils suivants :

- ✓ Eclipse avec le plugin Maven ;
- ✓ JDK 1.8 ;
- ✓ Connection à Internet pour permettre à Maven de télécharger les dépendances nécessaires.

II Développement de l'application

1. Création du projet avec Spring Initializr

- ❖ Lancer Spring Initializr et créer un projet java avec les dépendances suivantes :

Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

Spring Boot DevTools DEVELOPER TOOLS

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Spring Security SECURITY

Highly customizable authentication and access-control framework for Spring applications.

MySQL Driver SQL

MySQL JDBC and R2DBC driver.

- Ajouter ensuite au niveau du pom.xml les deux dépendances suivantes :

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

Cette dépendance est nécessaire pour pourvoir générer le token JWT, le valider et le parser.

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

Cette dépendance est nécessaire pour pouvoir utiliser l'API Bean validation (@Valid). Spring BOOT utilise l'implémentation de Hibernate Validator.

2. Le fichier pom.xml

Le contenu de votre fichier pom.xml devrait être le suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.5.7</version>
		<relativePath />
	</parent>
	<groupId>ma.formations</groupId>
	<artifactId>exemplejwtwithspringsecurity</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>exemplejwtwithspringsecurity</name>
	<description>Securiser une service web REST moyennant Spring Security et JWT</description>
	<properties>
		<java.version>1.8</java.version>
	</properties>
	<dependencies>
	 <dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-security</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
			<optional>true</optional>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
```

```
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>io.jsonwebtoken</groupId>
            <artifactId>jjwt</artifactId>
            <version>0.9.1</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-validation</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

```
</project>
```

3. Le fichier application.properties(*)

```
spring.datasource.url=jdbc:mysql://localhost:3306/db_springsecurity_jwt?createDatabaseIfNotExist=true&autoReconnect=true&useSSL=true&useUnicode=yes&useLegacyDatetimeCode=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
app.jwtSecret= @zeRtY193!
#1*24*60*60*1000; 1 jours
app.jwtExpirationMs= 86400000
```

(*)N'oublier pas de configurer : votre URL,votre login et votre password

4. Les classes modèles User, Role et Emp

❖ Créer les classes Role, User et Emp au niveau du package ma.formations.service.model :

```
package ma.formations.service.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Entity
@Table(name = "role")
@NoArgsConstructor
public class Role {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	@Column(name = "role_id")
	private int id;
	@Column(name = "role")
```

```
	private String role;

	public Role(String role) {
		this.role = role;
	}
}
```

```
package ma.formations.service.model;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
import javax.validation.constraints.NotEmpty;

import org.hibernate.validator.constraints.Length;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Entity
@Table(name = "user")
@NoArgsConstructor
public class User {
	@Id
	@GeneratedValue
	private Long id;

	@Length(min = 5, message = "*Your username must have at least 5 characters")
	@NotEmpty(message = "*Please provide an user name")
	private String username;

	@Length(min = 5, message = "*Your password must have at least 5 characters")
	@NotEmpty(message = "*Please provide your password")
	private String password;

	@ManyToMany(cascade = CascadeType.ALL)
	@JoinTable(name = "user_role", joinColumns = @JoinColumn(name = "user_id"),
```

```
inverseJoinColumns = @JoinColumn(name = "role_id"))
        private List<Role> roles = new ArrayList<Role>();
}
```

```
package ma.formations.service.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@Entity
public class Emp {
        @Id
        @GeneratedValue
        private Long id;
        private String name;
        private Double salary;
        private String fonction;

        public Emp(String name, Double salary, String fonction) {
                super();
                this.name = name;
                this.salary = salary;
                this.fonction = fonction;
        }
}
```

5. Les classes VO

❖ Créer les classes RoleVo, UserVo, EmpVo, RoleConverter, UserConverter et EmpConverter au niveau du package ma.formations.domaine

```
package ma.formations.domaine;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
```

```java
@NoArgsConstructor
public class RoleVo {
	private int id;
	private String role;

	public RoleVo(String role) {
		this.role = role;
	}
}
```

```java
package ma.formations.domaine;

import java.util.ArrayList;
import java.util.List;

import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
public class UserVo {
	private Long id;
	private String username;
	private String password;
	private List<RoleVo> roles = new ArrayList<RoleVo>();

	public UserVo(String username, String password, List<RoleVo> roles) {
		this.username = username;
		this.password = password;
		this.roles = roles;
	}
}
```

```java
package ma.formations.domaine;

import lombok.Data;

@Data
public class EmpVo {
	private Long id;
	private String name;
	private Double salary;
	private String fonction;
	public EmpVo() {
		super();
	}
	public EmpVo(Long id, String name, Double salary, String fonction) {
		this(name,salary,fonction);
		this.id = id;
```

```
	}

	public EmpVo(String name, Double salary, String fonction) {
		super();
		this.name = name;
		this.salary = salary;
		this.fonction = fonction;
	}
}
```

```
package ma.formations.domaine;

import java.util.ArrayList;
import java.util.List;

import ma.formations.service.Role;

public class RoleConverter {

	public static RoleVo toVo(Role bo) {
		if (bo == null)
			return null;
		RoleVo vo = new RoleVo();
		vo.setId(bo.getId());
		vo.setRole(bo.getRole());
		return vo;
	}

	public static Role toBo(RoleVo vo) {
		if (vo == null)
			return null;
		Role bo = new Role();
		bo.setId(vo.getId());
		bo.setRole(vo.getRole());
		return bo;
	}

	public static List<RoleVo> toVoList(List<Role> boList) {
		if (boList == null || boList.isEmpty())
			return null;
		List<RoleVo> voList = new ArrayList<>();
		for (Role role : boList) {
			voList.add(toVo(role));
		}
		return voList;
	}
```

```
	public static List<Role> toBoList(List<RoleVo> voList) {
		if (voList == null || voList.isEmpty())
			return null;
		List<Role> boList = new ArrayList<>();
		for (RoleVo roleVo : voList) {
			boList.add(toBo(roleVo));
		}
		return boList;
	}
}
```

```
package ma.formations.domaine;

import java.util.ArrayList;
import java.util.List;

import ma.formations.service.User;

public class UserConverter {
	public static UserVo toVo(User bo) {
		if (bo == null)
			return null;
		UserVo vo = new UserVo();
		vo.setId(bo.getId());
		vo.setUsername(bo.getUsername());
		vo.setPassword(vo.getPassword());
		vo.setRoles(RoleConverter.toVoList(bo.getRoles()));
		return vo;
	}

	public static User toBo(UserVo vo) {
		if (vo == null)
			return null;
		User bo = new User();
		if (vo.getId() != null)
			bo.setId(vo.getId());
		bo.setUsername(vo.getUsername());
		bo.setPassword(vo.getPassword());
		bo.setRoles(RoleConverter.toBoList(vo.getRoles()));
		return bo;
	}

	public static List<UserVo> toVoList(List<User> boList) {
		if (boList == null || boList.isEmpty())
			return null;
```

```
                List<UserVo> voList = new ArrayList<>();
                for (User user : boList) {
                        voList.add(toVo(user));
                }
                return voList;
        }

        public static List<User> toBoList(List<UserVo> voList) {
                if (voList == null || voList.isEmpty())
                        return null;
                List<User> boList = new ArrayList<>();
                for (UserVo userVo : voList) {
                        boList.add(toBo(userVo));
                }
                return boList;
        }
}
```

```
package ma.formations.domaine;

import java.util.ArrayList;
import java.util.List;

import ma.formations.service.Emp;

public class EmpConverter {
        public static EmpVo toVo(Emp bo) {
                if (bo == null || bo.getId() == null)
                        return null;
                EmpVo vo = new EmpVo();
                vo.setId(bo.getId());
                vo.setName(bo.getName());
                vo.setSalary(bo.getSalary());
                vo.setFonction(bo.getFonction());
                return vo;
        }

        public static Emp toBo(EmpVo vo) {
                Emp bo = new Emp();
                bo.setId(vo.getId());
                bo.setName(vo.getName());
                bo.setSalary(vo.getSalary());
                bo.setFonction(vo.getFonction());
                return bo;
        }
```

```
	public static List<EmpVo> toListVo(List<Emp> listBo) {
		List<EmpVo> listVo = new ArrayList<>();
		for (Emp emp : listBo) {
			listVo.add(toVo(emp));
		}
		return listVo;
	}
}
```

```
package ma.formations.domaine;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@NoArgsConstructor
public class TokenVo implements Serializable {
	private static final long serialVersionUID = -8983972106948531914L;
	private String username;
	private String jwttoken;
	private List<String> roles=new ArrayList<>();
}
```

6. Les interfaces Repository de la couche DAO

❖ Créer les interfaces RoleRepository, UserRepository et EmpRepository suivantes :

```
package ma.formations.dao;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;

import ma.formations.service.Role;

public interface RoleRepository extends JpaRepository<Role, Integer> {
	List<Role> findByRole(String role);
```

```
        List<Role> findAll();
}
```

```
package ma.formations.dao;

import org.springframework.data.jpa.repository.JpaRepository;

import ma.formations.service.User;

public interface UserRepository extends JpaRepository<User, Long> {
        User findByUsername(String userName);

        boolean existsByUsername(String username);
}
```

```
package ma.formations.dao;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import ma.formations.service.Emp;

public interface EmpRepository extends JpaRepository<Emp, Long> {
        List<Emp> findBySalary(Double salary);

        List<Emp> findByFonction(String designation);

        List<Emp> findBySalaryAndFonction(Double salary, String fonction);

        @Query(" SELECT e from Emp e where e.salary=(select MAX(salary) as salary
FROM Emp)")
        Emp getEmpHavaingMaxSalary();
}
```

7. La couche Service

❖ Créer la classe BusinessException au niveau du package ma.formations.service.exception :

```
package ma.formations.service.exception;


import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class BusinessException extends RuntimeException {
	private static final long serialVersionUID = 1L;
	private String message;
	public BusinessException(String message) {
		this.message = message;
	}
}
```

❖ Créer l'interface IUserService et la classe UserServiceImpl suivantes :

```
package ma.formations.service;


import java.util.List;

import org.springframework.security.core.userdetails.UserDetailsService;

import ma.formations.domaine.RoleVo;
import ma.formations.domaine.UserVo;

public interface IUserService extends UserDetailsService{
	void save(UserVo user);
	void save(RoleVo role);
	List<UserVo> getAllUsers();
	List<RoleVo> getAllRoles();
	RoleVo getRoleByName(String role);
	void cleanDataBase();
	boolean existsByUsername(String username);
	UserVo findByUsername(String username);
}
```

```
package ma.formations.service;
```

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import ma.formations.dao.EmpRepository;
import ma.formations.dao.RoleRepository;
import ma.formations.dao.UserRepository;
import ma.formations.domaine.RoleConverter;
import ma.formations.domaine.RoleVo;
import ma.formations.domaine.UserConverter;
import ma.formations.domaine.UserVo;
import ma.formations.service.exception.BusinessException;
import ma.formations.service.model.Role;
import ma.formations.service.model.User;

@Service
@Transactional
public class UserServiceImpl implements IUserService {
	@Autowired
	private UserRepository userRepository;
	@Autowired
	private RoleRepository roleRepository;
	@Autowired
	private BCryptPasswordEncoder bCryptPasswordEncoder;

	@Autowired
	private EmpRepository empRepository;

	@Override
	public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
		User user = userRepository.findByUsername(username);
		boolean enabled = true;
		boolean accountNonExpired = true;
		boolean credentialsNonExpired = true;
		boolean accountNonLocked = true;
		return new org.springframework.security.core.userdetails.User(user.getUsername(),
user.getPassword(), enabled,
```

```
                                accountNonExpired, credentialsNonExpired, accountNonLocked,
getAuthorities(user.getRoles()));
        }

        private Collection<? extends GrantedAuthority> getAuthorities(List<Role> roles) {
                List<GrantedAuthority> springSecurityAuthorities = new ArrayList<>();
                roles.forEach(r -> springSecurityAuthorities.add(new
SimpleGrantedAuthority(r.getRole())));
                return springSecurityAuthorities;
        }

        @Override
        public void save(UserVo userVo) {
                User user = UserConverter.toBo(userVo);
                user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
                List<Role> rolesPersist = new ArrayList<>();
                for (Role role : user.getRoles()) {
                        Role userRole = roleRepository.findByRole(role.getRole()).get(0);
                        rolesPersist.add(userRole);
                }
                user.setRoles(rolesPersist);
                userRepository.save(user);
        }

        @Override
        public void save(RoleVo roleVo) {
                roleRepository.save(RoleConverter.toBo(roleVo));
        }

        @Override
        public List<UserVo> getAllUsers() {
                return UserConverter.toVoList(userRepository.findAll());
        }

        @Override
        public List<RoleVo> getAllRoles() {
                return RoleConverter.toVoList(roleRepository.findAll());
        }

        @Override
        public RoleVo getRoleByName(String role) {
                return RoleConverter.toVo(roleRepository.findByRole(role).get(0));
        }

        @Override
        public void cleanDataBase() {
                userRepository.deleteAll();
```

```
                roleRepository.deleteAll();
                empRepository.deleteAll();
        }

        @Override
        public boolean existsByUsername(String username) {
                return userRepository.existsByUsername(username);
        }

        @Override
        public UserVo findByUsername(String username) {
                if (username == null || username.trim().equals(""))
                        throw new BusinessException("login is empty !!");

                User bo = userRepository.findByUsername(username);

                if (bo == null)
                        throw new BusinessException("No user with this login");

                UserVo vo = UserConverter.toVo(bo);
                return vo;
        }

}
```

❖ Créer l'interface IEmpService et la classe EmpServiceImpl suivantes :

```
package ma.formations.service;
import java.util.List;

import ma.formations.domaine.EmpVo;
public interface IEmpService {
        List<EmpVo> getEmployees();
        void save(EmpVo emp);
        EmpVo getEmpById(Long id);
        void delete(Long id);
        List<EmpVo> findBySalary(Double salary);
        List<EmpVo> findByFonction(String designation);
        List<EmpVo> findBySalaryAndFonction(Double salary, String fonction);
        EmpVo getEmpHavaingMaxSalary();
        //Pour la pagination
        List<EmpVo> findAll(int pageId, int size);
        //pour le tri
        List<EmpVo> sortBy(String fieldName);
```

```
}
```

```
package ma.formations.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import ma.formations.dao.EmpRepository;
import ma.formations.domaine.EmpConverter;
import ma.formations.domaine.EmpVo;
import ma.formations.service.model.Emp;

@Service
@Transactional
public class EmpServiceImpl implements IEmpService {
	@Autowired
	private EmpRepository empRepository;
	@Override
	public List<EmpVo> getEmployees() {
		List<Emp> list = empRepository.findAll();
		return EmpConverter.toListVo(list);
	}
	@Override
	public void save(EmpVo emp) {
		empRepository.save(EmpConverter.toBo(emp));
	}
	@Override
	public EmpVo getEmpById(Long id) {
		boolean trouve = empRepository.existsById(id);
		if (!trouve)
			return null;
		return EmpConverter.toVo(empRepository.getById(id));
	}
	@Override
	public void delete(Long id) {
```

```
                empRepository.deleteById(id);
        }
        @Override
        public List<EmpVo> findBySalary(Double salaty) {
                List<Emp> list = empRepository.findBySalary(salaty);
                return EmpConverter.toListVo(list);
        }
        @Override
        public List<EmpVo> findByFonction(String fonction) {
                List<Emp> list = empRepository.findByFonction(fonction);
                return EmpConverter.toListVo(list);
        }
        @Override
        public List<EmpVo> findBySalaryAndFonction(Double salary, String fonction) {
                List<Emp> list = empRepository.findBySalaryAndFonction(salary, fonction);
                return EmpConverter.toListVo(list);
        }
        @Override
        public EmpVo getEmpHavaingMaxSalary() {
                return EmpConverter.toVo(empRepository.getEmpHavaingMaxSalary());
        }
        @Override
        public List<EmpVo> findAll(int pageId, int size) {
                Page<Emp> result = empRepository.findAll(PageRequest.of(pageId, size, Direction.ASC,
"name"));
                return EmpConverter.toListVo(result.getContent());
        }
        @Override
        public List<EmpVo> sortBy(String fieldName) {
                return EmpConverter.toListVo(empRepository.findAll(Sort.by(fieldName)));
        }
}
```

8. Les classes JwtUtils, AuthEntryPointJwt et AuthTokenFilter

Créer les classe **JwtUtils, AuthEntryPointJwt et AuthTokenFilter** suivantes au niveau du package ma.formations.jwt :

```
package ma.formations.jwt;

import java.util.Date;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
```

```
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import io.jsonwebtoken.ExpiredJwtException;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.MalformedJwtException;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.SignatureException;
import io.jsonwebtoken.UnsupportedJwtException;

@Component
public class JwtUtils {
	private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);

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

	@Value("${app.jwtExpirationMs}")
	private int jwtExpirationMs;

	public String generateJwtToken(Authentication authentication) {

		UserDetails userPrincipal = (UserDetails) authentication.getPrincipal();

		return Jwts.builder().setSubject((userPrincipal.getUsername()))
				.setIssuedAt(new Date())
				.setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))
				.signWith(SignatureAlgorithm.HS512, jwtSecret)
				.compact();
	}

	public String getUserNameFromJwtToken(String token) {
		return
Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody().getSubject();
	}

	public boolean validateJwtToken(String authToken) {
		try {
			Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
			return true;
		} catch (SignatureException e) {
			logger.error("Invalid JWT signature: {}", e.getMessage());
		} catch (MalformedJwtException e) {
			logger.error("Invalid JWT token: {}", e.getMessage());
		} catch (ExpiredJwtException e) {
			logger.error("JWT token is expired: {}", e.getMessage());
		} catch (UnsupportedJwtException e) {
```

```
                    logger.error("JWT token is unsupported: {}", e.getMessage());
                } catch (IllegalArgumentException e) {
                    logger.error("JWT claims string is empty: {}", e.getMessage());
                }

                return false;
        }
}
```

```
package ma.formations.jwt;

import java.io.IOException;

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

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;

@Component
public class AuthEntryPointJwt implements AuthenticationEntryPoint {

        private static final Logger logger = LoggerFactory.getLogger(AuthEntryPointJwt.class);

        @Override
        public void commence(HttpServletRequest request, HttpServletResponse response,
                        AuthenticationException authException) throws IOException, ServletException {
                logger.error("Unauthorized error: {}", authException.getMessage());
                response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Error: Unauthorized");
        }

}
```

```
package ma.formations.jwt;

import java.io.IOException;

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

```
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.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import ma.formations.service.IUserService;

public class AuthTokenFilter extends OncePerRequestFilter {
	@Autowired
	private JwtUtils jwtUtils;

	@Autowired
	private IUserService userService;

	private static final Logger logger = LoggerFactory.getLogger(AuthTokenFilter.class);

	@Override
	protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain)
			throws ServletException, IOException {
		try {
			String jwt = parseJwt(request);
			if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
				String username = jwtUtils.getUserNameFromJwtToken(jwt);
				UserDetails userDetails = userService.loadUserByUsername(username);
				UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
				authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
				SecurityContextHolder.getContext().setAuthentication(authentication);
			}
		} catch (Exception e) {
			logger.error("Cannot set user authentication: {}", e);
		}

		filterChain.doFilter(request, response);
	}

	private String parseJwt(HttpServletRequest request) {
		String headerAuth = request.getHeader("Authorization");
```

```
                if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
                        return headerAuth.substring(7, headerAuth.length());
                }

                return null;
        }
}
```

9. La classes SecurityConfiguration

Créer les classe **SecurityConfiguration** suivante au niveau du package ma.formations.configuration :

```
package ma.formations.configuration;


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.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
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.web.authentication.UsernamePasswordAuthenticationFilter;

import ma.formations.jwt.AuthEntryPointJwt;
import ma.formations.jwt.AuthTokenFilter;
import ma.formations.service.IUserService;

@Configuration
@EnableWebSecurity
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

        @Autowired
        private IUserService userService;

        @Autowired
        private BCryptPasswordEncoder bCryptPasswordEncoder;

        @Bean
```

```
	public BCryptPasswordEncoder passwordEncoder() {
		BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();
		return bCryptPasswordEncoder;
	}

	@Autowired
	private AuthEntryPointJwt unauthorizedHandler;

	@Bean
	public AuthTokenFilter authenticationJwtTokenFilter() {
		return new AuthTokenFilter();
	}

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


	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws Exception {
		auth.userDetailsService(userService).passwordEncoder(bCryptPasswordEncoder);
	}

	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.cors();
		http.csrf().disable();
		http.exceptionHandling().authenticationEntryPoint(unauthorizedHandler);
		http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
		http.authorizeRequests().antMatchers("/auth/**").permitAll();

	http.authorizeRequests().antMatchers("/employees/**").hasAnyAuthority("ADMIN","CLIENT");

	http.authorizeRequests().antMatchers("/articles/**").hasAnyAuthority("ADMIN","CLIENT");

	http.authorizeRequests().antMatchers("/categories/**").hasAnyAuthority("ADMIN","CLIENT");

	http.authorizeRequests().antMatchers("/formations/**").hasAnyAuthority("ADMIN","CLIENT");
		http.authorizeRequests().antMatchers("/admin/**").hasAuthority("ADMIN");
		http.authorizeRequests().antMatchers("/client/**").hasAuthority("CLIENT");
		http.authorizeRequests().anyRequest().authenticated();
		http.addFilterBefore(authenticationJwtTokenFilter(),
UsernamePasswordAuthenticationFilter.class);
	}
```

```
	@Override
	public void configure(WebSecurity web) {
		web.ignoring().antMatchers("/resources/**", "/static/**", "/css/**", "/js/**",
"/images/**");
	}

}
```

10. Les contrôlleur AuthenticationController et EmpController

Créer les classes **AuthenticationController et EmpController** suivantes au niveau du package ma.formations.controller :

```
package ma.formations.controller;

import java.util.Collection;

import javax.validation.Valid;

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.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.CrossOrigin;
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 ma.formations.domaine.RoleVo;
import ma.formations.domaine.TokenVo;
import ma.formations.domaine.UserVo;
import ma.formations.jwt.JwtUtils;
import ma.formations.service.IUserService;
import ma.formations.service.exception.BusinessException;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
@RequestMapping("/auth")
public class AuthenticationController {
	@Autowired
	AuthenticationManager authenticationManager;
```

```
	@Autowired
	private IUserService userService;

	@Autowired
	private JwtUtils jwtUtils;

	@PostMapping("/signin")

	public ResponseEntity<TokenVo> authenticateUser(@Valid @RequestBody UserVo userVo) {

		try {
			Authentication authentication = authenticationManager
					.authenticate(new
UsernamePasswordAuthenticationToken(userVo.getUsername(), userVo.getPassword()));
			SecurityContextHolder.getContext().setAuthentication(authentication);
			String jwt = jwtUtils.generateJwtToken(authentication);

			TokenVo tokenVo = new TokenVo();
			tokenVo.setJwttoken(jwt);
			tokenVo.setUsername(userVo.getUsername());
			Collection<? extends GrantedAuthority> list = authentication.getAuthorities();
			list.forEach(authorite -> tokenVo.getRoles().add(authorite.getAuthority()));
			return ResponseEntity.ok(tokenVo);
		} catch (Exception e) {
			throw new BusinessException("Login ou mot de passe incorrect");
		}

	}

	@PostMapping("/signup")
	public ResponseEntity<?> registerUser(@Valid @RequestBody UserVo userVo) {
		if (userService.existsByUsername(userVo.getUsername())) {
			return ResponseEntity.badRequest().body("Error: Username is already taken!");
		}
		// par défaut, le client a le rôle CLIENT
		userVo.getRoles().add(new RoleVo("CLIENT"));
		userService.save(userVo);
		return ResponseEntity.ok("User registered successfully!");
	}

}
```

```
package ma.formations.controller;
```

```
import java.util.List;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import ma.formations.domaine.EmpVo;
import ma.formations.service.IEmpService;

@CrossOrigin(origins = "*", maxAge = 3600)
@RestController
public class EmpController {
	/**
	 * @Autowired permet d'injecter le bean de type IProdcutService (objet
	 *		représentant la couche métier). Ici, le Design Pattern qui est
	 *		appliqué est l'IOC (Inversion Of Control).
	 */
	@Autowired
	private IEmpService service;

	/**
	 * Pour chercher tous les emplyés
	 */
	@GetMapping(value = "/employees", produces = { MediaType.APPLICATION_XML_VALUE,
MediaType.APPLICATION_JSON_VALUE })

	public List<EmpVo> getAll() {
		return service.getEmployees();
	}

	/**
	 * Pour chercher un employé par son id
	 */
	@GetMapping(value = "/employees/{id}")
	public ResponseEntity<Object>  getEmpById(@PathVariable(value = "id") Long empVoId) {
```

```
		EmpVo empVoFound = service.getEmpById(empVoId);
		if (empVoFound == null)
			return new ResponseEntity<>("employee doen't exist", HttpStatus.OK);
		return new ResponseEntity<>(empVoFound, HttpStatus.OK);
	}

	/**
	 * Pour créer un nouveau employé
	 */
	@PostMapping(value = "/admin/create")
	public ResponseEntity<Object> createEmp(@Valid @RequestBody EmpVo empVo) {
		service.save(empVo);
		return new ResponseEntity<>("employee is created successfully",
HttpStatus.CREATED);
	}

	/**
	 * Pour modifier un produit par son id
	 */
	@PutMapping(value = "/admin/update/{id}")
	public ResponseEntity<Object> updateEmp(@PathVariable(name = "id") Long empVoId,
@RequestBody EmpVo empVo) {
		EmpVo empVoFound = service.getEmpById(empVoId);
		if (empVoFound == null)
			return new ResponseEntity<>("employee doen't exist", HttpStatus.OK);
		empVo.setId(empVoId);
		service.save(empVo);
		return new ResponseEntity<>("Employee is updated successsfully", HttpStatus.OK);
	}

	/**
	 * Pour supprimer un employé par son id
	 */
	@DeleteMapping(value = "/admin/delete/{id}")
	//@PreAuthorize("hasRole('ADMIN')")
	public ResponseEntity<Object> deleteEmp(@PathVariable(name = "id") Long empVoId) {
		EmpVo empVoFound = service.getEmpById(empVoId);
		if (empVoFound == null)
			return new ResponseEntity<>("employee doen't exist", HttpStatus.OK);
		service.delete(empVoId);
		return new ResponseEntity<>("Employee is deleted successsfully", HttpStatus.OK);
	}

	/**
	 * Pour chercher tous les emplyés
	 */
	@GetMapping(value = "/rest/sort/{fieldName}", produces = {
```

```
MediaType.APPLICATION_XML_VALUE, MediaType.APPLICATION_JSON_VALUE })
	public List<EmpVo> sortBy(@PathVariable String fieldName) {
		return service.sortBy(fieldName);
	}

	/**
	 * Afficher la liste des employés en utilisant la pagination
	 */
	@GetMapping("/rest/pagination/{pageid}/{size}")
	public List<EmpVo> pagination(@PathVariable int pageid, @PathVariable int size, Model m) {
		return service.findAll(pageid, size);
	}
}
```

11. La classe de démarrage MainApplication

Modifier la classe MainApplication comme suit :

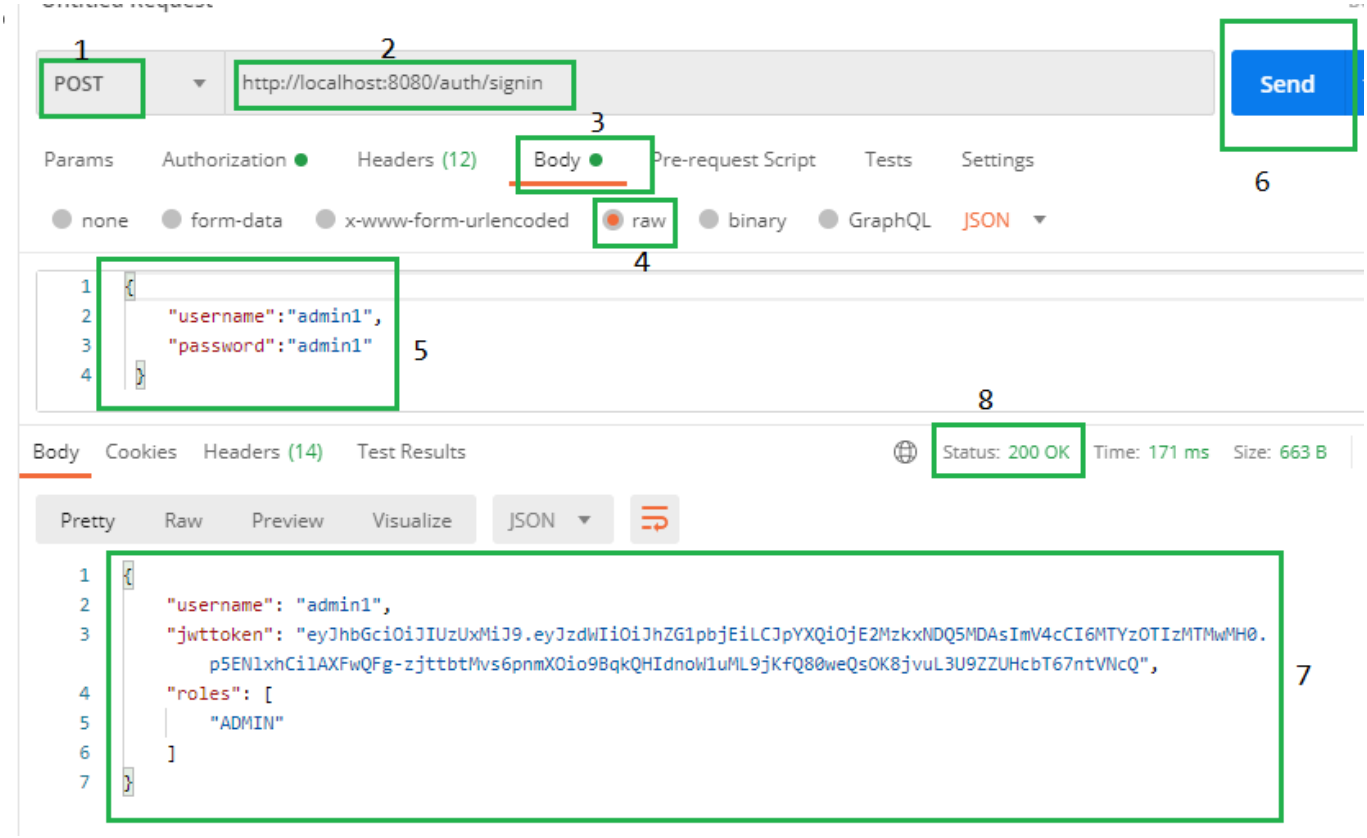
```
package ma.formations;


import java.util.Arrays;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import ma.formations.domaine.EmpVo;
import ma.formations.domaine.RoleVo;
import ma.formations.domaine.UserVo;
import ma.formations.service.IEmpService;
import ma.formations.service.IUserService;

@SpringBootApplication
public class MainApplication implements CommandLineRunner {

	@Autowired
```

```
	private IUserService userService;
	@Autowired
	private IEmpService empService;

	public static void main(String[] args) {
		SpringApplication.run(MainApplication.class, args);
	}

	@Override

	public void run(String... args) throws Exception {
		userService.cleanDataBase();
		userService.save(new RoleVo("ADMIN"));
		userService.save(new RoleVo("CLIENT"));

		RoleVo roleAdmin = userService.getRoleByName("ADMIN");
		RoleVo roleClient = userService.getRoleByName("CLIENT");
		UserVo admin1 = new UserVo("admin1", "admin1", Arrays.asList(roleAdmin));
		UserVo admin2 = new UserVo("admin2", "admin2", Arrays.asList(roleAdmin));
		UserVo client1 = new UserVo("client1", "client1", Arrays.asList(roleClient));
		UserVo client2 = new UserVo("client2", "client2", Arrays.asList(roleClient));
		userService.save(admin1);
		userService.save(client1);
		userService.save(client2);
		userService.save(admin2);

//		// **************
		empService.save(new EmpVo("emp1", 10000d, "Fonction1"));
		empService.save(new EmpVo("emp2", 20000d, "Fonction3"));
		empService.save(new EmpVo("emp3", 30000d, "Fonction4"));
		empService.save(new EmpVo("emp4", 40000d, "Fonction5"));
		empService.save(new EmpVo("emp5", 50000d, "Fonction6"));
	}
}
```

III. Tests

1. Test n°1 : Tester l'authentification admin1/admin1

Lancer POSTMAN et suivre les étapes suivantes :



1- Entrer la méthode : POST
2- Entrer l'URL : http://localhost:8080/auth/signin
3- Cliquer sur Body
4- Cliquer sur raw
5- Entrer le message en format JSON :

```
{
    "username":"admin1",
    "password":"admin1"
}
```

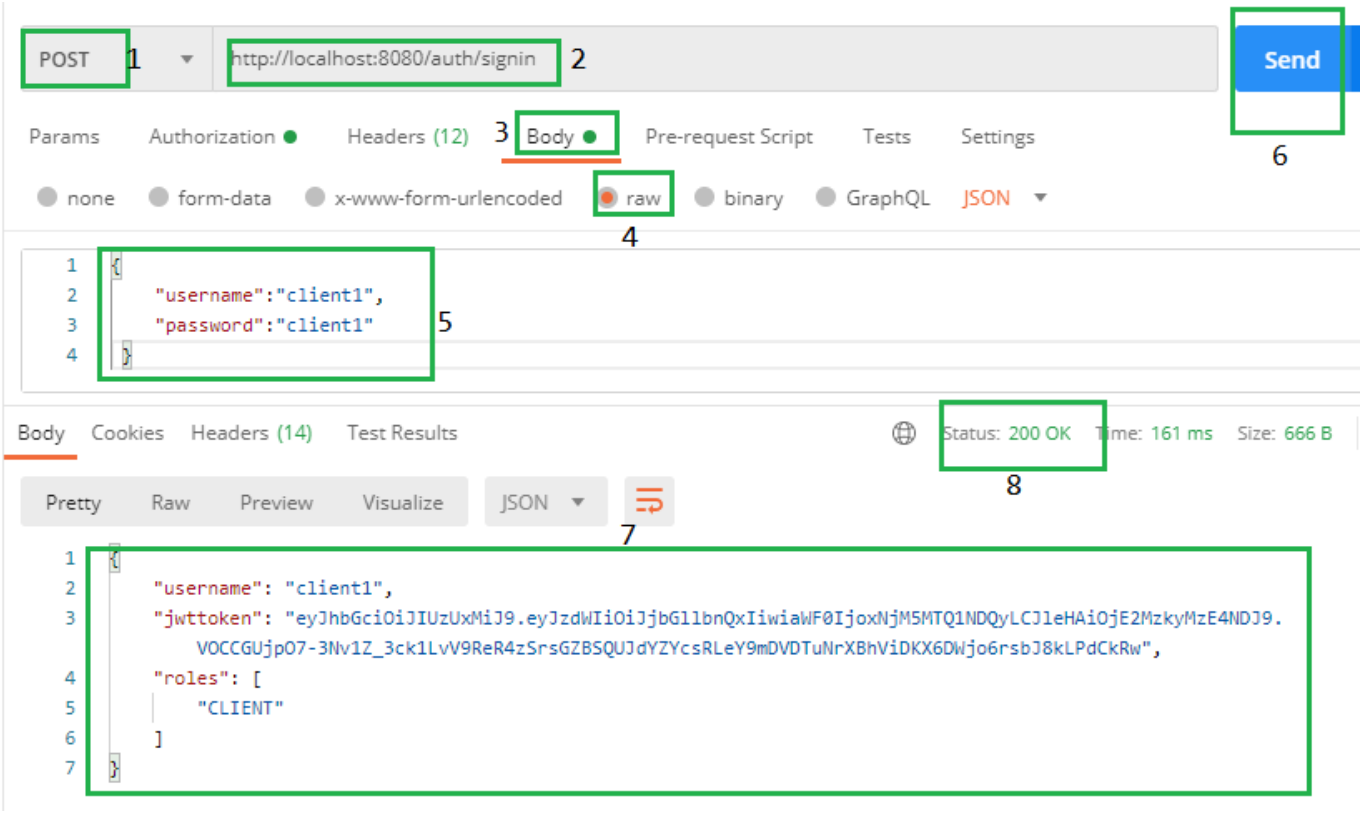
6- Cliquer sur le bouton Send
7- Le résultat devrait être l'objet TokenVo en format JSON.
Observer le token JWT généré :

```
yJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbjEiLCJpYXQiOjE2MzkxNDQ5MDAsImV4cCI6MTYzOTIzMTMwMH0.p
5ENlxhCilAXFwQFg-zjttbtMvs6pnmXOio9BqkQHIdnoW1uML9jKfQ80weQsOK8jvuL3U9ZZUHcbT67ntVNcQ
```

8- Observer le code 200 envoyé par le serveur.

2. Test n°2 : Tester l'authentification avec le compte : client1/client1

Lancer POSTMAN et suivre les étapes suivantes :



1- Entrer la méthode : POST
2- Entrer l'URL : http://localhost:8080/auth/signin
3- Cliquer sur Body
4- Cliquer sur raw
5- Entrer le message en format JSON :

```
{
    "username":"client1",
    "password":"client1"
}
```

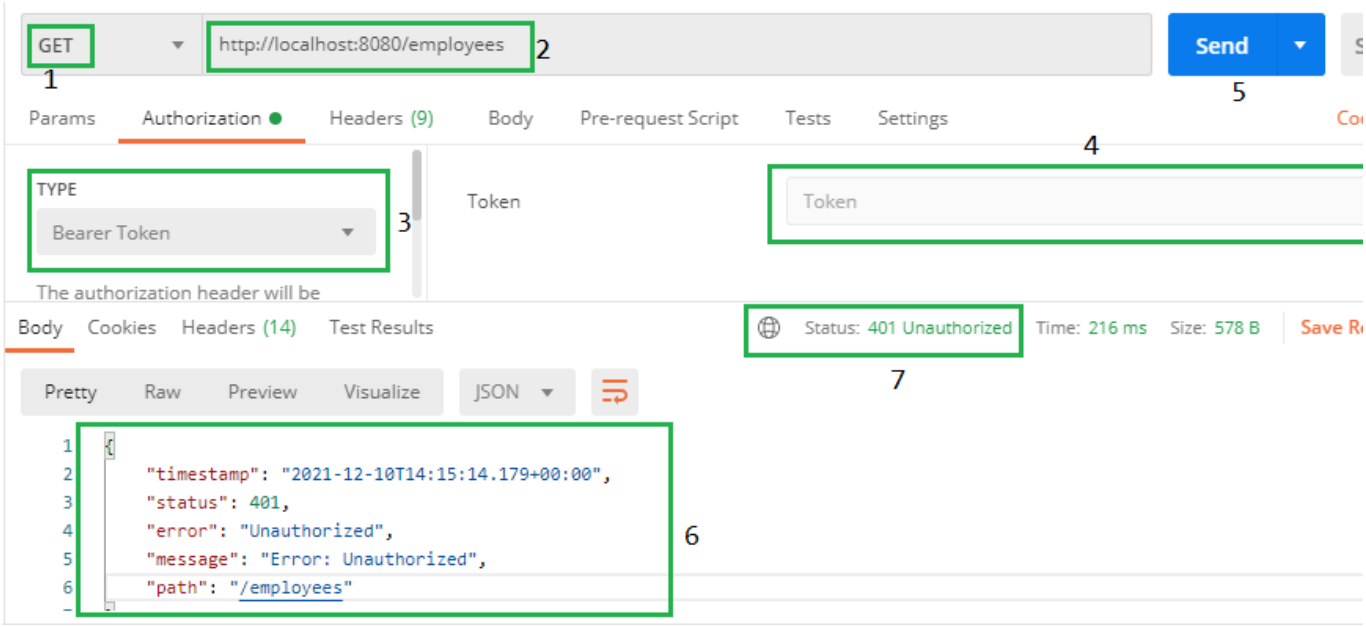
6- Cliquer sur le bouton Send
7- Le résultat devrait être l'objet TokenVo en format JSON.
Observer le token JWT généré :

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJjbGllbnQxIiwiaWF0IjoxNjM5MTQ1NDQyLCJleHAiOjE2MzkyMzE4NDJ9
.VOCCGUjpO7-3Nv1Z_3ck1LvV9ReR4zSrsGZBSQUJdYZYcsRLeY9mDVDTuNrXBhViDKX6DWjo6rsbJ8kLPdCkRw
```

8- Observer le code 200 envoyé par le serveur.

3. Test n°3 : Tester la consultation (/employees) sans token

Lancer POSTMAN et suivre les étapes suivantes :



Entrer la méthode : GET

Entrer l'URL : http://localhost:8080/employees

Choisir le type : « Bearer Token » dans Authorization

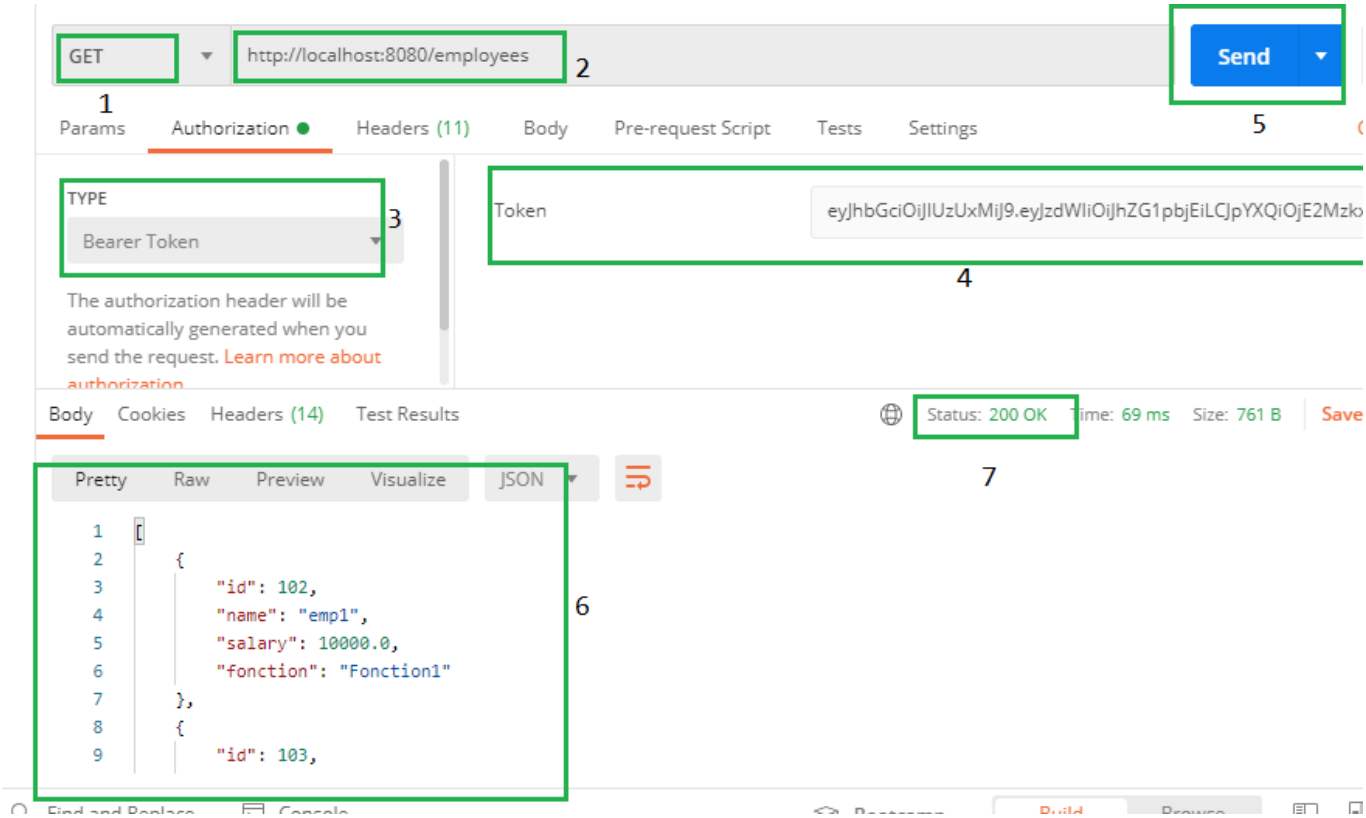
N'entrer pas le token

Cliquer sur le bouton Send

Observer le code erreur envoyé par le serveur.

4. Test n°4 : Tester la consultation (/employees) avec le token de l'utilisateur client1

Lancer POSTMAN et suivre les étapes suivantes :



1- Entrer la méthode : GET
2- Entrer l'URL : http://localhost:8080/employees
3- Choisir le type : « Bearer Token » dans Authorization
4- Entrer le token :

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbjEiLCJpYXQiOjE2MzkxNTA1MTAsImV4cCI6MTYzOTIzNjkxMH0.Hk_4PuADrCkqvFoZahPkrL4lU-xymXWTSfqwtIuZvUFCFKu4Fjwwq4HuhluOBxMEUXSMVul_T79f3Avukwe1dw
```

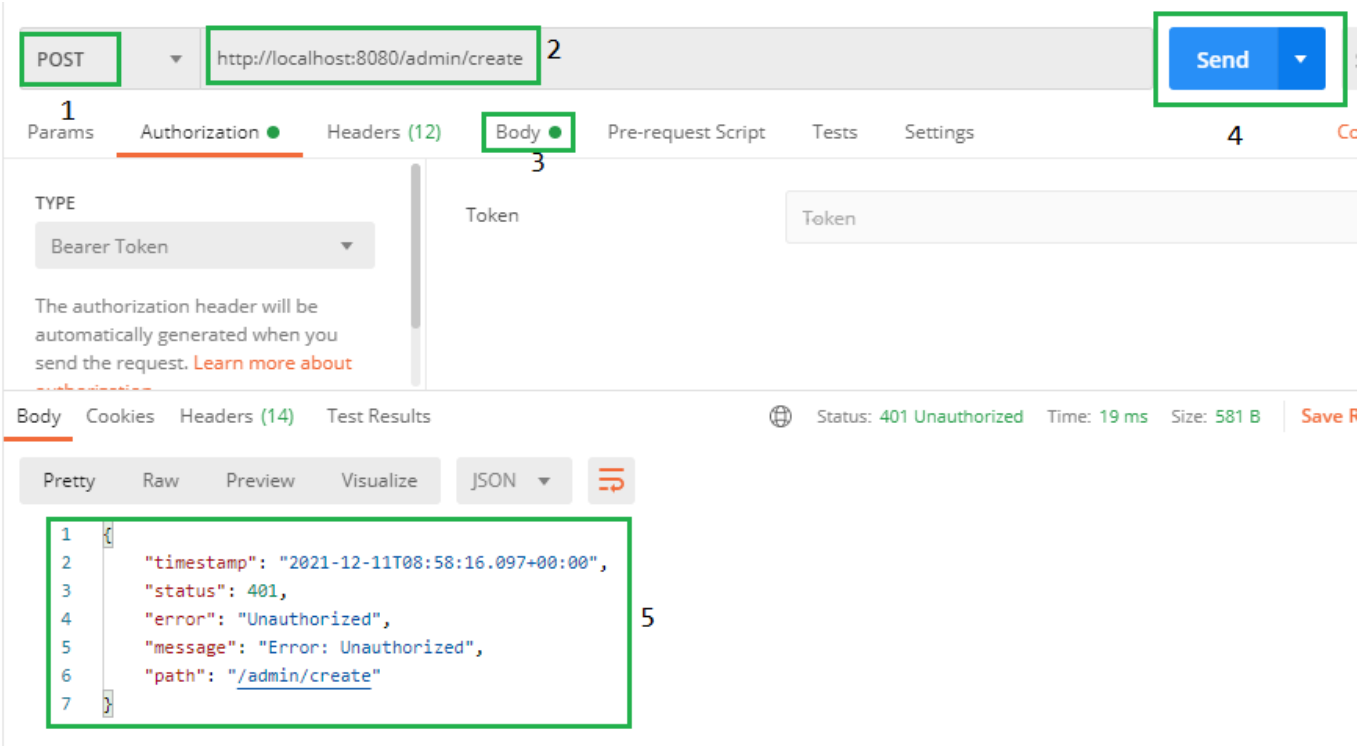
5- Cliquer sur le bouton Send
6- Observer la liste des employées.

5. Test n°5 : Tester la consultation (/employees) avec le token de l'utilisateur admin1

Refaire le même test précédent avec cette fois ci le token de l'utilisateur admin1.

6. Test n°6 : Tester la création (/admin/create) sans token

Lancer POSTMAN et suivre les étapes suivantes :



1- Entrer la méthode : POST
2- Entrer l'URL : http://localhost:8080/admin/create
3- Cliquer sur Body,ensuite sur raw et entrer le message en format JSON :

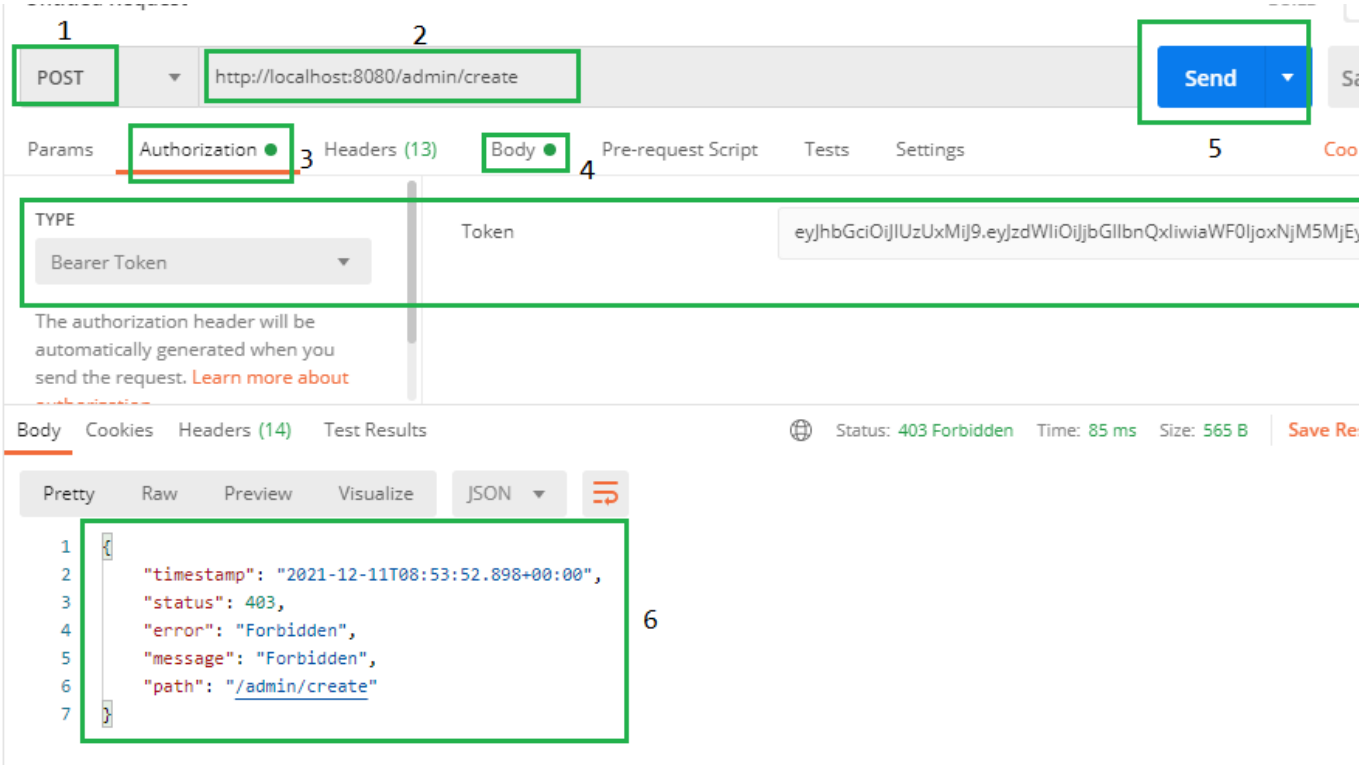
```
{
"name":"Foulane",
"salary":100000,
"fonction":"directeur"
}
```

4- Au niveau de « Bearer Token », laisser le token vide.
5- Cliquer sur le bouton Send
6- Le serveur devrait refiser la requête et envoyer le code erreur 403 :

```
{
    "timestamp": "2021-12-11T08:58:16.097+00:00",
    "status": 401,
    "error": "Unauthorized",
    "message": "Error: Unauthorized",
    "path": "/admin/create"
}
```

7. Test n°7 : Tester la création (/admin/create) avec le token de l'utilisateur client1

Lancer POSTMAN et suivre les étapes suivantes :



7- Entrer la méthode : POST
8- Entrer l'URL : http://localhost:8080/admin/create
9- Cliquer sur Body,ensuite sur raw et entrer le message en format JSON :

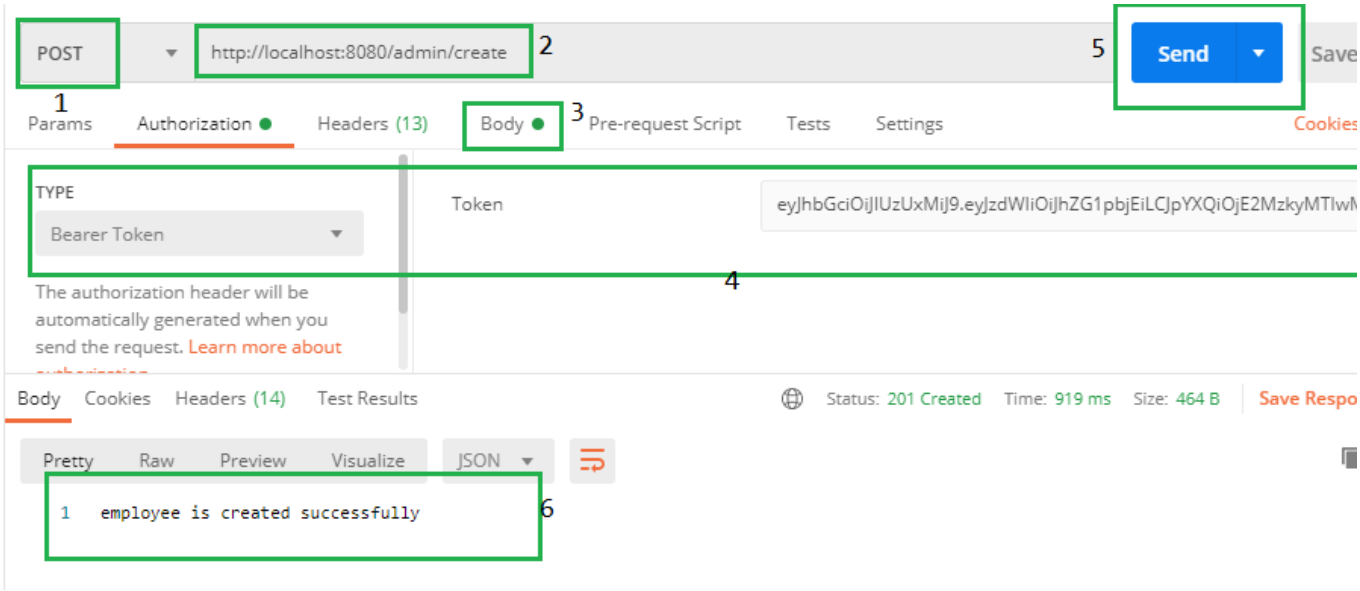
```
{
"name":"Foulane",
"salary":100000,
"fonction":"directeur"
}
```

10- Cliquer sur Authorization, choisir le type « Bearer Token » et entrer le token de l'utilisateur client1
11- Cliquer sur le bouton Send
12- Le serveurdevrait refiser la requête et envoyer le code erreur 403 :

```
{
    "timestamp": "2021-12-11T08:53:52.898+00:00",
    "status": 403,
    "error": "Forbidden",
    "message": "Forbidden",
    "path": "/admin/create"
}
```

8. Test n°8 : Tester la création (/admin/create) avec le token de l'utilisateur admin1

Lancer POSTMAN et suivre les étapes suivantes :



13- Entrer la méthode : POST

14- Entrer l'URL : http://localhost:8080/admin/create

15- Cliquer sur Body,ensuite sur raw et entrer le message en format JSON :

```
{
"name":"Foulane",
"salary":100000,
"fonction":"directeur"
}
```

16- Cliquer sur Authorization, choisir le type « Bearer Token » et entrer le token de l'utilisateur admin1

17- Cliquer sur le bouton Send

18- Le résultat devrait être :