

# **SOSIE 2 SERVICE WEB REST**

SENE MOUHAMED FADEL  
OUADJA NAPO

**Nous avons créé un projet Spring boot starter avec comme dépendances:**

- **Web**
- **JPA**
- **Devtools**
- **Mysql**

**Fichier principal de l'application:** ArloRestApplication.java

```
1 package com.arlo.rest;
2
3 import org.springframework.boot.SpringApplication;
4
5
6 @SpringBootApplication
7 public class ArloRestApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(ArloRestApplication.class, args);
11     }
12 }
13
```

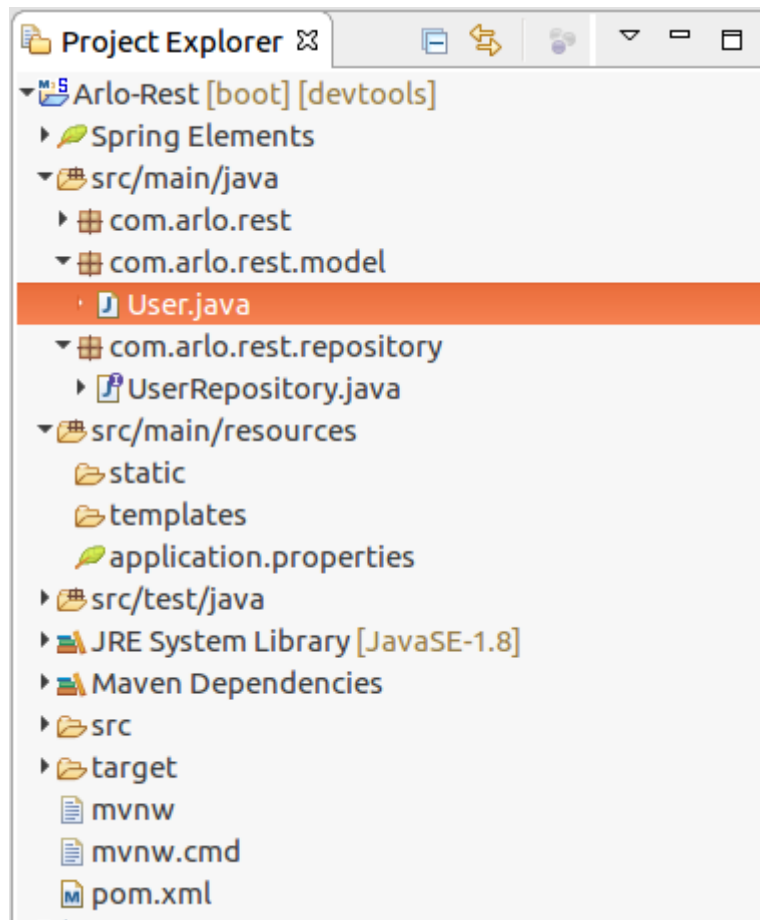
**Dans application.properties nous avons conservé la même config que celle de la version précédente et donc la même base de données:**

```
1 # =====
2 # = DATA SOURCE
3 # =====
4 spring.jpa.hibernate.ddl-auto=create
5 spring.datasource.url=jdbc:mysql://localhost:3306/architecture
6 spring.datasource.username=admin
7 spring.datasource.password=admin
8 server.port=8083
9
```

## Création du modèle:

**nouveau package:** com.arlo.rest.model

**dans lequel on crée une nouvelle classe:** User.java



## Le modèle contient les champs suivants:

- id
- username
- firstName
- lastName
- statut
- email
- password

**Nous avons ajouter le champs id pour que l'on puisse l'auto-incrémenter lors de la création d'un utilisateur avec les web services REST.**

User.java

```
1 package com.arlo.rest.model;
2
3+ import javax.persistence.Entity;
9
10 |
11
12 @Entity
13 @Table(name = "users2")
14 public class User {
15
16
17- @Id
18 @GeneratedValue(strategy = GenerationType.AUTO)
19 private Long id;
20
21- @NotNull
22 private Long username;
23
24- @NotNull
25 private String firstName;
26
27- @NotNull
28 private String lastName;
29
30- @NotNull
31 private String statut;
32
33- @NotNull
34 private String email;
35
36- @NotNull
37 private String password;
38
39- public Long getId() {
40     return id;
41 }
42
43- public void setId(Long id) {
44     this.id = id;
45 }
46
```

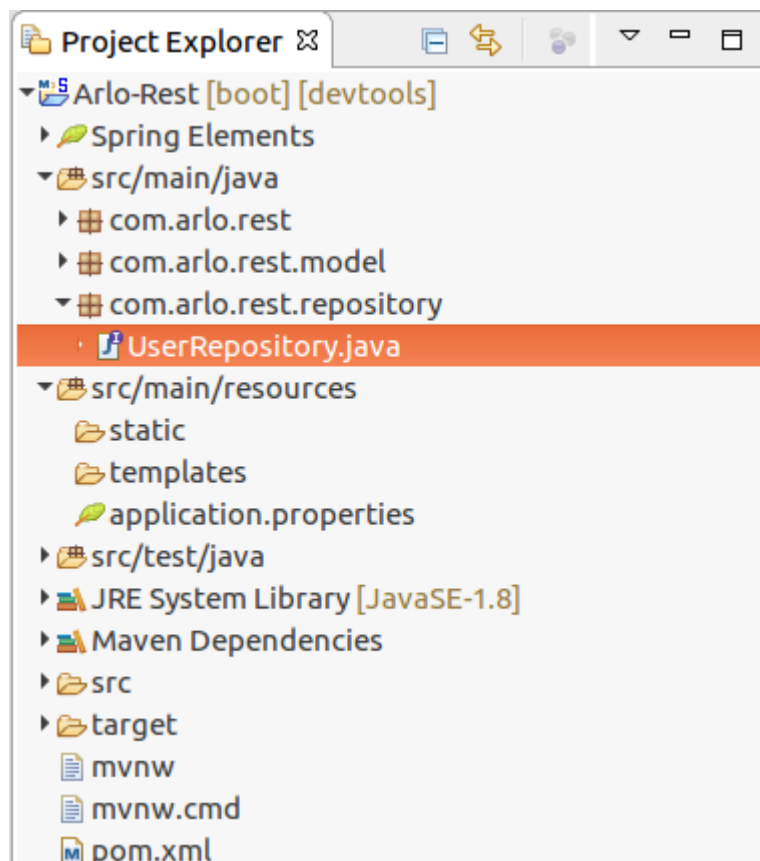
```

47- public Long getUsername() {
48     return username;
49 }
50
51- public void setUsername(Long username) {
52     this.username = username;
53 }
54
55- public String getFirstName() {
56     return firstName;
57 }
58
59- public void setFirstName(String firstName) {
60     this.firstName = firstName;
61 }
62
63- public String getLastName() {
64     return lastName;
65 }
66
67- public void setLastName(String lastName) {
68     this.lastName = lastName;
69 }
70
71- public String getStatut() {
72     return statut;
73 }
74
75- public void setStatut(String statut) {
76     this.statut = statut;
77 }
78
79- public String getEmail() {
80     return email;
81 }
82
83- public void setEmail(String email) {
84     this.email = email;
85 }
86
87- public String getPassword() {
88     return password;
89 }
90
91- public void setPassword(String password) {
92     this.password = password;
93 }
94
95- @Override
96 public String toString() {
97     return "User [id=" + id + ", username=" + username + ", firstName=" + firstName + ", lastName=" + lastName
98         + ", statut=" + statut + ", email=" + email + ", password=" + password + "]";
99 }
100
101
102
103 }

```

**Création d'un référentiel pour accéder aux données de notre base:**  
**Dans** Spring Data JPA **nous avons une interface** JpaRepository **qui inclut des méthodes pour créer, demander, modifier et supprimer des données. Ce sont des méthodes CRUD(Create Read Update Delete).**

**Nous avons donc créer un nouveau package:** com.arlo.rest.repository **et une interface appelée** UserRepository **qui hérite de la classe** JpaRepository:



## UserRepository.java

```
1 package com.arlo.rest.repository;
2
3
4+ import org.springframework.stereotype.Repository;
7
8
9 @Repository
10 public interface UserRepository extends JpaRepository<User, Long> {
11
12 }
13
```

Cette  
classe  
nous

permet d'appeler les méthodes JpaRepository:

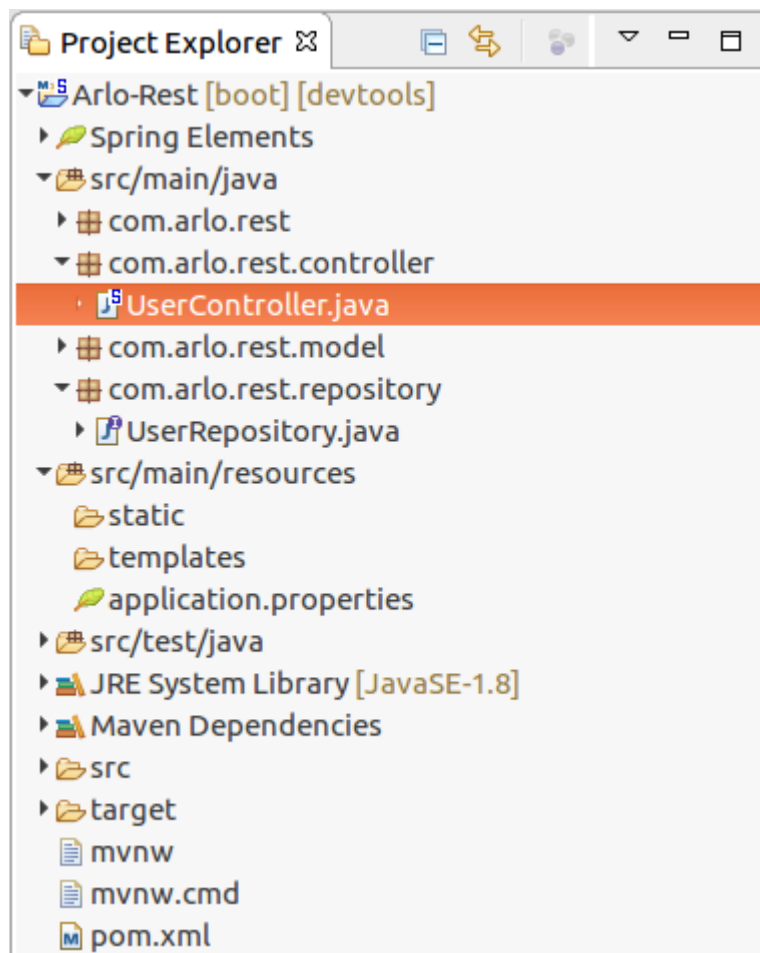
- save()
- findOne()
- findAll()
- count()
- delete()

## Création d'un contrôleur REST:

Ce contrôleur nous permet de créer les API REST en utilisant les méthodes précédentes pour créer, modifier, demander et supprimer des données utilisateurs.

**Nouveau package:** com.arlo.rest.controller

**avec une nouvelle classe:** UserController.java





## UserController.java

```
1 package com.arlo.rest.controller;
2
3 import java.util.List;
4
5 import javax.validation.Valid;
6
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.web.bind.annotation.*;
10 import com.arlo.rest.model.User;
11
12 import com.arlo.rest.repository.UserRepository;
13
14 @RestController
15 @RequestMapping("/api")
16 public class UserController {
17
18     @Autowired
19     UserRepository userRepository;
20
21
22     // Récupère les données utilisateur
23     @GetMapping("/users")
24     public List<User> getAllUsers() {
25         return userRepository.findAll();
26     }
27
28     // Création d'un utilisateur
29     @PostMapping("/users")
30     public User createUser(@Valid @RequestBody User user) {
31         return userRepository.save(user);
32     }
33
34     // Données d'un seul utilisateur
35     @GetMapping("/users/{id}")
36     public ResponseEntity<User> getUserById(@PathVariable(value = "id") Long userId) {
37         User user = userRepository.findOne(userId);
38         if (user == null) {
39             return ResponseEntity.notFound().build();
40         }
41         return ResponseEntity.ok().body(user);
42     }
43
44     // Modification des données d'un utilisateur
45     @PutMapping("/users/{id}")
46     public ResponseEntity<User> updateUser(@PathVariable(value = "id") Long userId,
47         @Valid @RequestBody User userDetails) {
48         User user = userRepository.findOne(userId);
49         if (user == null) {
50             return ResponseEntity.notFound().build();
51         }
52         user.setUsername(userDetails.getUsername());
53         user.setFirstName(userDetails.getFirstName());
54         user.setLastName(userDetails.getLastName());
55         user.setStatut(userDetails.getStatut());
56         user.setEmail(userDetails.getEmail());
57         user.setPassword(userDetails.getPassword());
58         User updatedUser = userRepository.save(user);
59         return ResponseEntity.ok(updatedUser);
60     }
61
62     // Suppression d'un utilisateur
63     @DeleteMapping("/users/{id}")
64     public ResponseEntity<User> deleteUser(@PathVariable(value = "id") Long userId) {
65         User user = userRepository.findOne(userId);
66         if (user == null) {
67             return ResponseEntity.notFound().build();
68         }
69         userRepository.delete(user);
70         return ResponseEntity.ok().build();
71     }
72 }
73
74
75
76
77
78
79
```

## Nous avons créer les API REST en utilisant certaines annotations Spring:

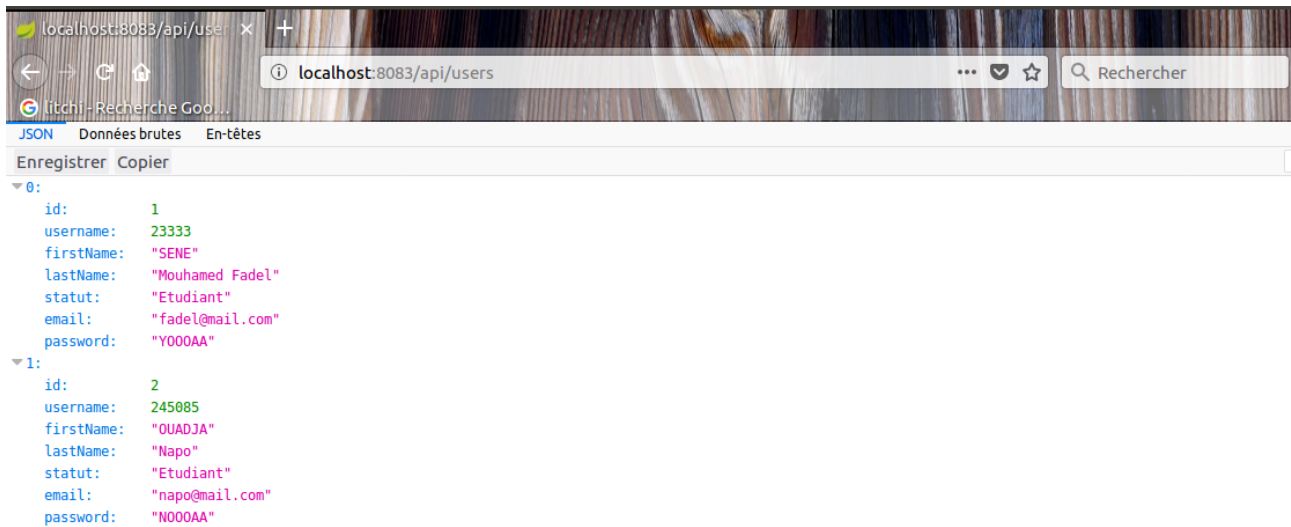
- **@RestController**

L'annotation **@RestController** est une combinaison des annotations **@Controller** et **@ResponseBody** de Spring. L'annotation **@Controller** est utilisée pour définir un contrôleur et l'annotation **@ResponseBody** est utilisée pour indiquer que la valeur de retour d'une méthode doit être utilisée comme corps de réponse à la requête.

- **@RequestMapping("/ api")** déclare que l'URL de tous les API du contrôleur vont commencer par un `/api`.
- **@GetMapping ("/users")** équivaut à un **@RequestMapping** ayant pour valeur `/users` et comme méthode de requête par défaut **POST**.
- **@RequestBody** lie le corps de la requête avec un paramètre de méthode.
- **@Valid** s'assure que le corps de la requête est valide.
- **@PathVariable** lie une variable de chemin avec un paramètre de méthode.

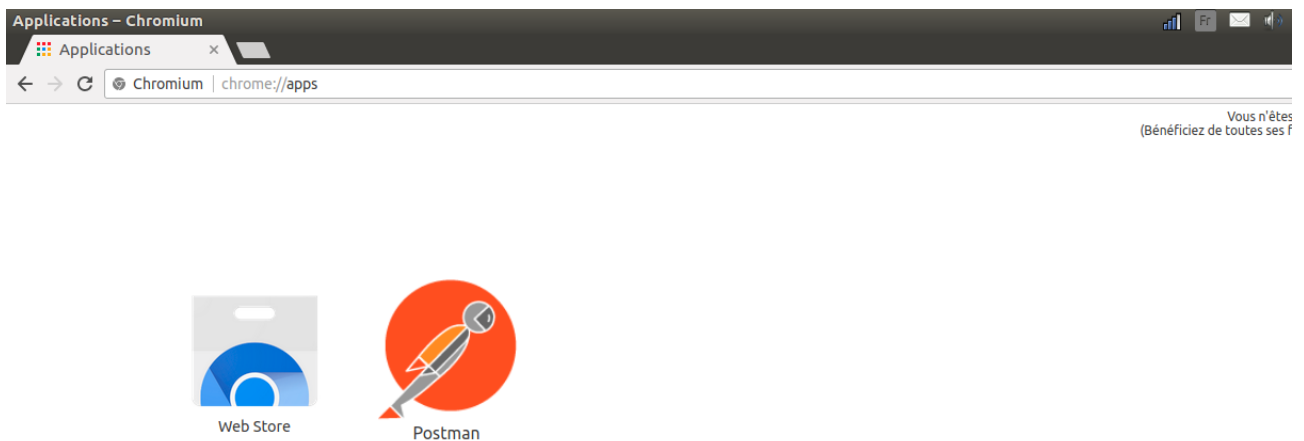
La classe **ResponseEntity** nous donne plus de flexibilité tout en renvoyant une réponse de l'API.

L'API utilisant du GET est accessible sur un navigateur web standard mais pour le reste nous avons utiliser un client REST avancé **POSTMAN** qui est une appli chrome.



Ici on récupère les données utilisateurs.

## POSTMAN

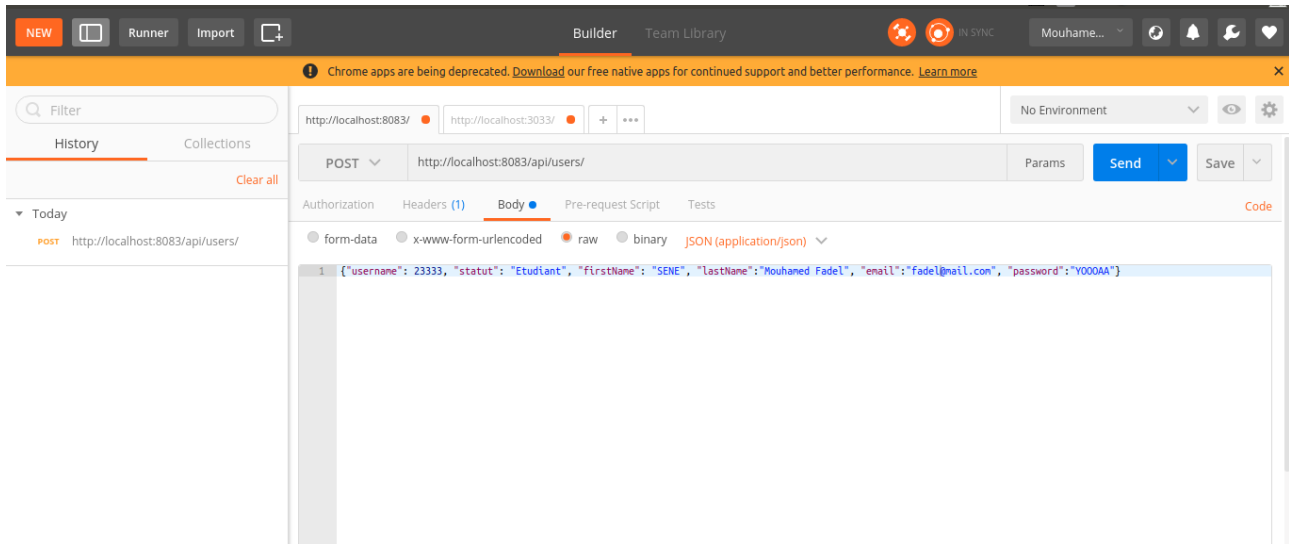


Création d'utilisateur:

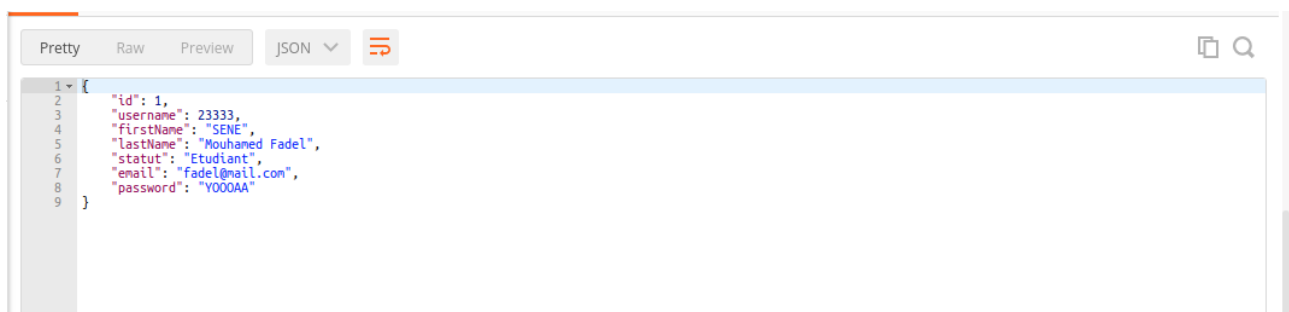
Méthode: POST

URL: <http://localhost:8083/api/users/>

puis on envoie la requête en cliquant sur send



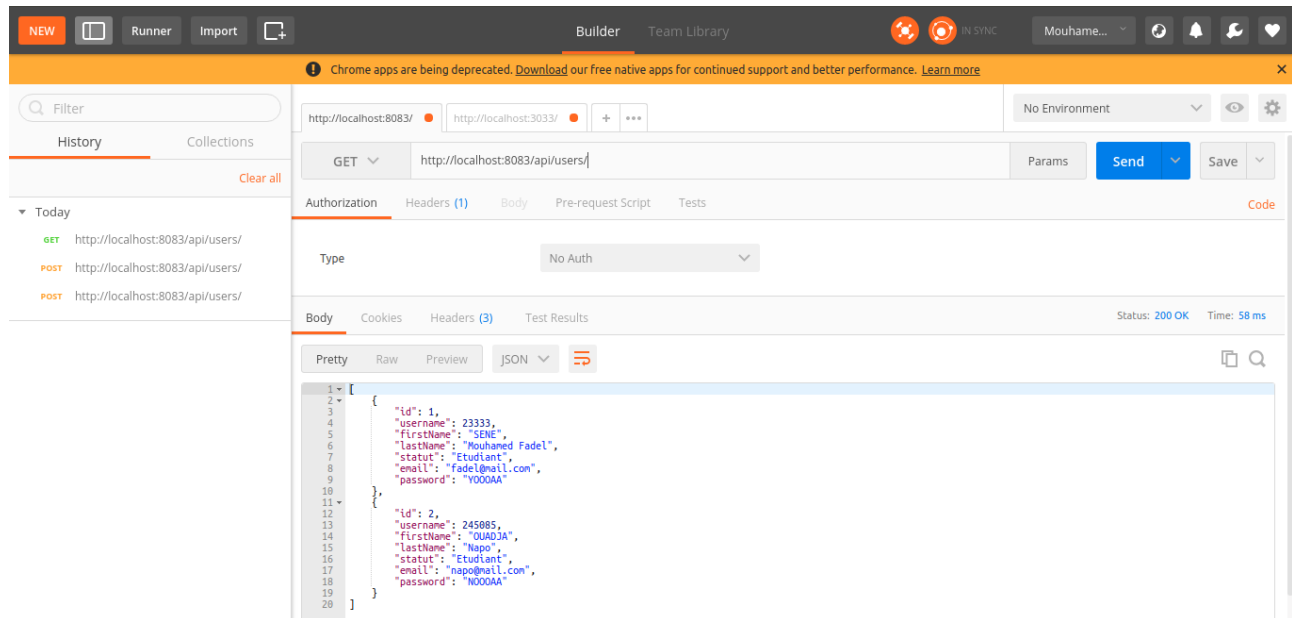
Réponse: Le format de réponse est ici du JSON



Récupération des données utilisateurs:

Méthode: GET

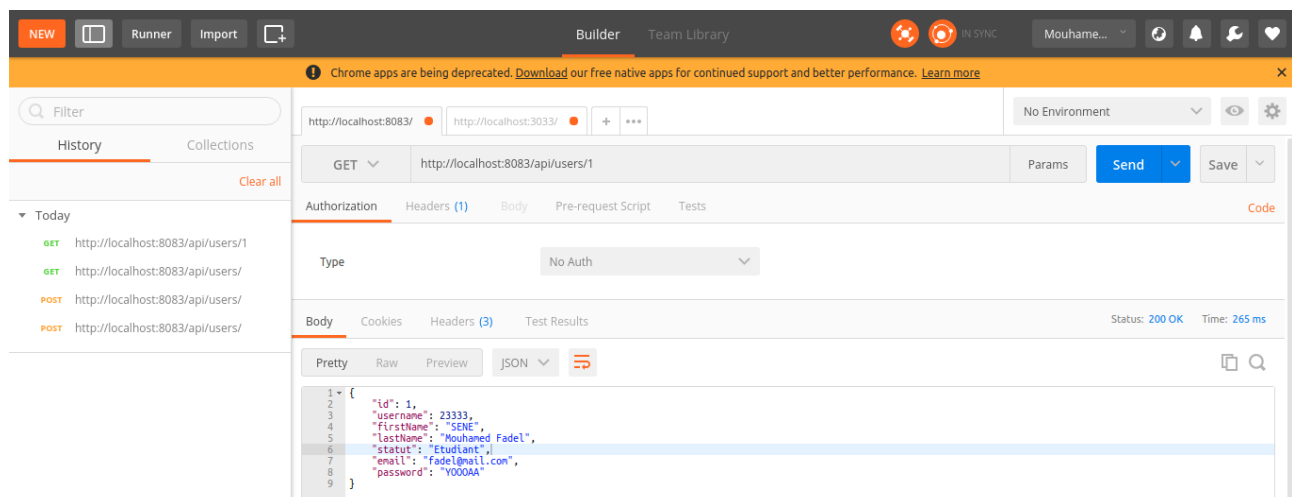
URL: <http://localhost:8083/api/users/>



Récupération des données d'un seul utilisateur:

Méthode: GET

URL: <http://localhost:8083/api/users/{id de l'utilisateur}>

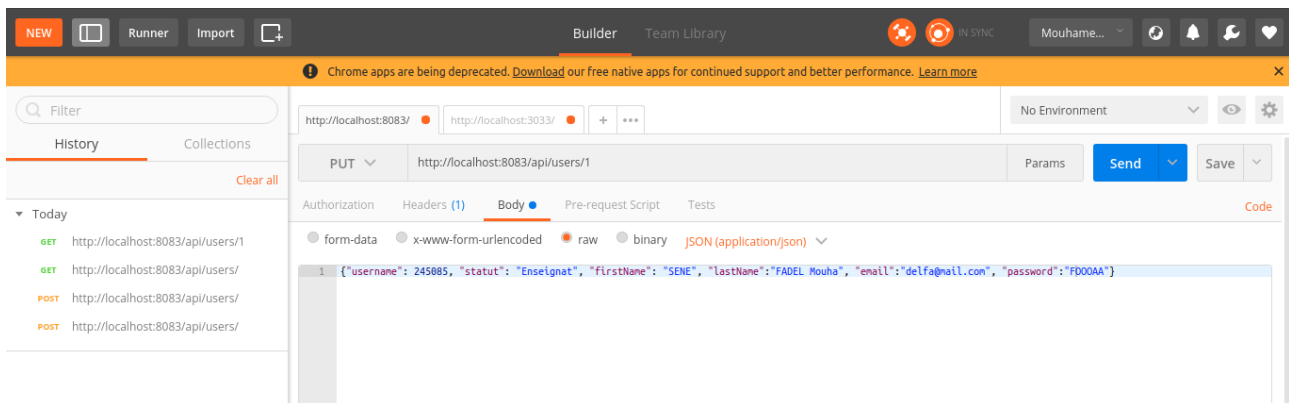


Ici on récupère les données de l'utilisateur 1.

Modification des données d'un seul utilisateur:

Méthode: PUT

URL: <http://localhost:8083/api/users/{id de l'utilisateur}>

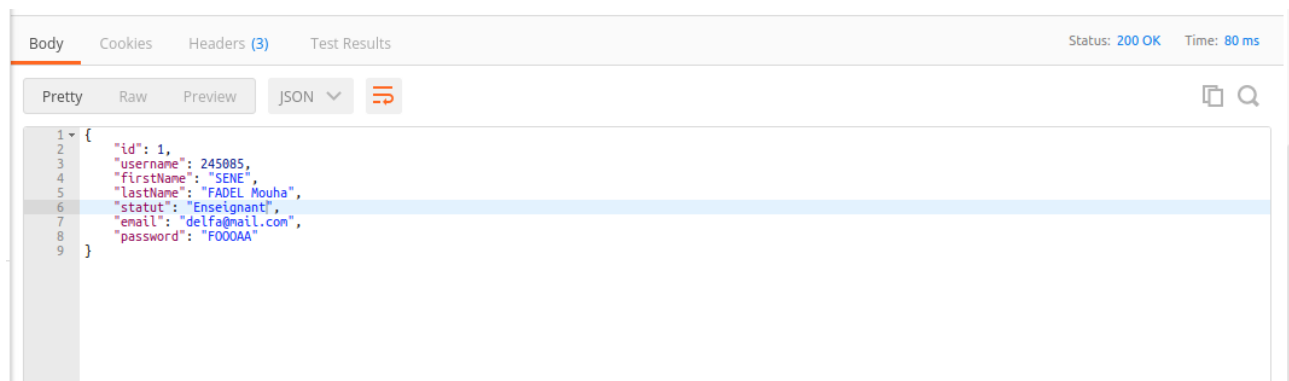


On a modifié les données de l'utilisateur 1.

Réponse:

Status: 200 OK

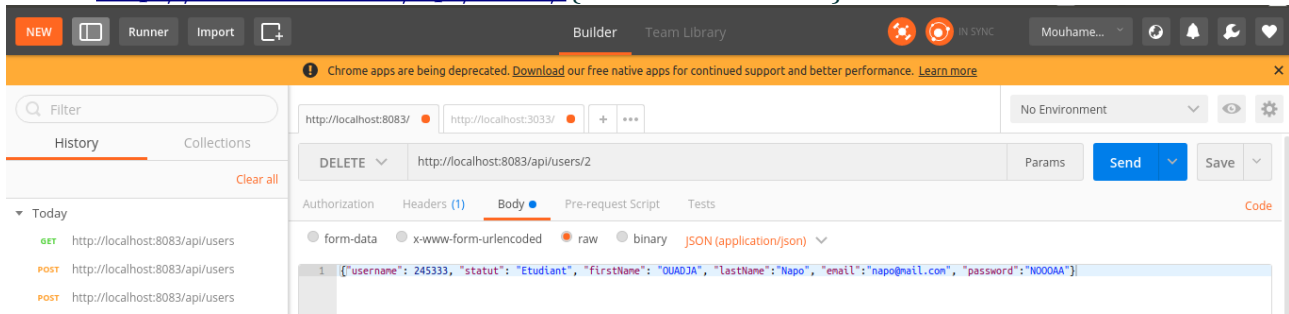
Time: 80ms



Suppression des données d'un seul utilisateur:

Méthode: DELETE

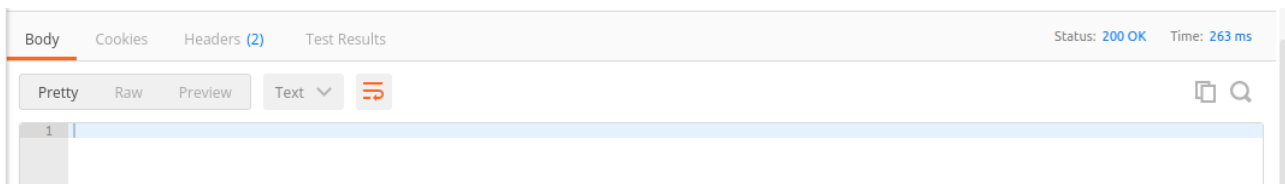
URL: <http://localhost:8083/api/users/{id de l'utilisateur}>



Réponse:

Status: 200 OK

Time: 263ms



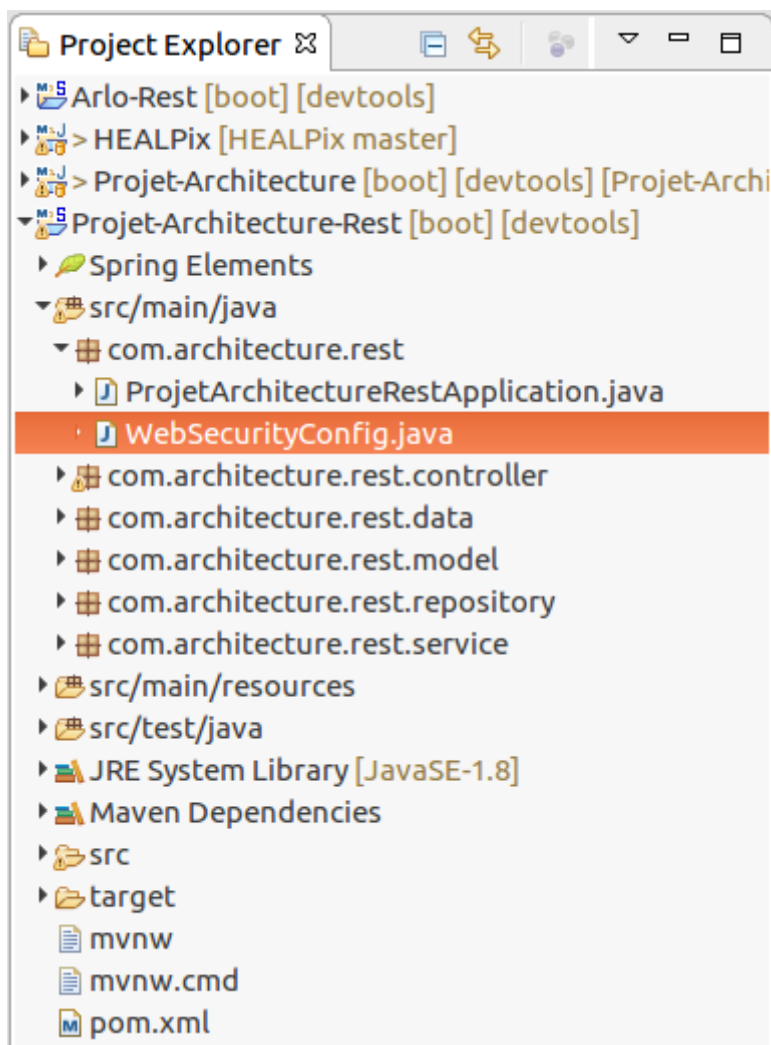
## Version avec IHM

**Dans la version avec IHM nous ajouter la classe**

**UserController.java dans l'ancien projet et modifier le fichier de configuration de la sécurité pour autoriser le path `/api/users` et `/api/users/id` de tourner sur l'appli.**

**Comme dépendance nous avons ajouter tymeleaf aux précédentes pour les templates. Nous également utiliser les ressources static(bootstrap, html, js) du projet de base.**

### Changements dans WebSecurityConfig.java





```

1 package com.architecture.rest;
2
3 import javax.sql.DataSource;
4
5 @Configuration
6 @EnableWebSecurity
7 public class WebSecurityConfig extends WebSecurityConfigurerAdapter{
8
9     @Autowired
10    private DataSource dataSource;
11
12    @Override
13    protected void configure(HttpSecurity http) throws Exception {
14        http
15            .authorizeRequests()
16                .antMatchers("/inscription").permitAll()
17                .antMatchers("/api/users").permitAll()
18                .antMatchers("/api/users/{id}").permitAll()
19                .anyRequest().authenticated()
20                .and()
21            .formLogin()
22                .loginPage("/login")
23                .permitAll()
24                .and()
25            .logout().logoutUrl("/logout").logoutSuccessUrl("/login?logout");
26    }
27
28 }

```

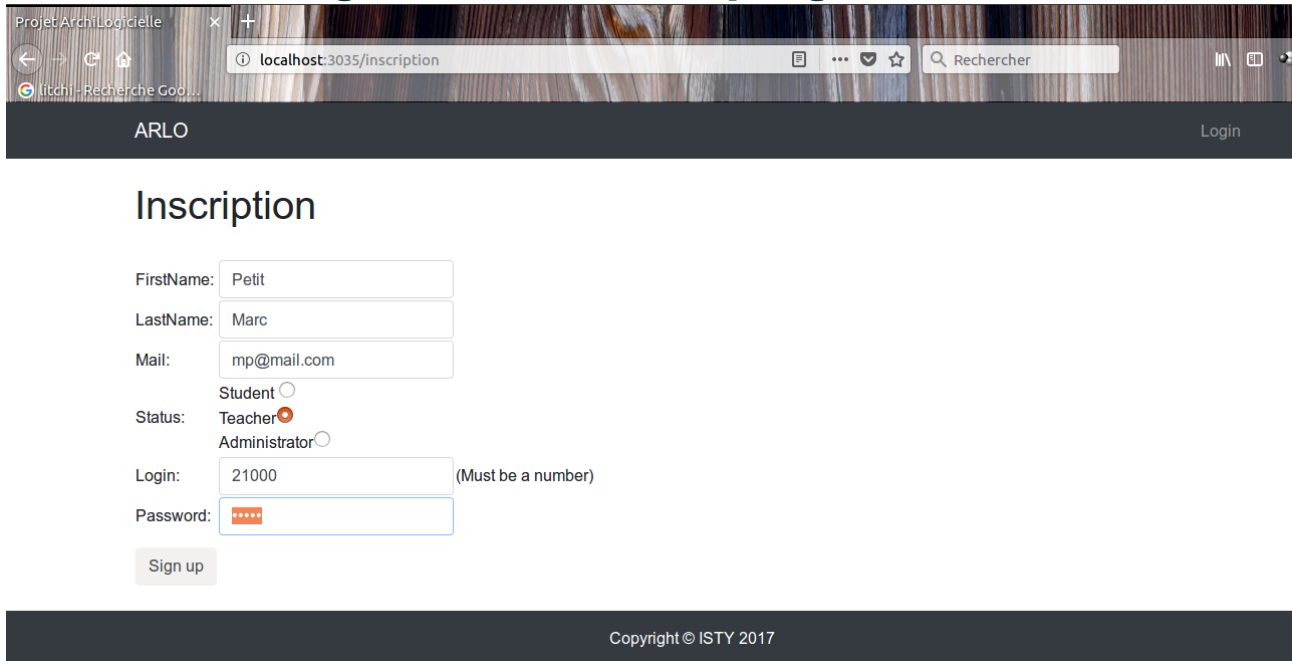
```

    .antMatchers("/api/users").permitAll()
    .antMatchers("/api/users/{id}").permitAll()

```

**L'idée est de créer des utilisateurs sur l'IHM est que derrière, les API interrogent le serveur sur ces données. Ceci marche bien avec la méthode GET.**

**Mais pour créer des utilisateurs avec le client POSTMAN nous avons un message de sécurité de Spring.**



The screenshot shows a web browser window with the address bar displaying 'localhost:3035/inscription'. The page has a dark header with 'ARLO' on the left and 'Login' on the right. The main content area is titled 'Inscription' and contains a registration form. The form fields are as follows:

- FirstName: Petit
- LastName: Marc
- Mail: mp@mail.com
- Status: Student ☐ Teacher ☒ Administrator ☐
- Login: 21000 (Must be a number)
- Password: [masked with red dots]

A 'Sign up' button is located below the password field. At the bottom of the page, a dark footer contains the text 'Copyright © ISTY 2017'.

Projet ArchiLogicielle - Mozilla Firefox

localhost:3035/inscription

ARLO

## Inscription

FirstName:

LastName:

Mail:

Status: ☒ Student ☐ Teacher ☐ Administrator

Login:  (Must be a number)

Password:

Copyright © ISTD 2017

## Récupération des données utilisateurs: Réponse JSON

URL: <http://localhost:3035/api/users/>

localhost:3035/api/users

localhost:3035/api/users

JSON Données brutes En-têtes

Enregistrer Copier

Filtrer le JSON

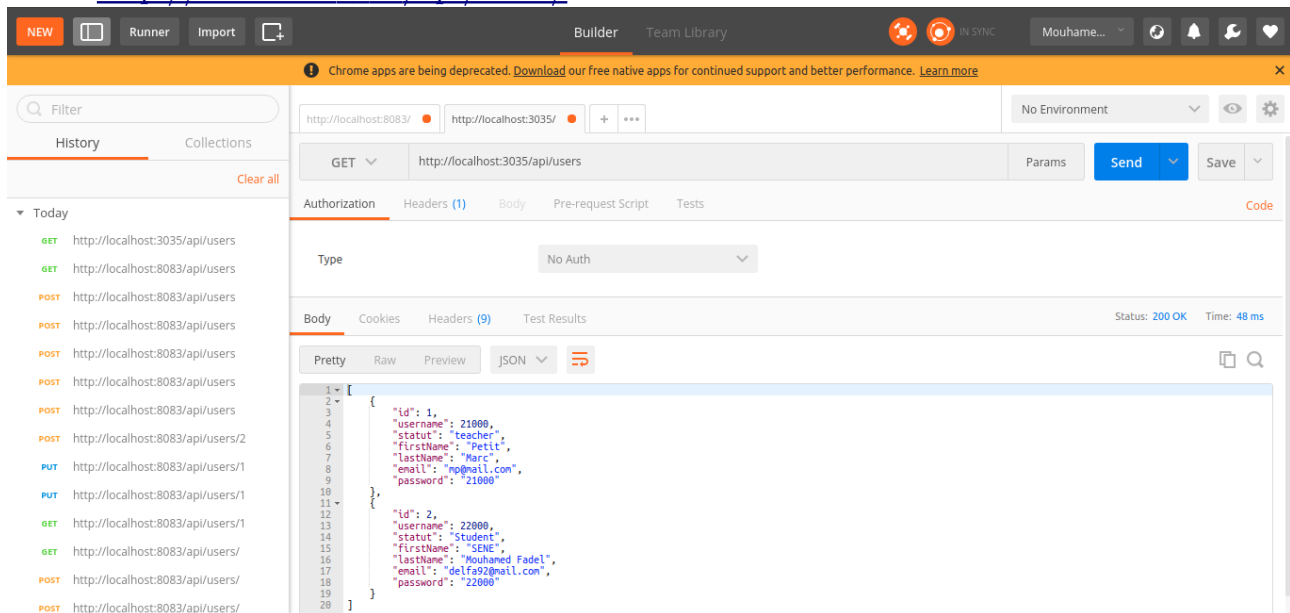
```
0:
  id: 1
  username: 21000
  statut: "teacher"
  firstName: "Petit"
  lastName: "Marc"
  email: "mp@mail.com"
  password: "21000"
1:
  id: 2
  username: 22000
  statut: "Student"
  firstName: "SENE"
  lastName: "Mouhamed Fadel"
  email: "delfa92@mail.com"
  password: "22000"
```

## Sur POSTMAN:

Récupération des données utilisateurs: Réponse JSON

Méthode: GET

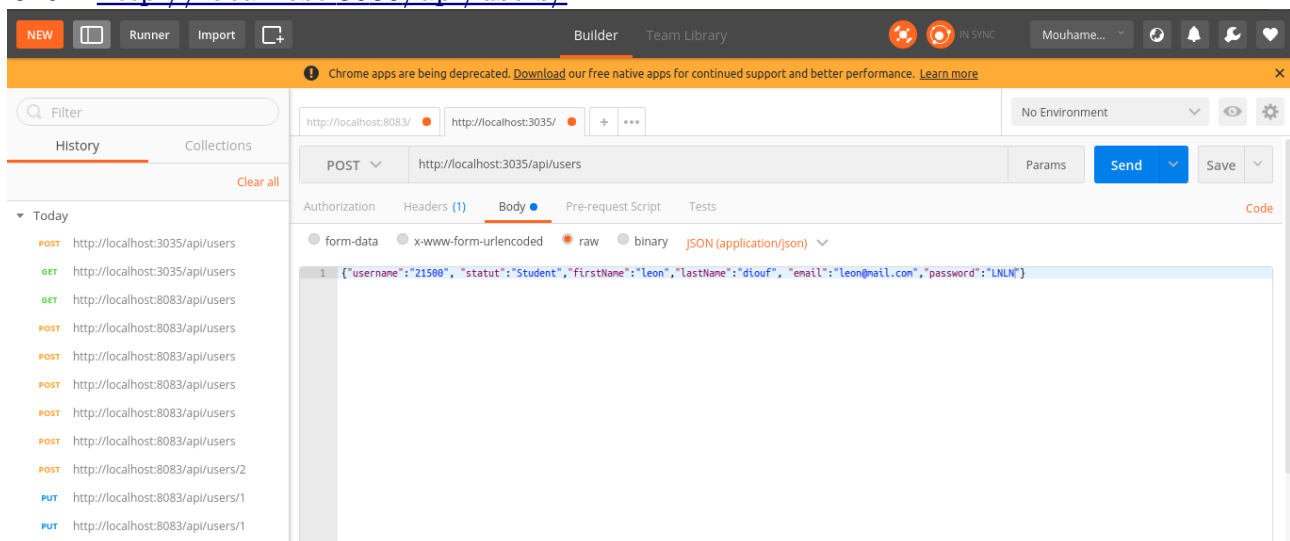
URL: <http://localhost:3035/api/users/>



## Création d'utilisateurs

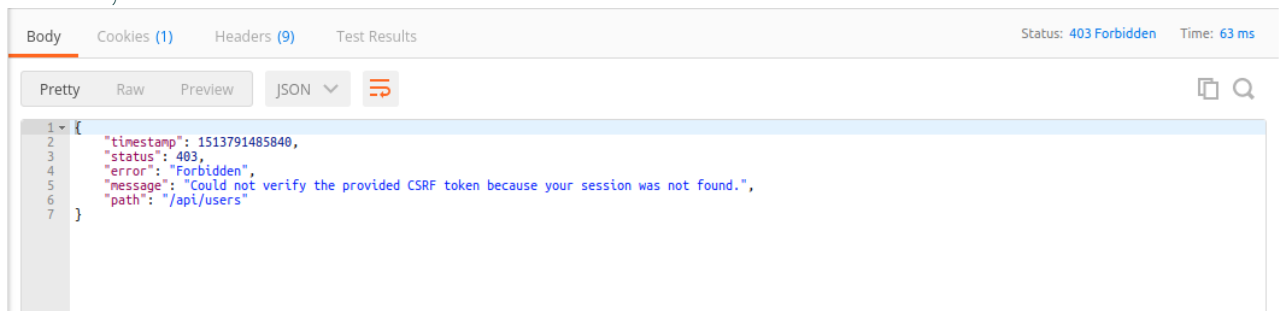
Méthode: POST

URL: <http://localhost:3035/api/users/>



Réponse:

"message": "Could not verify the provided CSRF token because your session was not found.",



### Information par rapport à CSRF:

En sécurité informatique, le Cross-Site Request Forgery, abrégé CSRF (parfois prononcé sea-surfing en anglais) ou XSRF, est un type de vulnérabilité des services d'authentification web. Apparemment le client POSTMAN devrait être authentifié pour pouvoir soumettre des requêtes de type POST à notre appli.