1. Za pomocą Spring Initializera stworzyć aplikację z zależnością **Spring Web.**
2. Stworzyć model dla **Student** zawierającą id, imię, nazwisko, wiek.
3. Stworzyć klasę **StudentService**, stworzyć w niej listę 5 studentów oraz dodać metodę **getAllStudents** zwracająca wszystkich studentów.
4. Stworzyć klasę **StudentController** która działa na uri “**/api/student**” zwracająca wszystkich studentów.
5. Dodać do **StudentService** metody:
   1. addStudent - dodającą studenta,
   2. editStudent - uaktualniając **tylko** wiek i nazwisko
   3. deleteStudent - usuwająca studenta
   4. getStudent - zwracającego studenta o podanym id
6. Dodać do **StudentController** endpointy obsługujące wyżej wymienione metody. Wykorzystaj odpowiednie adnotacje do obsługi zapytań (np. @RequestParam, @RequestBody)
7. Dodaj model **Teacher** zawierający id, imię, nazwisko, unikatową nazwę przedmiotu : [*ALGEBRA, LAW, BIOLOGY*]
8. Rozszerz model **Student** o listę przedmiotów na które chodzi.
9. Dodaj kolejnych 5 studentów do listy
10. Stwórz serwis **TeacherService**, stworzyć w niej listę 3 nauczycieli
11. W **TeacherService** stworzyć metody :
    1. addTeacher - dodaje nauczyciela
    2. deleteTeacher - usuwa nauczyciela
    3. getTeacher - zwraca nauczyciela
    4. getTeacherClass - zwraca wszystkich studentów chodzących na jego zajęcia
    5. deleteStudentFromClassByTeacherId - usuwa podanego studenta z kursu podanego nauczyciela
12. Stworzyć **TeacherController** obsługujący powyższe metody
13. Napisać testy do obu kontrolerów
14. Skonfigurować Swagger’a i zapoznać się z nim
15. Zmienić nazwę i opis swaggera
16. Upakować zwracane obiekty z klas **StudentController** i **TeacherController** w obiekt ResponseEntity - zapoznać się z tym obiektem i wykorzystać tak, żeby zarówno poprawne żądanie było obsłużone, jak i takie które jest niepoprawne ( np. nieistniejące id, niepoprawne dane w zapytaniu ). W obu przypadkach dodać nowy header : “successful” : “true” / “false”
17. Stworzyć **AuthorizationStudentFilter** filtr który będzie obsługiwał tylko **URI** dla StudentControllera i który będzie sprawdzał czy w nagłówkach zapytania znajduje się nagłówek

“**role**” : “**STUDENT\_ROLE**” lub

“**role**” : “**TEACHER\_ROLE**”

Jeśli nie to należy zwrócić wiadomość błędu :

**{"errorMessage":"User unauthorized!"}**

ze statusem **401 Unauthorized**

1. Stworzyć kolejny filtr który będzie się wykonywał **przed** *AuthorizationStudentFilter***,** który będzie wyświetlał w logach serwera różnice czasu rozpoczęciem i zakończeniem zapytania. Ten filtr ma działać przy każdym zapytaniu.
2. Stworzyć filtr **AuthorizationTeacherFilter**, który będzie zabezpieczał **URI** dla kontrolera **TeacherController** bazując na nagłówku :

“**role**” : “**TEACHER\_ROLE**”

W przypadku niepowodzenia zwrócić :

**{"errorMessage":"User unauthorized!"}**

ze statusem **401 Unauthorized**

1. Poprawić testy
2. Zakomentować **AuthorizationStudentFilter** i **AuthorizationTeacherFilter**
3. Stworzyć **AuthorizationService** który posiada metodę **isAuthorized** przyjmującą jako parametry URI oraz nagłówek **role (** jeśli istnieje **)** zapytania.
   1. Dla URI **StudentControllera** metoda ma zwracać true jeśli wartość nagłówka role to :

“**STUDENT\_ROLE**” lub “**TEACHER\_ROLE**”

* 1. Dla URI **StudentControllera** metoda ma zwracać true jeśli wartość nagłówka role to : “**TEACHER\_ROLE**”

1. Stworzyć interceptor **AuthorizeInterceptor,** wstrzyknąć do niego **AuthorizationService** i za pomocą metody **isAuthorized** sprawdzić czy podane zapytanie może się wykonać, jeśli nie, zwrócić :

**{"errorMessage":"User unauthorized!"}**

ze statusem **401 Unauthorized**

1. Jeśli trzeba poprawić testy

Jeśli udało Ci się skończyć przed czasem zachęcam Cię do zapoznania się z materiałami, które uważam za przydatne :

* **Jak stworzyć REST API** : <https://www.youtube.com/watch?v=RNLje5-tFco>
* **Spring Boot Tutorial**: <https://www.youtube.com/watch?v=9SGDpanrc8U>

(zachęcamy do obejrzenia całego, ale szczególnie momentów : [23:19](https://www.youtube.com/watch?v=9SGDpanrc8U&t=1399s) API Layer [26:38](https://www.youtube.com/watch?v=9SGDpanrc8U&t=1598s) Business Layer [29:08](https://www.youtube.com/watch?v=9SGDpanrc8U&t=1748s) Dependency Injection )

1. Za pomocą Spring Initializera stworzyć nową aplikację z zależnością **Spring Web.**
2. Stworzyć dwa kontrolery :
   1. **UserController** ( /api/user ) z metodami:
      1. **getUser** - zwraca **tylko** wiadomość “This is user!”
      2. **updateUser** - zwraca **tylko** wiadomość “User updated!”
   2. **AdminController** ( /api/admin ) z metodami:
      1. **getAdmin** - zwraca **tylko** wiadomość “This is admin!”
      2. **createUser** - zwraca **tylko** wiadomość “User created!”
      3. **deleteUser** - zwraca **tylko** wiadomość “User deleted!”
3. Upewnić się, że działa
4. Dodać spring security do **pom.xml:**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

1. w pliku **application.properties** nadpisać domyślnego użytkownika :

* username: user
* password: pass
* role : user

1. Zabezpieczyć :
   1. **UserController** tak, żeby używać go mógł tylko zalogowany użytkownik z rolami : “user” lub “admin”
   2. **AdminController** tak, żebym mógł go używać tylko zalogowany użytkownik z rolą “admin”

W tym momencie mając domyślnego użytkownika ( application.properties ) z rolą : “user” tylko UserController powinien być osiągalny.

1. Zaktualizować domyślnego użytkownika i zmienić mu rolę na “admin”
2. Przetestować czy oba kontrolery działają dla zalogowanego użytkownika z rolą **admin**
3. Nadpisać UserDetailsService żeby zwracał 2 użytkowników :
   1. username: **admin**; password: **admin**: rola: **admin**
   2. username: **user**; password: **user**: rola: **user**
4. Stworzyć i zastosować PasswordEncoder ( BCryptPasswordEncoder )
5. użyć passwordEncodera w tworzeniu użytkowników z punktu 33
6. Usunąć użytkownika z **application.properties**
7. Przetestować aplikację czy zachowuje się tak samo dla obu użytkowników.
8. Zmienić zabezpiecznia dla endpointów:
   1. **UserController** 
      1. **getUser** - (GET) dostępne dla wszystkich
      2. **updateUser** - (PUT) dostępne tylko dla roli user i admin
   2. **AdminController**
      1. **getAdmin** - (GET) dostępne tylko dla roli user i admin
      2. **createUser** - (POST) dostępne tylko dla roli admin
      3. **deleteUser** - (DELETE) dostępne tylko dla roli admin
9. Zmienić tworzenie użytkowników. Usunąć rolę i dodać uprawnienia :
   1. user : **USER\_READ**, **USER\_EDIT**
   2. admin: **ADMIN**
10. Dodać nowego użytkownika:
    1. username: **spectator**; password: **spectator**: uprawnienia: **USER\_READ**
11. Dostosować security do obsługi sprawdzania uprawnień ( nie roli )
    1. **UserController** 
       1. **getUser** - dostępne dla **USER\_READ i ADMIN**
       2. **updateUser** - dostępne dla **USER\_EDIT i ADMIN**
    2. **AdminController**
       1. **getAdmin** - dostępne dla **ADMIN**
       2. **createUser** - dostępne dla **ADMIN**
       3. **deleteUser** - dostępne dla **ADMIN**
12. Stworzyć osobny projekt i wykonać w nim : <https://spring.io/guides/gs/messaging-stomp-websocket/>
13. Zmodyfikować ten projekt dodając konfigurację **SchdeluerConfig** wraz z adnotacją @EnableScheduling. Następnie za pomocą **SimpleMessageTemplate** wysyłać automatycznie co 1 sekundę wiadomość z jednym losowym imieniem.
14. [**Zadanie dla chętnych - ta wiedza nie będzie później sprawdzana**] Jeżeli skończyłeś zachęcam Cię do zaimplementowania **JWT** w zadaniu z punktów **25) - 41)**.

Jeśli udało Ci się skończyć przed czasem zachęcam Cię do zapoznania się z materiałami, które uważam za przydatne :

# **Kurs Spring Boot #07 - Spring Security i JWT Token:** <https://www.youtube.com/watch?v=and2DR_N6tE>

# **Spring Security | FULL COURSE** : <https://www.youtube.com/watch?v=her_7pa0vrg>