**Spring Web Security**

**Теория**

Теория по Spring Security может быть найдена в файле Spring Security Reference.html

**Практическая часть**

**Security** добавляется к контроллерам при помощи аннотаций, либо при помощи xml-конфигурации.

**За основу будет взята Лабораторная работа №5.**

# Добавление зависимостей SpringSecurity к pom.xml

Добавить к pom.xml следующие зависимости:

…

<artifactId>spring-security-web</artifactId>

<version>4.2.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>4.2.3.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>4.2.3.RELEASE</version>

</dependency>

</dependencies>

<build>

<finalName>REST</finalName>

<pluginManagement>

…

# Конфигурация Security при помощи JavaConfig

Создать класс SecurityWebApplicationInitializer в пакете конфигурации, наследующийся от AbstractSecurityWebApplicationInitializer добавляющий Security-фильтр к приложению:

**public** **class** SecurityWebApplicationInitializer **extends** AbstractSecurityWebApplicationInitializer {

}

Класс в данном случае не переопределяет никаких методов суперкласса и существует ради переопределения абстрактного класса в зоне видимости контекста.

Создать класс WebSecurityConfig:

@EnableGlobalMethodSecurity(prePostEnabled = **true**)

@EnableWebSecurity

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Override

**protected** **void** configure(HttpSecurity http) {

http.httpBasic().and().authorizeRequests().antMatchers("/api/group/\*\*", "/api/student/\*\*").permitAll().and().csrf().disable();

}

}

В данном классе переопределяется метод, отвечающий за политику безопасности. В нашем случае – BasicAuthentication, разрешение всех запросов по указанным адресам, отключение csrf. Подробнее можно почитать в документации.

# Добавление UserDetailService

Для тестирования используется сервис, хранящий данные о тестовых пользователях в памяти. В дальнейшем он будет заменен на сервис, хранящий пользователей в БД.

В класс WebSecurityConfig добавляется бин сервиса:

@EnableGlobalMethodSecurity(prePostEnabled = **true**)

@EnableWebSecurity

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Bean

**public** UserDetailsService userDetailsService(){

InMemoryUserDetailsManager manager = **new** InMemoryUserDetailsManager();

manager.createUser(User.*withUsername*("user").password("password").roles(

"ADMIN").build());

**return** manager;

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.httpBasic().and().authorizeRequests().antMatchers("/api/group/\*\*", "/api/student/\*\*").permitAll().and().csrf().disable();

}

}

# Добавление security аннотаций на контроллеры

Аннотации @PreAuthorise имеют параметр, принимающий следующие значения (выдержка из документации):

|  |  |
| --- | --- |
| hasRole([role]) | Returns true if the current principal has the specified role. |
| hasAnyRole([role1,role2]) | Returns true if the current principal has any of the supplied roles (given as a comma-separated list of strings) |
| principal | Allows direct access to the principal object representing the current user |
| authentication | Allows direct access to the current Authentication object obtained from the SecurityContext |
| permitAll | Always evaluates to true |
| denyAll | Always evaluates to false |
| isAnonymous() | Returns true if the current principal is an anonymous user |
| isRememberMe() | Returns true if the current principal is a remember-me user |
| isAuthenticated() | Returns true if the user is not anonymous |
| isFullyAuthenticated() | Returns true if the user is not an anonymous or a remember-me user |

Так как к каждой сущности доступ может быть предоставлен разным ролям, абстрактный контроллер больше не может объединять общую реализацию контроллеров, и каждый из контроллеров описывается самостоятельно:

**Контроллер групы:**

@RestController

@RequestMapping(value = "api/group", produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** **class** GroupController {

@Autowired

**private** GroupService service;

@GetMapping

**public** ResponseEntity<List<Group>> get() {

List<Group> entities = service.read();

**if** (entities.isEmpty()) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(entities, HttpStatus.***OK***);

}

@GetMapping("/{id}")

**public** ResponseEntity<Group> getById(@PathVariable **long** id) {

Group entity = service.read(id);

**if** (entity == **null**) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(entity, HttpStatus.***OK***);

}

@GetMapping("/name/{name}")

**public** ResponseEntity<Group> getStudentsBySurname(@PathVariable String name) {

Group group = service.readByName(name);

**if** (group == **null**) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(group, HttpStatus.***OK***);

}

@PreAuthorize("hasRole('ADMIN')")

@PutMapping(consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> put(@RequestBody Group entity) {

service.save(entity);

**return** **new** ResponseEntity<>(HttpStatus.***CREATED***);

}

@PreAuthorize("hasRole('ADMIN')")

@PostMapping(consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> post(@RequestBody Group entity) {

service.save(entity);

**return** **new** ResponseEntity<>(HttpStatus.***OK***);

}

@PreAuthorize("hasRole('ADMIN')")

@DeleteMapping("/{id}")

**public** ResponseEntity<String> delete(@PathVariable **long** id) {

service.delete(id);

**return** **new** ResponseEntity<>(HttpStatus.***OK***);

}

}

**Контроллер студента:**

@RestController

@RequestMapping(value = "api/student", produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** **class** StudentController {

@Autowired

**private** StudentService service;

@GetMapping

**public** ResponseEntity<List<Student>> get() {

List<Student> entities = service.read();

**if** (entities.isEmpty()) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(entities, HttpStatus.***OK***);

}

@GetMapping("/{id}")

**public** ResponseEntity<Student> getById(@PathVariable **long** id) {

Student entity = service.read(id);

**if** (entity == **null**) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(entity, HttpStatus.***OK***);

}

@GetMapping("/group/{id}")

**public** ResponseEntity<List<Student>> getStudentsByGroup(@PathVariable **long** id) {

List<Student> students = service.readByGroupId(id);

**if** (students.isEmpty()) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(students, HttpStatus.***OK***);

}

@GetMapping("/surname/{surname}")

**public** ResponseEntity<List<Student>> getStudentsBySurname(@PathVariable String surname) {

List<Student> students = service.readBySurname(surname);

**if** (students.isEmpty()) {

**return** **new** ResponseEntity<>(HttpStatus.***NOT\_FOUND***);

}

**return** **new** ResponseEntity<>(students, HttpStatus.***OK***);

}

@PreAuthorize("hasRole('ADMIN')")

@PutMapping(consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> put(@RequestBody Student entity) {

service.save(entity);

**return** **new** ResponseEntity<>(HttpStatus.***CREATED***);

}

@PreAuthorize("hasRole('ADMIN')")

@PostMapping(consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<String> post(@RequestBody Student entity) {

service.save(entity);

**return** **new** ResponseEntity<>(HttpStatus.***OK***);

}

@PreAuthorize("hasRole('ADMIN')")

@DeleteMapping("/{id}")

**public** ResponseEntity<String> delete(@PathVariable **long** id) {

service.delete(id);

**return** **new** ResponseEntity<>(HttpStatus.***OK***);

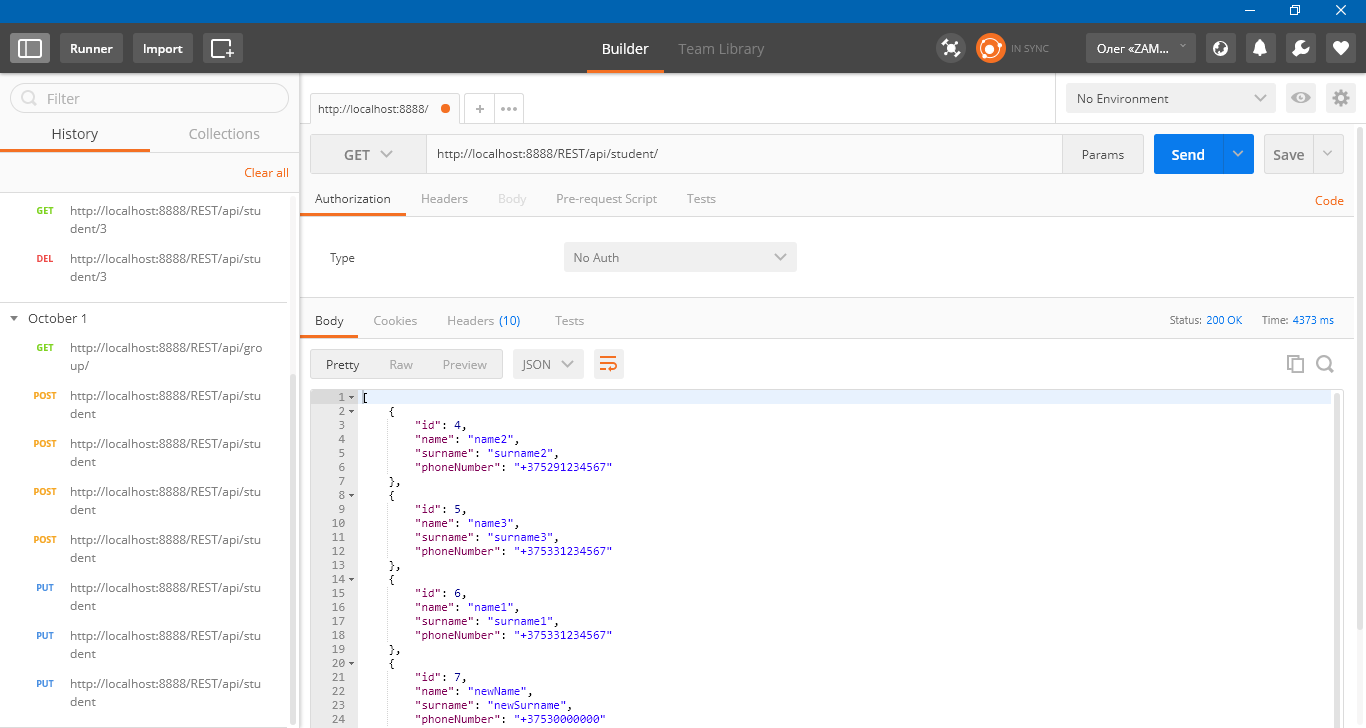
}

}

# Проверка security при помощи Postman

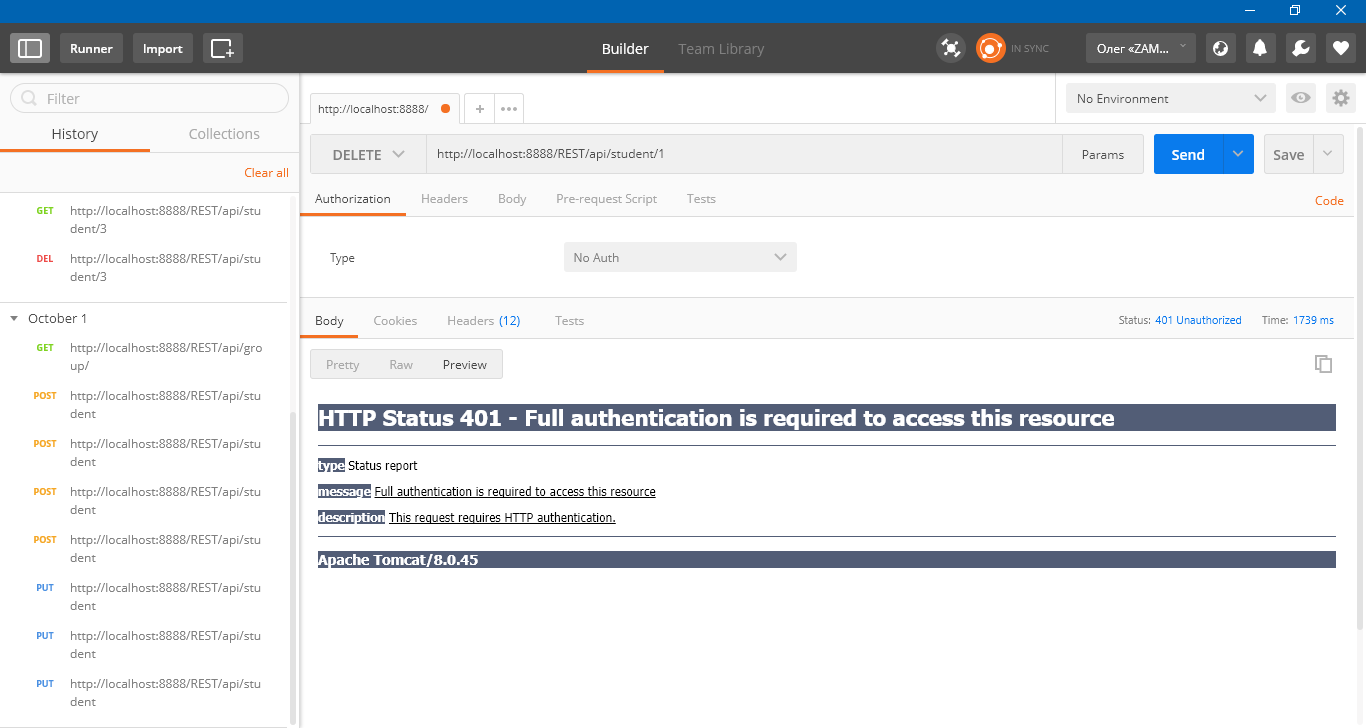
Проверить настройки security при помощи Postman

Запросы, помеченные как permitAll (либо не помеченные аннотацией) допускаются фильтром и не требуют аутентификации:

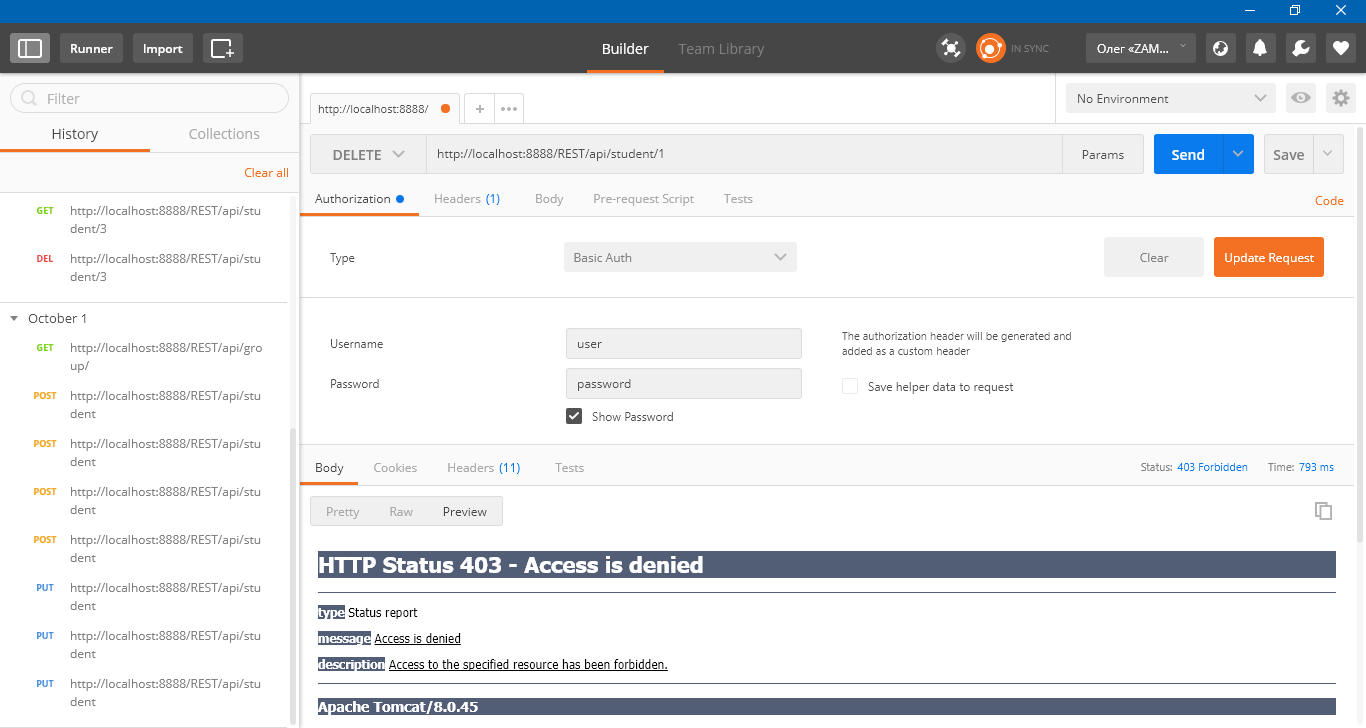


Однако при обращении к методу, помеченному аннотацией с параметром hasRole и hasAnyRole, требуется передача имени пользователя и пароля в каждом запросе.

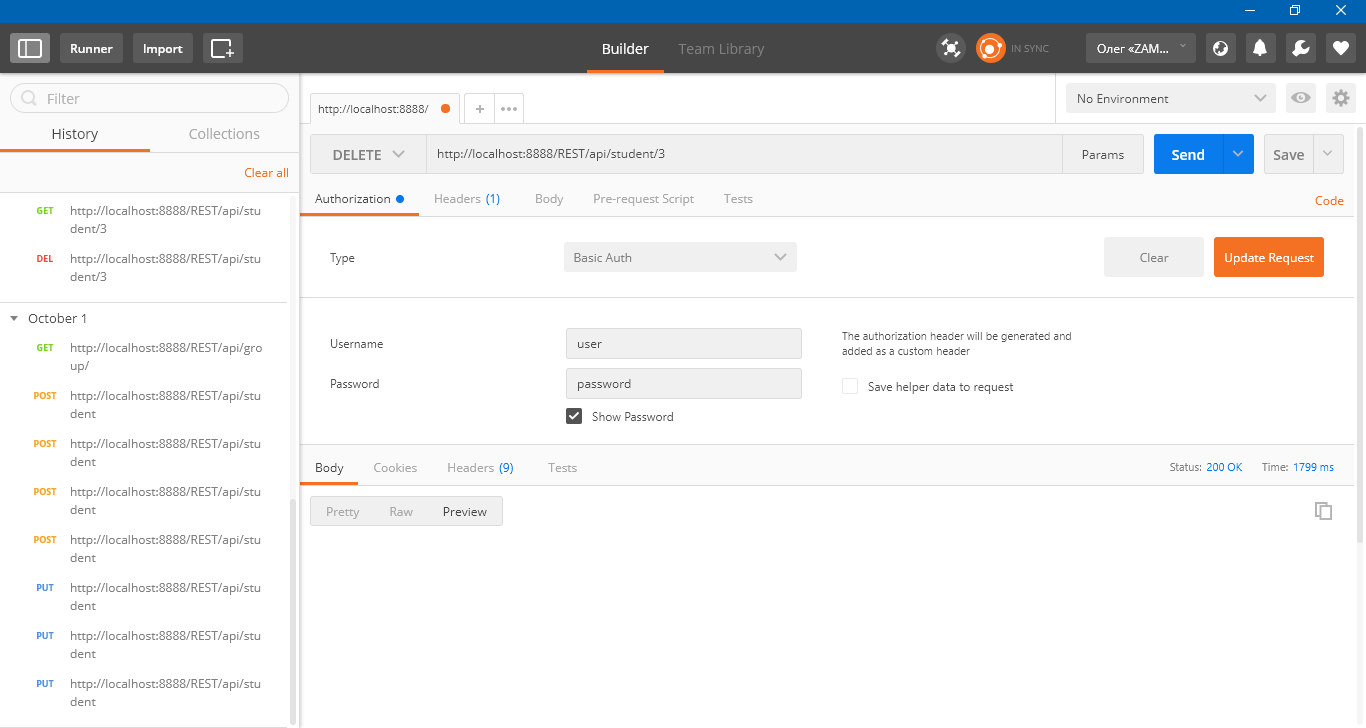
В случае если имя и пароль не передать, сервер вернет 401:



Если же пользователь не входит в группу разрешенных, то ответом будет 403:



При совпадении имени пользователя, пароля и требуемой роли, запрос будет выполнен:



# Переход к пользователям из базы данных

**Добавить в базу данных таблицы с ролями:**

**CREATE** **TABLE** `users` (

`id` **INT** **NOT** **NULL** AUTO\_INCREMENT,

`username` **varchar**(64) **NOT** **NULL**,

`password` **varchar**(64) **NOT** **NULL**,

`authority` **varchar**(64) **NOT** **NULL**,

**PRIMARY** **KEY** (`id`)

);

**insert** **into** `users` (id, username, password, authority) **values**

(1, 'admin', 'admin', 'ROLE\_ADMIN'),

(2, 'user', 'user','ROLE\_USER');

**Создать сущность User**

@Entity

@Table(name = "users")

**public** **class** User **extends** AbstractEntity {

@Column(nullable = **false**, unique = **true**)

**private** String username;

@Column(nullable = **false**)

**private** String password;

@Column(nullable = **false**)

**private** String authority;

**public** User() {

}

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** String getAuthority() {

**return** authority;

}

**public** **void** setAuthority(String authority) {

**this**.authority = authority;

}

}

**Создать Repository для User**

**public** **interface** UserRepository **extends** JpaRepository<User, Long> {

User findByUsername(String username);

}

**Создать свой UserDetailSercie в пакете security**

@Service

**public** **class** CustomUserDetailService **implements** UserDetailsService {

@Autowired

**private** UserRepository repository;

@Override

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

by.vstu.isap.zamok.lab5.entity.User user = repository.findByUsername(username);

**if** (user == **null**) {

**throw** **new** UsernameNotFoundException(username);

}

List<SimpleGrantedAuthority> grantedAuthorities = **new** ArrayList<>();

grantedAuthorities.add(**new** SimpleGrantedAuthority(user.getAuthority()));

**return** **new** User(user.getUsername(), user.getPassword(), grantedAuthorities);

}

}

**Убрать InMemoryUserDetailsManager и добавить сервис, связанный с БД в WebSecurityConfig:**

@EnableGlobalMethodSecurity(prePostEnabled = **true**)

@EnableWebSecurity

**public** **class** WebSecurityConfig **extends** WebSecurityConfigurerAdapter {

@Autowired

**private** CustomUserDetailService service;

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.httpBasic().and().authorizeRequests().antMatchers("/api/group/\*\*", "/api/student/\*\*").permitAll().and().csrf().disable();

}

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(service);

}

}

**Задание**: добавить пользователя USER, который бы имел доступ к методам, не изменяющим данные, пользователю ADMIN разрешить и просматривать и изменять данные, а неавторизированным пользователям запретить всё.