Всем привет!

В этой статья я продемонстрирую основные компоненты для создания Reactive RESTful миксросервисов, используя Spring Boot WebFlux, Spring Boot Security, Spring Cloud Netflix Eureka (Service Discovery), Hystrix (Circuit Breaker), Ribbon (Client Side Load Balancer), External Configuration (через git repository), Spring Cloud Sleuth, Spring Cloud Gateway, Spring Boot Reactive MongoDB. А также Spring Boot Admin и Zipkin для мониторинга.
<cut/>
Данный обзор был сделан после изучения книг Spring Microservices in Action и Hands-On Spring 5 Security for Reactive Applications.

В этой статье мы создадим элементарное приложение с тремя запросами: получить список игр, получить список игроков, создать игру из id игроков, запрос для проверки отката (Hystrix fallback) в случае долгого ожидания ответа. И реализацию аутентификации через JWT токен по мотивам книги Hands-On Spring 5 Security for Reactive Applications.

Я не буду расписывать как создается каждое приложение в IDE, так как эта статья рассчитана на опытного пользователя.

Структура проекта

Проект состоит из двух модулей. Модуль `spring-servers` можно смело копировать из проекта в проект. Там почти нет кода и конфигураций. Модуль `tictactoe-services` содержит модули и микросервисы нашего приложения. Сразу замечу, что добавляя в сервисы модули `auth-module` и `domain-module` я нарушаю один из принципов микросервисной архитектуры об автономности микросервисов. Но на этапе разработки этих модулей, я считаю, что это самое оптимальное решение.

Конфигурация Gradle

Мне нравится когда вся конфигурация Gradle находится в одном файле, поэтому я сконфигурировал весь проект в одном `build.gradle`.

<spoiler title="build.gradle">
```java
buildscript {
    ext {
        springBootVersion = '2.1.1.RELEASE'
        gradleDockerVersion = '0.20.1'
    }
    repositories {
        mavenCentral()
        maven { url "https://plugins.gradle.org/m2/" }
    }
    dependencies {
        classpath("org.springframework.boot:spring-boot-gradle-plugin:${springBootVersion}")
        classpath("gradle.plugin.com.palantir.gradle.docker:gradle-docker:${gradleDockerVersion}")
    }
}

allprojects {
    group = 'com.tictactoe'

    apply plugin: 'java'
    apply plugin: 'eclipse'
    apply plugin: 'org.springframework.boot'
    apply plugin: 'io.spring.dependency-management'
    apply plugin: 'com.palantir.docker'
```

```
    apply plugin: 'com.palantir.docker-run'
    apply plugin: 'com.palantir.docker-compose'
}

docker.name = 'com.tictactoe'
bootJar.enabled = false

sourceCompatibility = 11

repositories {
    mavenCentral()
    maven { url "https://repo.spring.io/milestone" }
}

subprojects {
    ext['springCloudVersion'] = 'Greenwich.M3'

    sourceSets.configureEach { sourceSet ->
        tasks.named(sourceSet.compileJavaTaskName, {
            options.annotationProcessorGeneratedSourcesDirectory = file("$buildDir/generated/
sources/annotationProcessor/java/${sourceSet.name}")
        })
    }

    repositories {
        mavenCentral()
        maven { url "https://repo.spring.io/milestone" }
    }

    dependencyManagement {
        imports {
            mavenBom "org.springframework.cloud:spring-cloud-dependencies:$
{springCloudVersion}"
        }
    }

    dependencies {
        compile fileTree(include: ['*.jar'], dir: 'libs')

        compileOnly('org.projectlombok:lombok')
        annotationProcessor('org.projectlombok:lombok')
    }
}

project(':spring-servers') {
    bootJar.enabled = false

    task cleanAll {
        dependsOn subprojects*.tasks*.findByName('clean')
    }

    task buildAll {
        dependsOn subprojects*.tasks*.findByName('build')
    }

    dockerCompose {
        template 'docker-compose.spring-servers.template.yml'
        dockerComposeFile 'docker-compose.spring-servers.yml'
    }
}
```

```
project(':tictactoe-services') {
    bootJar.enabled = false

    task cleanAll {
        dependsOn subprojects*.tasks*.findByName('clean')
    }

    task buildAll {
        dependsOn subprojects*.tasks*.findByName('build')
    }
}

// Tictactoe Modules
project(':tictactoe-services:domain-module') {
    bootJar.enabled = false

    jar {
        enabled = true
        group 'com.tictactoe'
        baseName = 'domain-module'
        version = '1.0'
    }

    dependencies {
        implementation('org.springframework.boot:spring-boot-starter-security')
        implementation('org.springframework.boot:spring-boot-starter-data-mongodb-reactive')
        implementation('org.springframework.boot:spring-boot-starter-validation')

        implementation('com.fasterxml.jackson.core:jackson-annotations:2.9.3')

        implementation 'com.intellij:annotations:+@jar'

        compileOnly('org.projectlombok:lombok')

        testCompile group: 'junit', name: 'junit', version: '4.12'
    }
}

project(':tictactoe-services:auth-module') {
    bootJar.enabled = false

    jar {
        enabled = true
        baseName = 'auth-module'
        version = '1.0'
    }

    dependencies {
        implementation project(':tictactoe-services:domain-module')

        implementation('org.springframework.boot:spring-boot-starter-webflux')
        implementation('org.springframework.boot:spring-boot-starter-data-mongodb-reactive')
        implementation('org.springframework.boot:spring-boot-starter-security')

        implementation('org.springframework.cloud:spring-cloud-starter-netflix-ribbon')

        implementation('org.springframework.security:spring-security-oauth2-core')
        implementation('org.springframework.security:spring-security-oauth2-jose')
```

```
        implementation 'com.intellij:annotations:+@jar'

        testImplementation('org.springframework.boot:spring-boot-starter-test')
        testImplementation('io.projectreactor:reactor-test')
        testImplementation('org.springframework.security:spring-security-test')
    }
}

project(':tictactoe-services:user-service') {
    bootJar {
        launchScript()
        baseName = 'user-service'
        version = '0.1.0'
    }

    dependencies {
        implementation project(':tictactoe-services:domain-module')
        implementation project(':tictactoe-services:auth-module')
    }
}

project(':tictactoe-services:game-service') {
    bootJar {
        launchScript()
        baseName = 'game-service'
        version = '0.1.0'
    }

    dependencies {
        implementation project(':tictactoe-services:domain-module')
        implementation project(':tictactoe-services:auth-module')
    }
}

project(':tictactoe-services:webapi-service') {
    bootJar {
        launchScript()
        baseName = 'webapi-service'
        version = '0.1.0'
    }

    dependencies {
        implementation project(':tictactoe-services:domain-module')
        implementation project(':tictactoe-services:auth-module')
    }
}

// Spring Servers
project(':spring-servers:discovery-server') {
    bootJar {
        launchScript()
        baseName = 'discovery-server'
        version = '0.1.0'
    }

    dependencies {
        implementation('org.springframework.cloud:spring-cloud-starter-netflix-eureka-server')
        implementation('org.springframework.boot:spring-boot-starter-security')

        compile('javax.xml.bind:jaxb-api:2.3.0')
```

```
        compile('javax.activation:activation:1.1')
        compile('org.glassfish.jaxb:jaxb-runtime:2.3.0')

        testImplementation('org.springframework.boot:spring-boot-starter-test')
    }
}

project(':spring-servers:config-server') {
    bootJar {
        launchScript()
        baseName = 'config-server'
        version = '0.1.0'
    }

    dependencies {
        implementation('org.springframework.boot:spring-boot-starter-security')

        implementation('org.springframework.cloud:spring-cloud-config-server')
        implementation('org.springframework.cloud:spring-cloud-starter-config')
        implementation('org.springframework.cloud:spring-cloud-starter-netflix-eureka-client')

        testImplementation('org.springframework.boot:spring-boot-starter-test')
    }
}

project(':spring-servers:gateway-server') {
    bootJar {
        launchScript()
        baseName = 'gateway-server'
        version = '0.1.0'
    }

    dependencies {
        implementation('org.springframework.boot:spring-boot-starter-webflux')
        implementation('org.springframework.boot:spring-boot-starter-actuator')

        implementation('org.springframework.cloud:spring-cloud-starter-gateway')
        implementation('org.springframework.cloud:spring-cloud-starter-config')
        implementation('org.springframework.cloud:spring-cloud-starter-netflix-ribbon')
        implementation('org.springframework.cloud:spring-cloud-starter-netflix-eureka-client')

        testImplementation('org.springframework.boot:spring-boot-starter-test')
    }
}

project(':spring-servers:admin-server') {
    ext['springBootAdminVersion'] = '2.1.1'

    bootJar {
        launchScript()
        baseName = 'admin-server'
        version = '0.1.0'
    }

    dependencies {
        implementation('org.springframework.boot:spring-boot-starter-web')
        implementation('org.springframework.boot:spring-boot-starter-security')

        implementation('de.codecentric:spring-boot-admin-starter-server')
```

```
        implementation('org.springframework.cloud:spring-cloud-starter-config')
        implementation('org.springframework.cloud:spring-cloud-starter-netflix-eureka-client')

        testImplementation('org.springframework.boot:spring-boot-starter-test')
        testImplementation('org.springframework.security:spring-security-test')
    }

    dependencyManagement {
        imports {
            mavenBom "de.codecentric:spring-boot-admin-dependencies:$
{springBootAdminVersion}"
        }
    }
}

subprojects { subproject ->
    if (file("${subproject.projectDir}/docker/Dockerfile").exists()) {
        docker {
            // workingbit - replace with your dockerhub's username
            name "workingbit/${subproject.group}.${subproject.bootJar.baseName}"
            tags 'latest'
            dockerfile file("${subproject.projectDir}/docker/Dockerfile")
            files tasks.bootJar.archivePath, 'docker/run.sh'
            buildArgs "JAR_FILE": "${subproject.bootJar.baseName}-$
{subproject.bootJar.version}.jar",
                    "RUN_SH": "run.sh"
        }
    } else {
        docker.name = 'noop'
    }

    if (subproject.name.endsWith('service')) {
        dependencies {
            implementation('org.springframework.boot:spring-boot-starter-actuator')
            implementation('org.springframework.boot:spring-boot-starter-webflux')
            implementation('org.springframework.boot:spring-boot-starter-data-mongodb-reactive')
            implementation('org.springframework.boot:spring-boot-starter-security')

            implementation('org.springframework.security:spring-security-oauth2-core')
            implementation('org.springframework.security:spring-security-oauth2-jose')

            implementation('org.springframework.cloud:spring-cloud-starter-config')
            implementation('org.springframework.cloud:spring-cloud-starter-netflix-eureka-client')
            implementation('org.springframework.cloud:spring-cloud-starter-netflix-hystrix')
            implementation('org.springframework.cloud:spring-cloud-starter-netflix-ribbon')
            implementation('org.springframework.cloud:spring-cloud-starter-sleuth')
            implementation('org.springframework.cloud:spring-cloud-starter-zipkin')

            implementation('org.springframework.security:spring-security-rsa')

            implementation('com.intellij:annotations:+@jar')
            implementation('org.apache.commons:commons-lang3:3.8.1')
            runtimeOnly('org.springframework.boot:spring-boot-devtools')

            testImplementation('org.springframework.boot:spring-boot-starter-test')
            testImplementation('de.flapdoodle.embed:de.flapdoodle.embed.mongo')
            testImplementation('io.projectreactor:reactor-test')
        }
    }
}
```

```
</spoiler>

Использование общего конфигурационного файла позволяет вынести общие для микросервисов зависимости, в данном случае сервисы с именем оканчивающимся на “service”, в одно место. НО, это снова нарушает принцип автономности микросервисов. Кроме общих зависимостей в субпроекты можно добавлять задачи. Я добавил задачи плагина `gradle.plugin.com.palantir.gradle.docker:gradle-docker` для работы с `Docker`.

# Модуль auth-module

Теперь, рассмотрим модуль аутентификации по JWT. Описание пакета `auth` этого модуля можно найти в книге по реактивной аутентификации, которую я указал выше.




А, на пакете `config` остановимся подробней.

# Класс “сложных” свойств ApplicationClientsProperties.java

```
@Data
@Component
@ConfigurationProperties("appclients")
public class ApplicationClientsProperties {
    private List<ApplicationClient> clients = new ArrayList<>();

    @Data
    public static class ApplicationClient {
        private String username;
        private String password;
        private String[] roles;
    }

}
```

Этот класс содержит “сложные” свойства для конфигурации inMemory базы данных.

Класс конфигурации модуля AuthModuleConfig.java

```
@Data
@Configuration
@PropertySource("classpath:moduleConfig.yml")
public class AuthModuleConfig {

    @Value("${tokenExpirationMinutes:60}")
    private Integer tokenExpirationMinutes;

    @Value("${tokenIssuer:workingbit-example.com}")
    private String tokenIssuer;

    @Value("${tokenSecret:secret}") // length minimum 256 bites
    private String tokenSecret;
}
```

В файле ресурсов необходимо указать данные переменные. В моей конфигурации токен протухает через 10 часов.
```

Класс конфигурации матчеров фильтров MicroserviceServiceJwtAuthWebFilter.java

```
public class MicroserviceServiceJwtAuthWebFilter extends JwtAuthWebFilter {

    private final String[] matchersStrings;

    public MicroserviceServiceJwtAuthWebFilter(JwtService jwtService, String[] matchersStrings) {
        super(jwtService);
        this.matchersStrings = matchersStrings;
    }

    @Override
    protected ServerWebExchangeMatcher getAuthMatcher() {
        List<ServerWebExchangeMatcher> matchers = Arrays.stream(this.matchersStrings)
                .map(PathPatternParserServerWebExchangeMatcher::new)
                .collect(Collectors.toList());
        return ServerWebExchangeMatchers.matchers(new
OrServerWebExchangeMatcher(matchers));
    }
}
```

В этот фильтр, при конструкции, передается сервис для работы с JWT и список путей, которые будет обрабатывать этот фильтр.

Класс конфигурации Reactive Spring Boot Security MicroserviceSpringSecurityWebFluxConfig.java

```
@ConditionalOnProperty(value = "microservice", havingValue = "true")
@EnableReactiveMethodSecurity
@PropertySource(value = "classpath:/application.properties")
public class MicroserviceSpringSecurityWebFluxConfig {

    @Value("${whiteListedAuthUrls}")
    private String[] whiteListedAuthUrls;
    @Value("${jwtTokenMatchUrls}")
    private String[] jwtTokenMatchUrls;

    /**
     * Bean which configures whiteListed and JWT filter urls
     * Also it configures authentication for Actuator. Actuator takes configured
AuthenticationManager automatically
     * which uses MapReactiveUserDetailsService to configure inMemory users
     */
    @Bean
    public SecurityWebFilterChain springSecurityFilterChain(
            ServerHttpSecurity http, JwtService jwtService
    ) {
        MicroserviceServiceJwtAuthWebFilter userServiceJwtAuthWebFilter
                = new MicroserviceServiceJwtAuthWebFilter(jwtService, jwtTokenMatchUrls);

        http.csrf().disable();

        http
                .authorizeExchange()
                .pathMatchers(whiteListedAuthUrls)
                .permitAll()
```

```
                .and()
                .authorizeExchange()
                .pathMatchers("/actuator/**").hasRole("SYSTEM")
                .and()
                .httpBasic()
                .and()
                .addFilterAt(userServiceJwtAuthWebFilter, SecurityWebFiltersOrder.AUTHENTICATION);

        return http.build();
    }

}
```

Здесь есть сразу три интересные аннотации.

```
@ConditionalOnProperty(value = "microservice", havingValue = "true")
```

Аннотация, которая подключает этот модуль в зависимости от переменной microservice в файле конфигураций, который указан в аннотации. Это необходимо для того, чтобы в некоторых модулях общую проверку по токену. В данном приложении это сервис `webapi-service`, который имеет свою реализацию бина `SecurityWebFilterChain`.

```
@PropertySource(value = "classpath:/application.properties")
```

Так же эта аннотация позволяет брать свойства из главного сервиса в который импортируется этот модуль. Другими словами, переменные

```
    @Value("${whiteListedAuthUrls}")
    private String[] whiteListedAuthUrls;
    @Value("${jwtTokenMatchUrls}")
    private String[] jwtTokenMatchUrls;
```

Берут свои значения из конфигурации микросервиса.

И, аннотация, которая позволяет навешивать аннотации безопасности `@PreAuthorize(“hasRole(‘MY_ROLE’)”)`

```
@EnableReactiveMethodSecurity
```

И в этом модуле создается бин `SecurityWebFilterChain`, который выполняет конфигурацию доступа к актуатору, разрешенным url и url на которых выполняется проверка JWT токена. Нужно заметить, что доступ к фильтру JWT токена должен быть открыт.

Конфигурация SpringWebFluxConfig.java

В этой конфигурации создаются бины `MapReactiveUserDetailsService` для конфигурации актуатора и других системных пользователей в памяти.

```
@Bean
@Primary
public MapReactiveUserDetailsService userDetailsRepositoryInMemory() {
  List<UserDetails> users = applicationClients.getClients()
```

```
        .stream()
        .map(applicationClient ->
                User.builder()
                        .username(applicationClient.getUsername())
                        .password(passwordEncoder().encode(applicationClient.getPassword()))
                        .roles(applicationClient.getRoles()).build())
        .collect(toList());
  return new MapReactiveUserDetailsService(users);
}
```

Бин `ReactiveUserDetailsService` который необходим для сшивки репозитория нашего пользователя с `Spring Security`.

```
@Bean
public ReactiveUserDetailsService userDetailsRepository(UserRepository users) {
  return (email) -> users.findByEmail(email).cast(UserDetails.class);
}
```

Бин создания `WebClient` - клиента для выполнения реактивных запросов.

```
@Bean
public WebClient loadBalancedWebClientBuilder(JwtService jwtService) {
  return WebClient.builder()
        .filter(lbFunction)
        .filter(authorizationFilter(jwtService))
        .build();
}

private ExchangeFilterFunction authorizationFilter(JwtService jwtService) {
  return ExchangeFilterFunction
        .ofRequestProcessor(clientRequest ->
                ReactiveSecurityContextHolder.getContext()
                        .map(securityContext ->
                                ClientRequest.from(clientRequest)
                                        .header(HttpHeaders.AUTHORIZATION,

jwtService.getHttpAuthHeaderValue(securityContext.getAuthentication()))
                                        .build()));
}
```

Во время создания добавляются два фильтра. `LoadBalancer` и фильтр который берет из контекста `ReactiveSecurityContext` инстанс `Authentication` и из него создает токен для того, чтобы его аутентифицировал фильтр целевого сервер и соответственно авторизовал.

И для удобства работы с типом MongoDB `ObjectId` и датами, я добавил бин создания objectMapper'a:

```
@Bean
@Primary
ObjectMapper objectMapper() {
  Jackson2ObjectMapperBuilder builder = new Jackson2ObjectMapperBuilder();
  builder.serializerByType(ObjectId.class, new ToStringSerializer());
  builder.deserializerByType(ObjectId.class, new JsonDeserializer() {
    @Override
```

```
    public Object deserialize(JsonParser p, DeserializationContext ctxt) throws IOException {
      Map oid = p.readValueAs(Map.class);
      return new ObjectId(
            (Integer) oid.get("timestamp"),
            (Integer) oid.get("machineIdentifier"),
            ((Integer) oid.get("processIdentifier")).shortValue(),
            (Integer) oid.get("counter"));
    }
  });
  builder.featuresToDisable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
  return builder.build();
}
```

Микросервис game-service

Микросервис game-service имеет следующую структуру:

Как можно увидеть в нем только один файл конфигурации ApplicationConfig

Конфигуратор ApplicationConfig.java

```
@Data
@Configuration
@EnableReactiveMongoRepositories("com.tictactoe.gameservice.repository")
@Import({ApplicationClientsProperties.class, SpringWebFluxConfig.class,
MicroserviceSpringSecurityWebFluxConfig.class})
public class ApplicationConfig {

    @Value("${userserviceUrl}")
    private String userServiceUrl;

}
```

В нем содержится переменная с адресом сервиса `user-service` и есть две интересные аннотации:

```
@EnableReactiveMongoRepositories("com.tictactoe.gameservice.repository")
```

Эта аннотация необходим для того, чтобы указать конфигуратору репозиторий MongoDB.

```
@Import({ApplicationClientsProperties.class, SpringWebFluxConfig.class,
MicroserviceSpringSecurityWebFluxConfig.class})
```

Эта аннотация импортирует конфигурации из модуля `auth-module`.

Сервис GameService.java

В этом сервисе есть только следующий интересный код:

```
    @HystrixCommand
    public Flux<Game> getAllGames() {
```

```
        return gameRepository.findAll();
    }

    @HystrixCommand(fallbackMethod = "buildFallbackAllGames",
            threadPoolKey = "licenseByOrgThreadPool",
            threadPoolProperties =
                    {@HystrixProperty(name = "coreSize", value = "30"),
                            @HystrixProperty(name = "maxQueueSize", value = "10")},
            commandProperties = {
                    @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "10"),
                    @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "75"),
                    @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value =
"7000"),
                    @HystrixProperty(name = "metrics.rollingStats.timeInMilliseconds", value = "15000"),
                    @HystrixProperty(name = "metrics.rollingStats.numBuckets", value = "5")}
    )
    public Flux<Game> getAllGamesLong() {
    // logger.debug("LicenseService.getLicensesByOrg  Correlation id: {}",
UserContextHolder.getContext().getCorrelationId());
        randomlyRunLong();
        return gameRepository.findAll();
    }
```

Этот метод случайно выбрасывает исключение и Hystrix в соответствии с аннотацией возвращает результат работы следующего метода:

```
private Flux<Game> buildFallbackAllGames() {
    User fakeUserBlack = new User("fakeUserBlack", "password", Collections.emptyList());
    User fakeUserWhite = new User("fakeUserBlack", "password", Collections.emptyList());
    Game game = new Game(fakeUserBlack, fakeUserWhite);
    List<Game> games = List.of(game);
    return Flux.fromIterable(games);
}
```

Как говорилось в упомянутой выше книге, если что-то сломалось, тогда давайте лучше покажем закешированные или альтернативные данные, чем ничего.

Микросервис webapi-service

Это своеобразный middleware между Gateway и внутренними микросервисами, которые не видны снаружи. Цель этого сервиса получить выборку с других сервисов и сформировать на ее основе ответ пользователю.

Начнем рассмотрение с конфигурации.

Конфигурация SpringSecurityWebFluxConfig.java

```
@Configuration
@EnableReactiveMethodSecurity
public class SpringSecurityWebFluxConfig {

    private static final String AUTH_TOKEN_PATH = "/auth/token";

    @Value("${whiteListedAuthUrls}")
    private String[] whiteListedAuthUrls;
```

```
    @Value("${jwtTokenMatchUrls}")
    private String[] jwtTokenMatchUrls;

    @Bean
    @Primary
    public SecurityWebFilterChain systemSecurityFilterChain(
            ServerHttpSecurity http, JwtService jwtService,
            @Qualifier("userDetailsRepository") ReactiveUserDetailsService userDetailsService
    ) {
```

Здесь мы создаем менеджер аутентификации с сервисов `userDetailsService`, который мы определили ранее в модуле `auth-module`.

```
        UserDetailsRepositoryReactiveAuthenticationManager authenticationManager
                = new UserDetailsRepositoryReactiveAuthenticationManager(userDetailsService);
```

И создаем фильтр с этим менеджером, а также добавляем конвертер инстанса Authentication для того, чтобы получить данные пользователя закодированные в `x-www-form-urlencoded`.

```
        AuthenticationWebFilter tokenWebFilter = new
AuthenticationWebFilter(authenticationManager);
        tokenWebFilter.setServerAuthenticationConverter(exchange ->
                Mono.justOrEmpty(exchange)
                        .filter(ex ->
AUTH_TOKEN_PATH.equalsIgnoreCase(ex.getRequest().getPath().value()))
                        .flatMap(ServerWebExchange::getFormData)
                        .filter(formData -> !formData.isEmpty())
                        .map((formData) -> {
                            String email = formData.getFirst("email");
                            String password = formData.getFirst("password");
                            return new UsernamePasswordAuthenticationToken(email, password);
                        })
        );
```

Добавляем обработчик успешной авторизации суть которого положить JWT токен в сгенерированный из `Authentication` заголовок запроса, чтобы аутентифицироваться можно было только по валидному гостевому токену.

```
        tokenWebFilter.setAuthenticationSuccessHandler(new JwtAuthSuccessHandler(jwtService));

        MicroserviceServiceJwtAuthWebFilter webApiJwtServiceWebFilter = new
MicroserviceServiceJwtAuthWebFilter(jwtService, jwtTokenMatchUrls);
        http.csrf().disable();

        http
                .authorizeExchange()
```

Разрешаем адреса из белого списка. Как я писал ранее адреса, которые будут обрабатываться JWT фильтром так же надо открывать

```
                .pathMatchers(whiteListedAuthUrls)
                .permitAll()
                .and()
                .authorizeExchange()
```

Защищаем базовой аутентификацией актуатор и некоторые адреса

```
            .pathMatchers("/actuator/**").hasRole("SYSTEM")
            .pathMatchers(HttpMethod.GET, "/url-protected/**").hasRole("GUEST")
            .pathMatchers(HttpMethod.POST, "/url-protected/**").hasRole("USER")
            .and()
            .httpBasic()
            .and()
            .authorizeExchange()
```

Делаем обязательной аутентификацию для доступа к токену

```
            .pathMatchers(AUTH_TOKEN_PATH).authenticated()
            .and()
```

Добавляем фильтры. Для аутентификации и проверки JWT токена.

```
            .addFilterAt(webApiJwtServiceWebFilter, SecurityWebFiltersOrder.AUTHENTICATION)
            .addFilterAt(tokenWebFilter, SecurityWebFiltersOrder.AUTHENTICATION);

        return http.build();
    }
```

И как писал выше, этот сервис отключает общую для остальных сервисов проверку JWT токена, указав значение переменной `micoservice=false` в файле `application.properites`.

Контроллер выдачи токенов, регистрации и авторизации AuthController.java

Описывать этот контроллер я не буду, так как он сугубо специфичен.

Сервис WebApiService.java

Этот сервис вызывается в контроллере `WebApiMethodProtectedController.jav`a и имеет интересную аннотацию:

```
@PreAuthorize("hasRole('GUEST')")
public Flux<User> getAllUsers() {
}
```

Эта аннотация разрешает доступ к методы только авторизованным пользователям с ролью гость.

Как тестировать

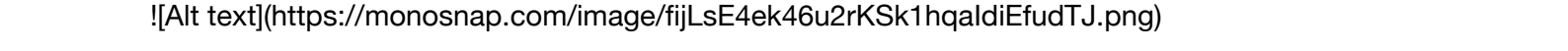
Создайте окружение:

Получите токен

Обновите в окружении переменную TOKEN полученным токеном.

Зарегистрируйте нового пользователя

После регистрации вы получите токен пользователя. Он истекает через 10 часов. Когда он истечет нужно получить новый. Для этого снова запросите гостевой токен, обновите окружение и выполните запрос



Далее, можно получить список пользователей, игр или создать новую игру. А так же протестировать Hystrix, посмотреть конфиги сервисов и зашифровать переменные для git репозитория.

Ссылки

* [Репозиторий на GitHub](https://github.com/lynx-r/tictactoe-microservices-example)
* [Репозиторий конфигураций](https://github.com/lynx-r/tictactoe-config-repo)
* [Spring Microservices in Action](https://www.manning.com/books/spring-microservices-in-action)
* [Hands-On Spring 5 Security for Reactive Applications](https://www.packtpub.com/application-development/hands-spring-security-5-reactive-applications)
* [The code for the book Spring Microservices in Action](https://github.com/carnellj?tab=repositories)
* [The code for the book Hands-On Spring 5 Security for Reactive Applications](https://github.com/lynx-r/Hands-On-Spring-Security-5-for-Reactive-Applications)