Search Tutorials

Spring Boot Security + JWT Hello World Example

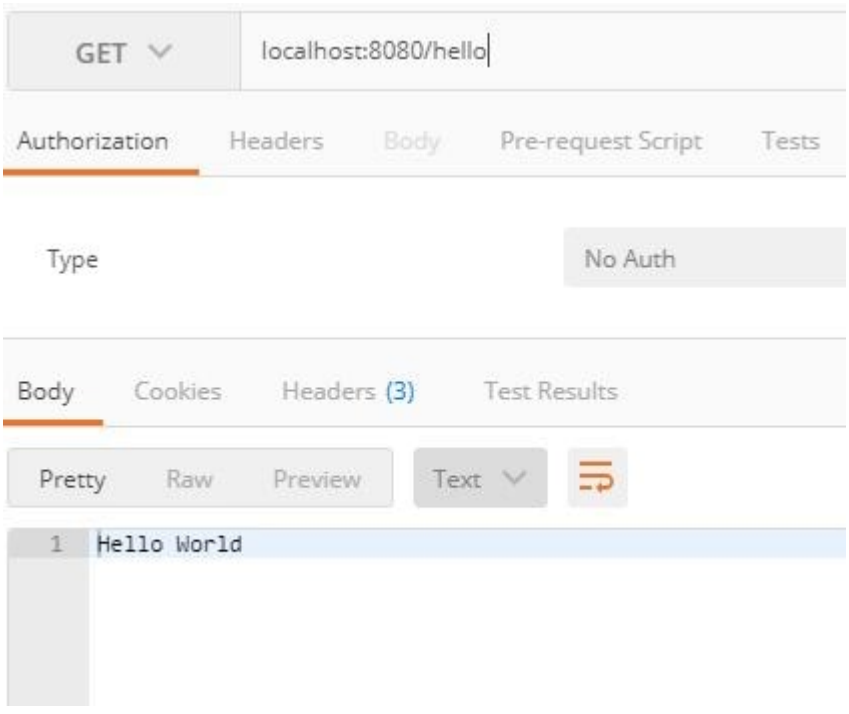
In this tutorial we will be developing a Spring Boot Application that makes use of JWT authentication for securing an exposed REST API. In this example we will be making use of hard coded user values for User Authentication. In next tutorial we will be implementing Spring Boot + JWT + MYSQL JPA for storing and fetching user credentials. (/spring/boot-jwt-mysql) Any user will be able to consume this API only if it has a valid JSON Web Token(JWT). In a previous tutorial we have seen what is JWT, when and how to use it. (/spring/jwt)

Spring Boot JSON Web Token- Table of Contents

```
package com.javainuse;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplicati

@SpringBootApplication
public class SpringBootHelloWorldApplication {

	public static void main(String[] args) {
		SpringApplication.run(SpringBootHelloWorldApplica
	}
}
```

Compile and the run the SpringBootHelloWorldApplication.java as a Java application.

Go to **localhost:8080/hello**



Spring Security and JWT Configuration

We will be configuring Spring Security and JWT for performing 2 operations-

The sequence flow for these operations will be as follows-

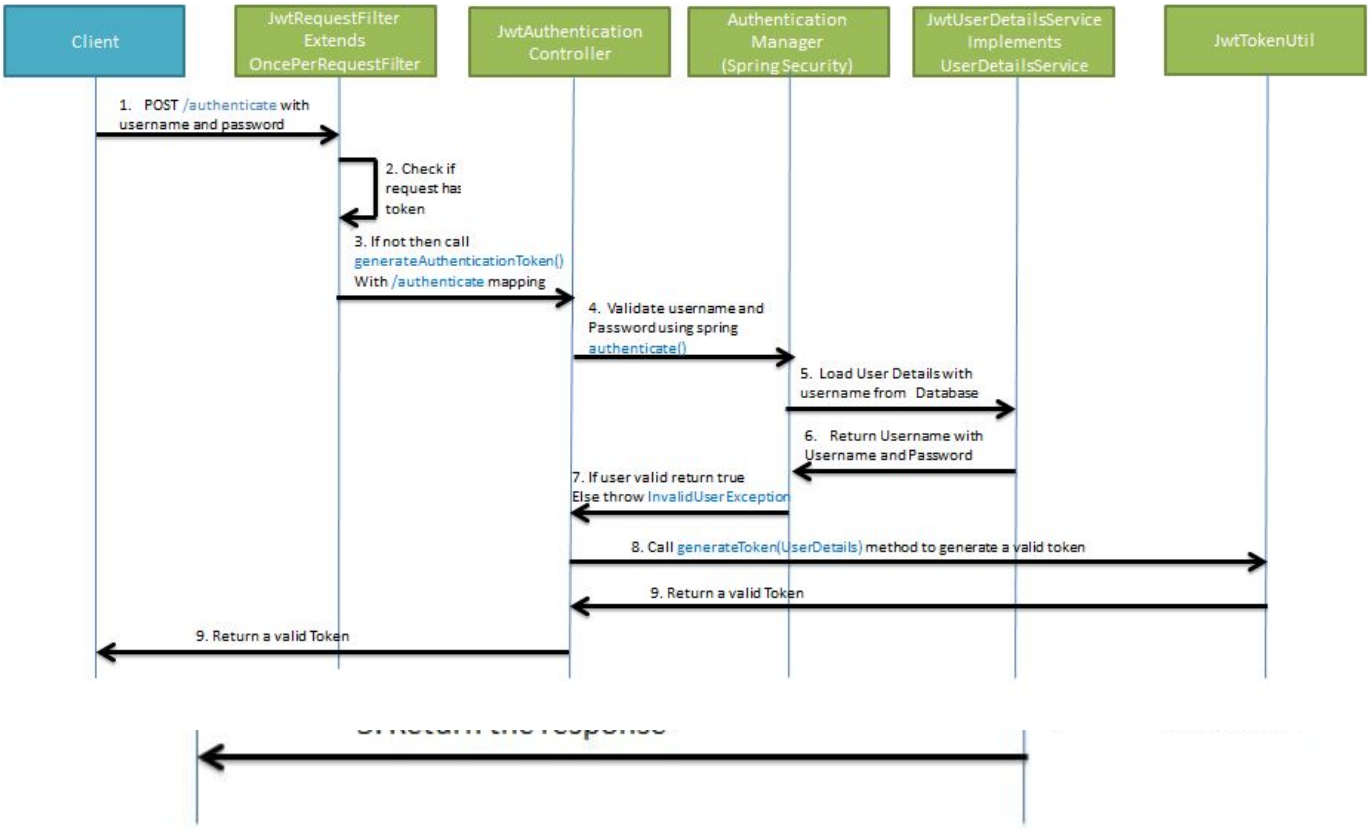
Let's Begin

Generating JWT

For better understanding we will be developing the project in stages

- Develop a Spring Boot Application to expose a Simple REST GET API with mapping /hello.
- Configure Spring Security for JWT. Expose REST POST API with mapping /authenticate using which User will get a valid JSON Web Token. And then

allow the user access to the api /hello only if it has a valid token

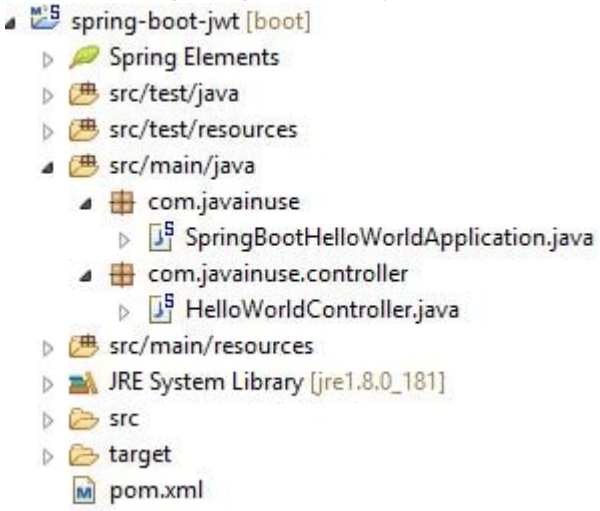


Develop a Spring Boot Application to expose a GET REST API

Validating JWT

Maven Project will be as follows-

Add the Spring Security and JWT dependencies



The pom.xml is as follows-

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="htt
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 htt
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.javainuse</groupId>
	<artifactId>spring-boot-jwt</artifactId>
	<version>0.0.1-SNAPSHOT</version>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactI
		<version>2.1.1.RELEASE</version>
		<relativePath /> <!-- lookup parent from reposito
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.buil
		<project.reporting.outputEncoding>UTF-8</project.
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupI
			<artifactId>spring-boot-starter-web</arti
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupI
			<artifactId>spring-boot-starter-security<
		</dependency>
		<dependency>
			<groupId>io.jsonwebtoken</groupId>
			<artifactId>jjwt</artifactId>
			<version>0.9.1</version>
		</dependency>
	</dependencies>
```

```
</project>
```

```
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloWorldController {

    @RequestMapping({ "/hello" })
    public String hello() {
        return "Hello World";
    }
}
```

- Define the application.properties. As see in previous JWT tutorial, we specify the secret key using which we will be using for hashing algorithm. (/spring/jwt) The secret key is combined with the header and the payload to create a unique hash. We are only able to verify this hash if you have the secret key.

```
jwt.secret=javainuse
```

- **JwtTokenUtil**

  The JwtTokenUtil is responsible for performing JWT operations like creation and validation.It makes use of the io.jsonwebtoken.Jwts for achieving this.

Create the bootstrap class with SpringBoot Annotation

```
package com.javainuse.config;

import java.io.Serializable;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDeta
import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

@Component
public class JwtTokenUtil implements Serializable {

        private static final long serialVersionUID = -2550185

        public static final long JWT_TOKEN_VALIDITY = 5 * 60

        @Value("${jwt.secret}")
        private String secret;

        //retrieve username from jwt token
        public String getUsernameFromToken(String token) {
                return getClaimFromToken(token, Claims::getSu
        }

        //retrieve expiration date from jwt token
        public Date getExpirationDateFromToken(String token)
                return getClaimFromToken(token, Claims::getEx
        }
```

```
    public <T> T getClaimFromToken(String token, Function
            final Claims claims = getAllClaimsFromToken(t
            return claimsResolver.apply(claims);
    }
//for retrieveing any information from token we will need
    private Claims getAllClaimsFromToken(String token) {
            return Jwts.parser().setSigningKey(secret).pa
    }

    //check if the token has expired
    private Boolean isTokenExpired(String token) {
            final Date expiration = getExpirationDateFrom
            return expiration.before(new Date());
    }

    //generate token for user
    public String generateToken(UserDetails userDetails)
            Map<String, Object> claims = new HashMap<>();
            return doGenerateToken(claims, userDetails.ge
    }

    //while creating the token -
    //1. Define  claims of the token, like Issuer, Expira
    //2. Sign the JWT using the HS512 algorithm and secre
    //3. According to JWS Compact Serialization(https://t
    //   compaction of the JWT to a URL-safe string
    private String doGenerateToken(Map<String, Object> cl

            return Jwts.builder().setClaims(claims).setSu
                                .setExpiration(new Date(Syste
                                .signWithSignatureAlgorithm.
    }

    //validate token
    public Boolean validateToken(String token, UserDetail
            final String username = getUsernameFromToken(
```

```
                return (username.equals(userDetails.getUserna
        }
}
```

- **JWTUserDetailsService**

  JWTUserDetailsService implements the Spring Security UserDetailsService interface. It overrides the loadUserByUsername for fetching user details from the database using the username. The Spring Security Authentication Manager calls this method for getting the user details from the database when authenticating the user details provided by the user. Here we are getting the **user details from a hardcoded User List**. In the next tutorial we will be adding the DAO implementation for fetching User Details from the Database. (/spring/boot-jwt-mysql) Also the password for a user is stored in encrypted format using BCrypt. Previously we have seen Spring Boot Security - Password Encoding Using Bcrypt. (/spring/boot_security_jdbc_authentication_bcrypt) Here using the Online Bcrypt Generator you can generate the Bcrypt for a password. (/onlineBcrypt)

```
package com.javainuse.service;

import java.util.ArrayList;

import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDeta
import org.springframework.security.core.userdetails.UserDeta
import org.springframework.security.core.userdetails.Username
import org.springframework.stereotype.Service;

@Service
public class JwtUserDetailsService implements UserDetailsServ

        @Override
        public UserDetails loadUserByUsername(String username
                if ("javainuse".equals(username)) {
                        return new User("javainuse", "$2a$10$
                                        new ArrayList<>());
                } else {
                        throw new UsernameNotFoundException("
                }
        }
}
```

- **JwtAuthenticationController**

  Expose a POST API /authenticate using the JwtAuthenticationController. The POST API gets username and password in the body- Using Spring Authentication Manager we authenticate the username and password.If the credentials are valid, a JWT token is created using the JWTTokenUtil and provided to the client.

```
package com.javainuse.controller;

import java.util.Objects;

import org.springframework.beans.factory.annotation.Autowired
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.Authentica
import org.springframework.security.authentication.BadCredent
import org.springframework.security.authentication.DisabledEx
import org.springframework.security.authentication.UsernamePa
import org.springframework.security.core.userdetails.UserDeta
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController
import com.javainuse.service.JwtUserDetailsService;


import com.javainuse.config.JwtTokenUtil;
import com.javainuse.model.JwtRequest;
import com.javainuse.model.JwtResponse;

@RestController
@CrossOrigin
public class JwtAuthenticationController {

        @Autowired
        private AuthenticationManager authenticationManager;

        @Autowired
        private JwtTokenUtil jwtTokenUtil;

        @Autowired
        private JwtUserDetailsService userDetailsService;
```

```
        @RequestMapping(value = "/authenticate", method = Req
        public ResponseEntity<?> createAuthenticationToken(@R

                authenticate(authenticationRequest.getUsernam

                final UserDetails userDetails = userDetailsSe
                                .loadUserByUsername(authentic

                final String token = jwtTokenUtil.generateTok

                return ResponseEntity.ok(new JwtResponse(toke
        }

        private void authenticate(String username, String pas
                try {
                        authenticationManager.authenticate(ne
                } catch (DisabledException e) {
                        throw new Exception("USER_DISABLED",
                } catch (BadCredentialsException e) {
                        throw new Exception("INVALID_CREDENTI
                }
        }
}
```

- **JwtRequest**

  This class is required for storing the username and password we recieve from the client.

```
package com.javainuse.model;

import java.io.Serializable;

public class JwtRequest implements Serializable {

	private static final long serialVersionUID = 59264685

	private String username;
	private String password;

	//need default constructor for JSON Parsing
	public JwtRequest()
	{

	}

	public JwtRequest(String username, String password) {
		this.setUsername(username);
		this.setPassword(password);
	}

	public String getUsername() {
		return this.username;
	}

	public void setUsername(String username) {
		this.username = username;
	}

	public String getPassword() {
		return this.password;
	}

	public void setPassword(String password) {
		this.password = password;
```

```
        }
}
```

- **JwtResponse**

  This is class is required for creating a response containing the JWT to be returned to the user.

```
package com.javainuse.model;

import java.io.Serializable;

public class JwtResponse implements Serializable {

        private static final long serialVersionUID = -8091879
        private final String jwttoken;

        public JwtResponse(String jwttoken) {
                this.jwttoken = jwttoken;
        }

        public String getToken() {
                return this.jwttoken;
        }
}
```

- **JwtRequestFilter**

  The JwtRequestFilter extends the Spring Web Filter OncePerRequestFilter class. For any incoming request this Filter class gets executed. It checks if the request has a valid JWT token. If it has a valid JWT Token then it sets the Authentication in the context, to specify that the current user is authenticated.

```
package com.javainuse.config;

import java.io.IOException;

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

import org.springframework.beans.factory.annotation.Autowired
import org.springframework.security.authentication.UsernamePa
import org.springframework.security.core.context.SecurityCont
import org.springframework.security.core.userdetails.UserDeta
import org.springframework.security.web.authentication.WebAut
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import com.javainuse.service.JwtUserDetailsService;

import io.jsonwebtoken.ExpiredJwtException;

@Component
public class JwtRequestFilter extends OncePerRequestFilter {

        @Autowired
        private JwtUserDetailsService jwtUserDetailsService;

        @Autowired
        private JwtTokenUtil jwtTokenUtil;

        @Override
        protected void doFilterInternal(HttpServletRequest re
                        throws ServletException, IOException

                final String requestTokenHeader = request.get
```

```
        String username = null;
        String jwtToken = null;
        // JWT Token is in the form "Bearer token". R
        // only the Token
        if (requestTokenHeader != null && requestToke
                jwtToken = requestTokenHeader.substri
                try {
                        username = jwtTokenUtil.getUs
                } catch (IllegalArgumentException e)
                        System.out.println("Unable to
                } catch (ExpiredJwtException e) {
                        System.out.println("JWT Token
                }
        } else {
                logger.warn("JWT Token does not begin
        }

        // Once we get the token validate it.
        if (username != null && SecurityContextHolder

                UserDetails userDetails = this.jwtUse

                // if token is valid configure Spring
                // authentication
                if (jwtTokenUtil.validateToken(jwtTok

                        UsernamePasswordAuthenticatio
                                        userDetails,
                        usernamePasswordAuthenticatio
                                        .setDetails(n
                        // After setting the Authenti
                        // that the current user is a
                        // Spring Security Configurat
                        SecurityContextHolder.getCont
                }
        }
```

```
                chain.doFilter(request, response);
        }

}
```

- **JwtAuthenticationEntryPoint**

  This class will extend Spring's AuthenticationEntryPoint class and override its method commence. It rejects every unauthenticated request and send error code 401

```
package com.javainuse.config;

import java.io.IOException;
import java.io.Serializable;

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

import org.springframework.security.core.AuthenticationExcept
import org.springframework.security.web.AuthenticationEntryPo
import org.springframework.stereotype.Component;

@Component
public class JwtAuthenticationEntryPoint implements Authentic

        private static final long serialVersionUID = -7858869

        @Override
        public void commence(HttpServletRequest request, Http
                        AuthenticationException authException

                response.sendError(HttpServletResponse.SC_UNA
        }
}
```

- **WebSecurityConfig**

  This class extends the WebSecurityConfigurerAdapter is a convenience class that allows customization to both WebSecurity and HttpSecurity.

```
package com.javainuse.config;

import org.springframework.beans.factory.annotation.Autowired
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.Authentica
import org.springframework.security.config.annotation.authent
import org.springframework.security.config.annotation.method.
import org.springframework.security.config.annotation.web.bui
import org.springframework.security.config.annotation.web.con
import org.springframework.security.config.annotation.web.con
import org.springframework.security.config.http.SessionCreati
import org.springframework.security.core.userdetails.UserDeta
import org.springframework.security.crypto.bcrypt.BCryptPassw
import org.springframework.security.crypto.password.PasswordE
import org.springframework.security.web.authentication.Usernam

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class WebSecurityConfig extends WebSecurityConfigurerA

        @Autowired
        private JwtAuthenticationEntryPoint jwtAuthentication

        @Autowired
        private UserDetailsService jwtUserDetailsService;

        @Autowired
        private JwtRequestFilter jwtRequestFilter;

        @Autowired
        public void configureGlobal(AuthenticationManagerBuil
                // configure AuthenticationManager so that it
                // user for matching credentials
                // Use BCryptPasswordEncoder
```

```
                auth.userDetailsService(jwtUserDetailsService
        }

        @Bean
        public PasswordEncoder passwordEncoder() {
                return new BCryptPasswordEncoder();
        }

        @Bean
        @Override
        public AuthenticationManager authenticationManagerBea
                return super.authenticationManagerBean();
        }

        @Override
        protected void configure(HttpSecurity httpSecurity) t
                // We don't need CSRF for this example
                httpSecurity.csrf().disable()
                                // dont authenticate this par
                                .authorizeRequests().antMatch
                                // all other requests need to
                                anyRequest().authenticated().
                                // make sure we use stateless
                                // store user's state.
                                exceptionHandling().authentic
                                .sessionCreationPolicy(Sessio

                // Add a filter to validate the tokens with e
                httpSecurity.addFilterBefore(jwtRequestFilter
        }
}
```

Start the Spring Boot Application

- Generate a JSON Web Token -

Create a POST request with url localhost:8080/authenticate. Body should have valid username and password. In our case username is javainuse and password is password.

- **Validate the JSON Web Token**

  - Try accessing the url localhost:8080/hello using the above generated token in the header as follows

Download Source Code

Download it -
Spring Boot + JWT without JPA (/zip/spring/sec/spring-boot-jwt-without-JPA.rar)

Popular Posts

- Spring Boot Interview Questions (/spring/SpringBootInterviewQuestions)
- Implement Spring Boot Security and understand Spring Security Architecture (/webseries/spring-security-jwt/chap3)
- E-commerce Website - Online Book Store using Angular 8 + Spring Boot (/fullstack/ecommerce)

- Spring Boot +JSON Web Token(JWT) Hello World Example (/spring/boot-jwt)
- Angular 7 + Spring Boot Application Hello World Example (/spring/ang7-hello)
- Build a Real Time Chat Application using Spring Boot + WebSocket + RabbitMQ (/spring/boot-websocket-chat)
- Pivotal Cloud Foundry Tutorial - Deploy Spring Boot Application Hello World Example (/pcf/pcf-hello)
- Deploying Spring Based WAR Application to Docker (/devOps/docker/docker-war)
- EIP patterns using Apache Camel (/camel/camel_EIP)
- Spring Cloud- Netflix Eureka + Ribbon Simple Example (/spring/spring_ribbon)
- Spring Cloud- Netflix Hystrix Circuit Breaker Simple Example (/spring/spring_hystrix_circuitbreaker)
- Spring Boot + Swagger Example Hello World Example (/spring/boot_swagger)
- Spring Boot Batch Simple example (/spring/bootbatch)

- Spring Boot + Apache Kafka Example (/spring/spring-boot-apache-kafka-hello-world)
- Spring Boot Admin Simple Example (/spring/boot-admin)

- Spring Boot Security - Introduction to OAuth (/spring/spring-boot-oauth-introduction)
- Spring Boot OAuth2 Part 1 - Getting The Authorization Code (/spring/spring-boot-oauth-authorization-code)
- Spring Boot OAuth2 Part 2 - Getting The Access Token And Using it to Fetch Data. (/spring/spring-boot-oauth-access-token)
- JBoss Drools Hello World-Stateful Knowledge Session using KieSession (/drools_hello_kie)
- Understand Drools Stateful vs Stateless Knowledge Session (/drools_states)
- JBoss Drools- Understanding Drools Decision Table using Simple Example (/drools/drools_decision)

See Also

- Spring Batch Interview Questions (/spring/sprbatch_interview)
- Spring AOP Interview Questions (/spring/spring-AOP-interview-quesions)
- Angular 2 Interview Questions (/angular/ang2_intvw)

- Apache Camel Interview Questions (/camel/Apache_Camel_Questions)
- JBoss Fuse Interview Questions (/camel/JBoss_Fuse_Questions)
- Drools Interview Questions (/drools/drools_intvw)
- Java 8 Interview Questions (/java/java8_intvw)
- Spring Cloud Interview Questions (/spring/spring-cloud-interview-questions)
- Microservices Interview Questions (/spring/microservices-interview-quesions)
- Java HashMap and ConcurrentHashMap Interview Questions (/java/java_map_intvw)
- Snowflake frequently asked interview questions (/prep/snowflake)
- SAP FI - Accounts Receivable frequently asked interview questions (/prep/sap1)
- Top SAP ALV Interview Questions (/prep/sap3)
- Top SAP Business Objects Administration Interview Questions (/prep/sap2)
- EC2 frequently asked interview questions (/prep/ec2)
- Mule ESB frequently asked interview questions (/misc/muleintvw)

- Apache Kafka Interview Questions (/misc/apache-kafka-interview-questions)
- Tosca Testing Tool Interview Questions (/misc/tosca-testing-tool-interview-questions)
- Top Maven Build Tool Interview Questions (/misc/maven-interview-questions)
- Top Gradle Build Tool Interview Questions (/misc/gradle-interview-questions)
- Miscellaneous Topics (/misc)