Spring Boot REST

How services are delivered?

* For example, You want to create an application, which can create a pdf document for the given data and you want to deliver it as a service for other applications.
* So, You can package the application as distributable file format like jar file, and you can place it in some public domain.
* The other applications, who requires this service, will download and will add it as a dependency and can use the service.
* So, basically an application services are made available for other applications as jar files.

limitations:

* If service is upgraded to a new version , then all the applications which are dependent on that service also need to be updated to avail the services added/updated in the new version.
* The services provided as jars are language dependent. So, only Java applications can utilize the services.
* If a service is also utilizing a database data, then that service can not be delivered as a jar for other applications.

what are web services?

* web services are just like web applwebications, they execute on a server and provides services to the clients across the internet.
* web services runs on its own address space, will have its own database, and will provide its services or data to the other applications over the internet.
* If a web service is upgraded with new features then the client applications will get the new version of the services or data.
* The web service developed in one language can provide the service to the client applications developed in the same language or in the different languages.
* So, web services is a technology, which is essential in modern interconnected software design, to achieve interoperbility, scalability and flexibility.

what is interoperability?

* The applications developed in different languages, working on different platforms, running on different server can communicate. This is called interoperability.

How web services are different from web applications?

* web applications are created to provide web pages to the users.
* web services are created to share the data/services to the client applications.
* web applications are designed for customer to business interaction (C2B). But web services are desinged for business to business interaction(B2B).

Types of web services:

* There are two approaches to build web services.

1. SOAP(Simple Object Access Protocol)
2. REST(Representational State Transfer)

* SOAP is a protocol. It defines some rules to structure the messages.
* SOAP is a XML-based message format. SOAP messages are sent between the service consumer application and the webservice(service provider) as a payload in the http request/response message.
* SOAP web services are complex and performance is slow, because of

xml processing.

* REST is not a protocol. It is an architectural style of building web services.
* REST uses XML or JSON or other message formats like HTML,plain, etc.
* REST web services are simple, looks like web applications and light weight.
* REST web services provides better performance because lighter payloads are sent over HTTP.

What is Representational State Transfer?

* In REST terms, the data or the services are called resources.
* The resource could be an object like a user or an employee or a customer and it can be represented in different formats like XML or JSON, or HTML, etc.
* So, the server provides the representation of a resource.
* When a client interacts with the server, it is transfering the state of the resource with a representation. The server also returns the state of the resource with a representation. So, that’s why the name is given as Representational State Transfer.
* With Spring REST API, we can build REST webservices or we can build REST clients.
* In Java, if you want to build RESTful webservices, then we have two options.
* 1. JAX-RS API
* 2. Spring REST API
* JAX-RS = Java Api for Xml – Rest Service
* JAX-RS is a specification, it has different implementations like Jersey, AXIS2, CXF, etc..
* Spring REST API is not a specification, it is directly an implementation from Spring framework , to build RESTful webservices.

Spring REST annotations:

1. @RestController
2. @RequestMapping

@GetMapping

@PostMapping

@PutMapping

@DeleteMapping

@PatchMapping

1. @RequestParam
2. @RequestBody
3. @PathVariable
4. @ResponseStatus
5. @ExceptionHandler, etc..

* @RestController = @Controller + @ResponseBody
* @RestController tells the spring that this class handles HTTP request and the returned data should be directly written to the HTTP response body.
* @RequestMapping maps HTTP requests to the handler methods of MVC controllers and REST controllers.
* @RequestMapping can be used at class level and also at method level to specify the request path and HTTP method,etc.

For ex:

@RestController

@RequestMapping(“/api”)

public class HelloController {

@RequestMapping(value=”/hello”, method=RequestMethod.GET)

public String sayHello() {

return “Hello”;

}

}

@GetMapping: Handles GET requests

@PostMapping: Handles POST requests

@PutMapping: Handles PUT requests

@DeleteMapping: Handles DELETE requests

@PatchMapping: Handles PATCH requests

@RequestParam: Binds request parameters to the method arguments.

@RequestBody: Binds the HTTP request body to a method argument.

This annotation de-serializes the data passed in the request body with

a format like XML or JSON or any other format into a Java object.

@PathVariable: While mapping a request path to the controller method, we can define template variables. This @PathVariable binds the template varaibles to method arguments.

@RestController

public class ItemController {

@GetMapping(“/item/{id}”)

public Item getItem(@PathVariable(“id”) int itemId) {

//logic

}

@PostMapping(“/create”)

@ResponseStatus(HttpStatus.CREATED)

public Item create(@RequestBody Item newItem) {

}

}

@ResponseStatus marks that when the controller method is returning the response,

the response should include the given HTTP status code.

@ExceptionHandler marks that a method in the controller class is a exception handler method for a specific exception. The method will handle the exception and returns appropriate response.

For ex:

@RestController

public class ItemController {

@ExceptionHandler(ItemNotFoundException.class)

@ResponseStatus(HttpStatus.NOT\_FOUND)

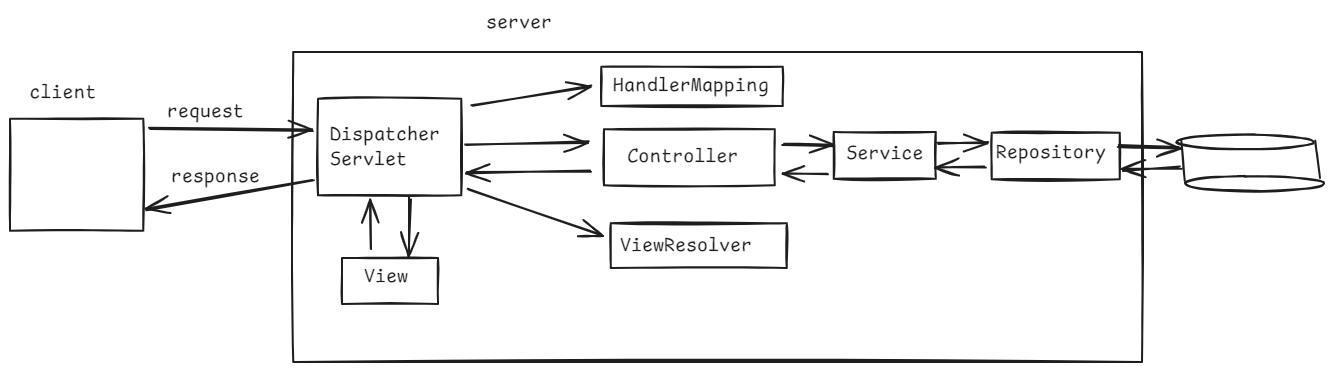
public String handleItemNotFound() {

//logic

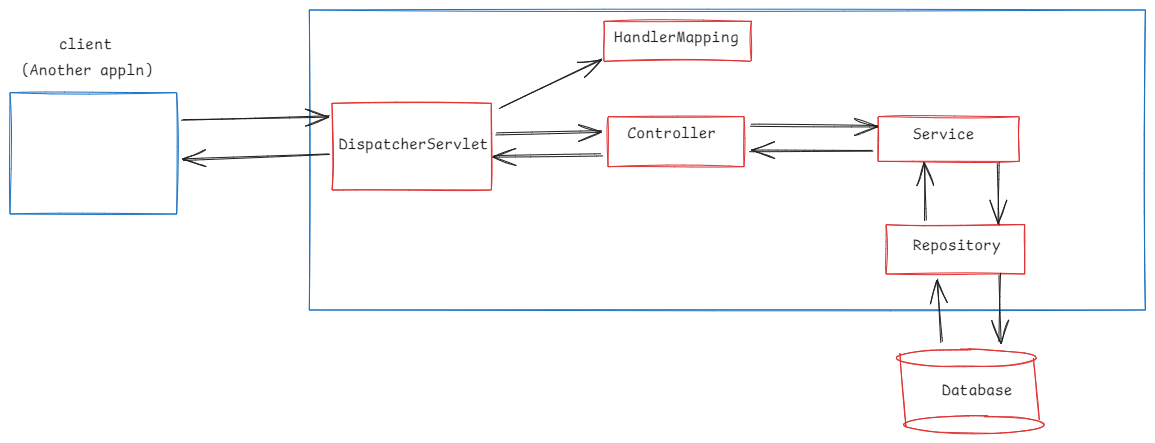
}

}

MVC flow:



REST flow:



YAML file:

* YAML ---- YAML Ain’t Markup Language
* YAML is a more user-friendly and more readable hierarchical data and structured with indentation.
* We can use YAML file as an alternate for a traditional properties file, to configure the application configuration properties.
* Properties file is a simple configuration file, where the settings are stroed in the form

key-value pairs, one per line.

* YAML files is a good fit for defining both a simple or complex configuration settings for an application.
* The file name could be application.yml or application.yaml at src/main/resources.
* The Spring Boot will look for application.properties in the src/main/resources and if it is present, then it reads the properties and configures for the application.
* If Spring Boot finds application.yml , then it reads the properties and configures for the application.
* If both the files are defined and if there is any matching key in both the files then the value loaded from properties file will be overridden with the value from the yaml file.

Github url for Employee CRUD Rest application:

<https://github.com/ShekherJava/SB-EmpCURD-REST.git>

ResponseEntity:

* ResponseEntity is a class in spring framework, it represents the full HTTP response that a REST API can send to the client.
* A ResponseEntity object, includes status code, response headers and response body.
* While creating a rest controller method, we can use ResponseEntity as a return type to the controller method.
* The controller method returns ResponseEntity object to the DispatcherServlet.
* The DispatcherServlet will fetch/retrieve the data from the ResponseEntity object and prepares HTTP response, then sends to the client.

RestTemplate class:

. It is a class in spring framework, which can be used to consume REST API in a spring application.

. With the help of RestTemplate class, a spring application can invoke the different REST endpoints of a service provider application.

. RestTemplate class has provided methods to access the endpoints with various HTTP methods like GET, PUT, POST, DELETE and PATCH.

* some of the methods of RestTemplate class are,

1. getForObject() : By making GET request, it will fetch a resource and converts it into a specified Java object.

for ex:

String url = “http://localhost:8080/api/7878”;

Employee e = restTemplate.getForObject(url, Employee.class);

1. getForEntity(): By making HTTP GET request, it will fetch a resource and returns a ResponseEntity with status code, headers and body.

for ex:

String url= “http://localhost:8080/api/7878”;

ResponseEntity<Employee> re = restTemplate.getForEntity(url, Employee.class);

1. postForObject(): By making HTTP POST request, it will send a Java object in request body and returns a Java object.

For ex:

String postUrl=”http://localhost:8080/api/add”;

Employee newEmployee = new Employee(2323, “ABCD”,4000,20);

String msg = restTemplate.postForObject(postUrl,newEmployee,

String.class);

1. postForEntity(): By making HTTP POST request, it will send a Java object in request body and returns a ResponseEntity object.

for ex:

String postUrl=”http://localhost:8080/api/add”;

Employee newEmployee = new Employee(2323, “ABCD”,4000,20);

ResponseEntity<String> re = restTemplate.postForEntity(postUrl, newEmployee, String.class);

1. exchange(): This method can invoke the given url with a given HTTP method. Returns ResponseEntity object.

<https://github.com/ShekherJava/SB-EmpCRUD-Client.git>

===================================================================

WebClient:

. RestTemplate is a legacy class, can be used to invoke the REST API of a Service provider application, from a spring client application.

. WebClient is a new class introduced in Spring 5, as part of the Spring WebFlux module.

. RestTemplate executes the request in a blocking manner, meaning the thread is blocked until the response is received.

. WebClient operates in a non-blocking manner, meaning the thread is not blocked until the response is received.

. RestTemplate provides synchronous communication and WebClient provides asynchronous communication.

If you're using **Spring Boot** and want to use **WebClient**, make sure you have the following dependency in your pom.xml:

<dependency>

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

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

</dependency>

* **Mono** and **Flux** are core types provided by the **Reactor** library (used in **Spring WebFlux**) for handling asynchronous, non-blocking, and reactive programming.
* Mono is used when you expect **at most one** value (e.g., retrieving a single object).
* If no value is emitted, it completes successfully with or without data.
* If an error occurs during the operation, it propagates the error.

 Flux is used when you expect **a stream of data** or multiple values (e.g., reading a list, a collection, or data in chunks).

* Once all values are emitted, it sends a completion signal.
* If an error occurs, it propagates the error.

<https://github.com/ShekherJava/SB-EmpCRUD-WebClient.git>

swagger

* When a REST API is developed, it must be made available for the clients to access.
* Clients can understand your REST API, with the help of the document provided to the clients.
* This document can be prepared either manually or with the help of a tool.
* To prepare manually, you should use HTML, CSS and Javascript.
* Manual preparation is a time taken and each time when the REST API is updated, the documentation is also need to be updated manually.
* So, you have use a tool like swagger to prepare the documentation.
* Swagger is a powerful tool, it allows to build the document and also consume the REST API’s.
* It provides a User interface to interact with the endpoints and helps the clients to understand the endpoints.
* To integrate swagger with spring boot, we need to use Springfox or Springdoc libraries.
* Springfox library is not property supporting Spring Boot 3.3.x. But Springdoc library supports.
* To add Springdoc library support, we need to add the below dependency to the pom.xml file.

<dependency>

<groupId>org.springdoc</groupId>

<artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>

<version>2.1.0</version>

</dependency>

* The swagger url to access the documentation is,

<http://localhost:8080/swagger-ui/index.html>

* We can customize the documentation, with the help of swagger annotations.

1. @Tag : This annotation can be used to group the related operations under a tag.

. We can configure a name and a description about the group.

. This annotation is used at rest controller class level.

ex:

@Tag(name=”employees”, description=”Operations on employees”)

@RestController

@RequestMapping(“/api”)

public class EmployeeAPI {

* - - -
* - - -

}

1. @Operation : It is used to customize the individual endpoints by adding a short message and a detailed message

ex:

@Tag(name=”employees”, description=”Operations on employees”)

@RestController

@RequestMapping(“/api”)

public class EmployeeAPI {

@Operation(summary=”Get employee by ID,

description=”Fetch employee’s details by its ID”)

@GetMapping(“/{id}”)

public Employee getEmployeeById(@PathVariable int id) {

* - - -
* - - -

}

}

1. @ApiResponse and @ApiResponses :

@ApiResponse is used to customize the response status code.

. To customize multiple response status codes, you have to

wrap up @ApiResponse annotations into @ApiResponses

ex:

@Operation(summary = "Get employee by ID", description = "Fetch employee's details by its ID")

@ApiResponses(

value = {

@ApiResponse(responseCode = "200", description = " Employee successfully retrieved"),

@ApiResponse(responseCode = "404", description = "Employee not found")

})

//produces: defines the type of data, also called media type,

//the method will return in the response.

@GetMapping( value = "/{id}", produces = "application/json")

@ResponseStatus(HttpStatus.***OK***)

**public** Employee getEmployeeById(@PathVariable **int** id) {

**return** service.fetchEmpById(id);

}

1. @Parameter :

It is used to customize the description for the method parameters.

@GetMapping( value = "/{id}", produces = "application/json")

@ResponseStatus(HttpStatus.***OK***)

**public** Employee getEmployeeById(@Parameter(description="ID of the employee to retrieve") @PathVariable **int** id) {

**return** service.fetchEmpById(id);

}

1. @Schema:

It is used to define description for the models and its fields.

ex:

@Schema(description = "Employee model")

**public** **class** Employee {

@Schema(description = "Unique ID of the employee", example ="7101")

@Id

**private** Integer empno;

@Schema(description = "Name of the employee")

**private** String ename;

**private** Double sal;

**private** Integer deptno;

}

1. @Hidden:

. It is used to hide an endpoint from the documentation.

for ex:

@Hidden

@GetMapping("/internal")

**public** String internalEndpoint() {

**return** "This is internal endpoint";

}

==================================================================

Exception Handling in REST API:

1. local Exception handling
2. global Exception handling

. Local exception handling is controller class level. It means, the exception handling is specific to the exceptions raised by that controller class methods.

. @ExceptionHandler is used to create a method inside that controller to handle the exceptions.

. A controller class can have multiple exception handler methods.

. When an exception is thrown by the controller method, then spring will first search for exception handler for the specific type and if it is not found then will search for generic type.

@ExceptionHandler(UserNotFoundException.**class**)

**public** ResponseEntity<String> handleUserNotFoundException(UserNotFoundException ex) {

**return** **new** ResponseEntity<>("Error : "+ ex.getMessage(), HttpStatus.***NOT\_FOUND***);

}

@ExceptionHandler(Exception.**class**)

**public** ResponseEntity<String> handleException(Exception ex) {

**return** **new** ResponseEntity<>("Invalid request : "+ ex.getMessage(), HttpStatus.***BAD\_REQUEST***);

}

. Global exception handling applies to across the entire spring boot application. It is not for a specific controller class-level, it is for the entire application-level.

. we can centralize the error handling logic, by creating global exception handling.

. The annotations for global exception handling are,

@RestControllerAdvice -- used at class level

@ExceptionHandler ---- used at method level

@RestController = @Controller + @ResponseBody

@RestControllerAdvice = @ControllerAdvice + @ResponseBody

For example,

@RestControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(UserNotFoundException.**class**)

**public** ResponseEntity<String> handleUserNotFoundException(UserNotFoundException ex) {

**return** **new** ResponseEntity<>("Error : "+ ex.getMessage(), HttpStatus.***NOT\_FOUND***);

}

@ExceptionHandler(Exception.**class**)

**public** ResponseEntity<String> handleException(Exception ex) {

**return** **new** ResponseEntity<>("Invalid request : "+ ex.getMessage(), HttpStatus.***BAD\_REQUEST***);

}

}

<https://github.com/ShekherJava/SB-REST-Exceptions.git>

Spring Security

. spring security is one of the modules of the spring framework, using which we can provide security for spring mvc and spirng REST applications.

. It provides support for authentication, authorization and other security features.

. authentication is a process of verifying the user credentials.

. authorization is a process of verifying whether a user is permitted to access a particular resource or not.

. Spring security has provided a group of filters, which is also called security filter chain and this chain consists multiple filters and each filter will perform the different aspects of the security.

. We no need to configure all the security filters, instead we need to configure FilterChainProxy component and it will delegate the request to appropriate filters.

. If a user makes a login request, UsernamePasswordAuthenticationFilter, collects the credentials, and creates Authentication object.

. Now AuthenticationManager is invoked. It will pass the Authentication object to AuthenticationProvider to validate the credentials.

. AuthenticationProvider now attempts to authenticate the user by comparing the given credentials against the stored credentials in a database.

. AuthenticationProvider uses UserDetailsService object, to load the user’s details.

. If authentication successful, the Authentication object has along with credentials, it contains roles also. Now AuthenticationManager stores the fully loaded Authentication object, in the SecurityContext. This SecurityContext is managed by SecurityContextHolder.

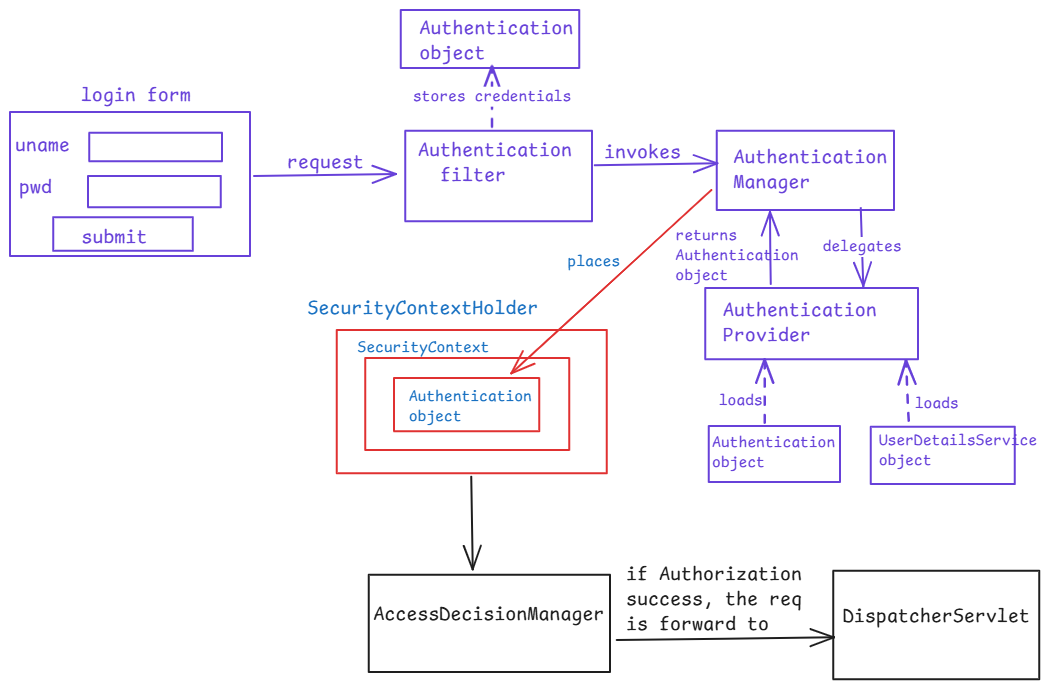
. After this authentication, AccessDecisionManager component is invoked for authorization.

. This AccessDecisionManager, will evaluate whether the user’s roles are permitted to access the resource or not.

. If authorization is successful, then the request reaches to the controller.

. If authorization fails, then ExceptionTranslationFilter will handle by redirecting the user to an error page or returns 403(Forbidden) response.

. If authentication fails, then also the same filter will handle by redirecting the user to an error page or returns 401(Unauthorized) response.



* create two tables in the database like below.

use test;

create table users (

username varchar(50) primary key,

password varchar(200) not null,

enabled boolean not null

);

create table authorities (

username varchar(50),

authority varchar(50),

constraint fk\_authorities\_users foreign key(username) references users(username)

);

Data preparation application github link

<https://github.com/ShekherJava/SB-Data-Preparation.git>

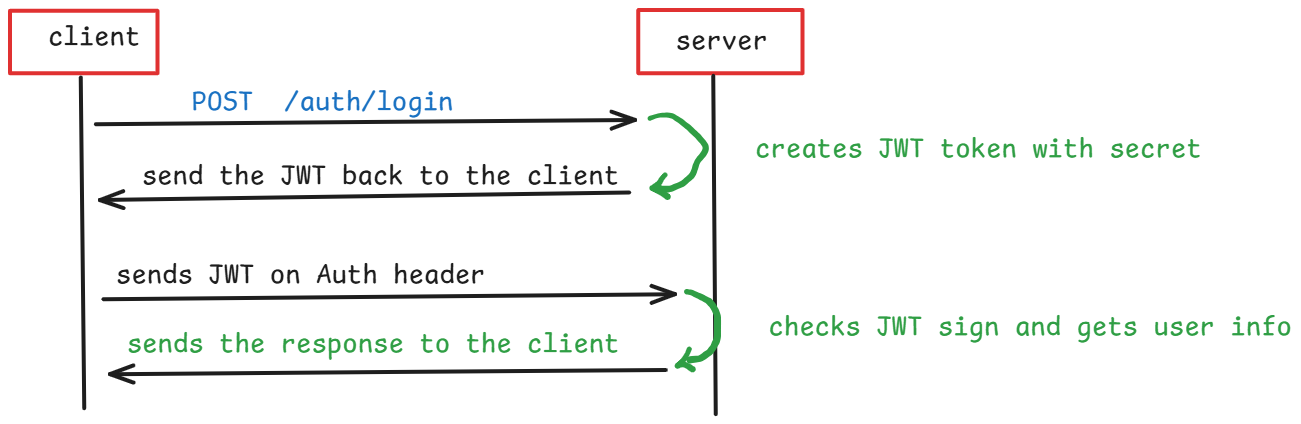
Rest API with Spring security configuration included

<https://github.com/ShekherJava/SB-EmpCURD-REST.git>

JWT(JSON Web Token):

Flow:

1. The client sends credentials to the server.
2. The server will verify the credentials and generates the JWT token.
3. The server will tranmit the token back to the client.
4. The client will store the token in a local storage.
5. The client will send the token in the subsequent requests, in Authorization header.
6. The server validates the token on each request. If it is valid, the request will process, otherwise request is rejected.



How can we compare JWT with Spring security?

* Spring security relies on server-side sessions. When a user logs in, a session is created for storing the session data on the server and every time when a user makes a request, the server will look up for the session and verifies who is the user.
* So, Spring security exhibits stateful behaviour.
* In JWT, when a user logs in, the server creates a token containing all the necessary information like user identity, roles, etc, signs it and then sends to the user. Every time when a user makes a request, the token is also sent in the header and the server verfies the token, without maintaining any sessions.
* So, JWT exhibits stateless behaviour.
* In Spring security, if you want to scale the application on to the multiple servers, you also need to share the sessions between the servers. So, scaling is not too easy.
* In JWT, if you want to scale the application then you can easily do it, because there is no need to share the sessions between servers.
* So, for single server environments, Spring security could be used and for distributed systems, JWT’s is a preferred way.

JWT token structure:

* The JWT token contains 3 parts, separated by dots(.)
* 1. header
* 2. payload
* 3. signature
* Header part contains the type of the token i.e., JWT and the algorithm used to generate the signature either HMACSHA256 or RSA.
* Payload part contains the user info like username, roles, email, etc.. and additional information like token creation time and expire time etc..
* Signature part ensures that the JWT token has not been tampared.

For example, header looks like,

{

“alg”: “HS256”,

“typ”: “jwt”

}

Then, this JSON is Base64Url encoded to form the first part of the JWT token.

For example, payload looks like,

{

“sub”: “123456789”,

“name”: “john”,

“admin”: true

}

Then, this JSON is Base64Url encoded to form the second part of the JWT token.

* To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.
* For example if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:
* HMACSHA256(

Base64UrlEncode(header)+ “.” + Base64UrlEncode(payload), secret

)

* The following shows a JWT that has the previous header and payload encoded, and it is signed with a secret.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c

what if token is hacked?

------------------------

* In JWT, the client has to send the token in the Authorization header with each request.
* In the middle, a hacker can hack/hijak the token.
* if hacker uses the same token, and sends a request then the server sends the response to the hacker’s request also.
* So, to prevent this hacker’s attack it is recommended to use JWT, in a secured layer i.e., https.
* If the hacker decodes the token, made changes to the payload, and sends the request with the new JWT token then what happens?
* When the server receives the JWT token in a request, first it will generate a signature with the header,payload from the token and with the secret key maintained in the server.
* Now, the server checks the signature in the jwt token sent in the request and the signature generated now, are matched or not.
* If doesn’t match then the server identifies it as an attack and rejects the request.
* When creating a Spring Boot with JWT application, we create the classes like JwtUtil, JwtFilter and SecurityConfiguration for the below reasons.
* The util class (often called JwtUtil or JwtTokenProvider) is responsible for handling JWT-related tasks like generating, validating, and parsing JWT tokens. Its purpose is to isolate token management logic, making the code more modular and reusable.
* **Token Generation**: Generates a JWT when a user successfully logs in, using their username and a secret key.
* **Token Validation**: Verifies if a token is valid, e.g., by checking its signature and expiration time.
* **Extracting Claims**: Retrieves claims (such as the username) from the token payload, which can be used for authentication.
* The filter class (often named JwtFilter or JwtAuthenticationFilter) intercepts incoming HTTP requests to check for a JWT in the Authorization header.
* This class verifies the token and, if valid, sets the user details in the SecurityContext, which Spring Security uses to handle authentication and authorization.
* **Extracts the JWT from Request Headers**: Parses the Authorization header to retrieve the JWT.
* **Token Validation**: Calls JwtUtil to check if the token is valid.
* **Setting Authentication Context**: If the token is valid, it loads the user’s details from the database and sets the SecurityContext with an Authentication object, effectively logging in the user for that request.
* The security configuration class (often called SecurityConfig) is used to define Spring Security’s behavior and to set up the application’s security parameters.
* This class configures what URLs are protected, how authentication is handled, and which filters to apply.
* **Enabling/Disabling Security Features**: Disables CSRF (since JWT provides sufficient security) and configures Spring Security’s session policy (typically set to stateless when using JWT).
* **Configuring URL Access**: Specifies which endpoints are publicly accessible (e.g., /login or /register) and which require authentication.
* **Adding JWT Filter**: Adds the JWT filter to the security filter chain so that the filter is applied to every incoming request, checking for valid JWTs before allowing access to secured resources.

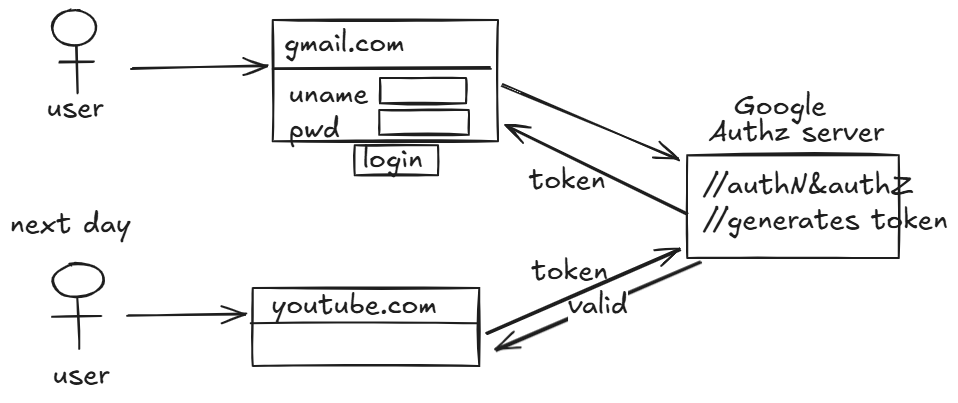
<https://github.com/ShekherJava/SB-Rest-JWT.git>

====================================================================

OAUTH2

How come Google allowing me to use all it’s products with the same account?

* To access any Google product/app, gmail account is mandatory.
* While creating a gmail account, the Authorization server of Google, it will read the user’s data and maintains in the Authorization server.
* Once account is created, for example, you are going to login into gmail.com website.
* when enter the credentials, the gmail.com application connects to Authorization server, and this server authenticates the users and generates a token, and the token is provided to the gmail.com application.
* This token gets also stored in the local storage of user’s browser.
* Now if you are trying to login to another Google product, say youtube.com, after a few days.
* Without providing the credentials, you are logged in into the youtube.com application. The reason is, the token is sent with request to youtube.com, and it connects with Authorization server to validate the token and if it is valid, the youtube.com application will allow the user to access the resources.
* This is all possible, because of OAUTH2 protocol.



* OAUTH stands of Open Authorization
* OAUTH2 is a security standard, it is a specification from Open Web Foundataion, which has defined a set of rules and guidelines to give one application, permission to access the data in another application.
* Oauth2 terminology:
  1. resource owner: The end user, who own the resources(data)
  2. client: It is a website/app, which interacts with Authz server, after taking the permission from the resource owner.
  3. Authz server: This is a server, in which the resource owner have an account and which has the authorization logic.
  4. resource server: This is a server which provides the required resources to the client, by hosting API’s.

For example,

As a user(resource owner), I have opened udemy.com, on my

browser.

udemy.com shows a button to login with Google, when a user

clicks on that button, it redirects the user to the Google

Authz server.

On successful authorization, a token is granted to the udemy

to access the resources.

With this token, udemy will access the user’s profile data

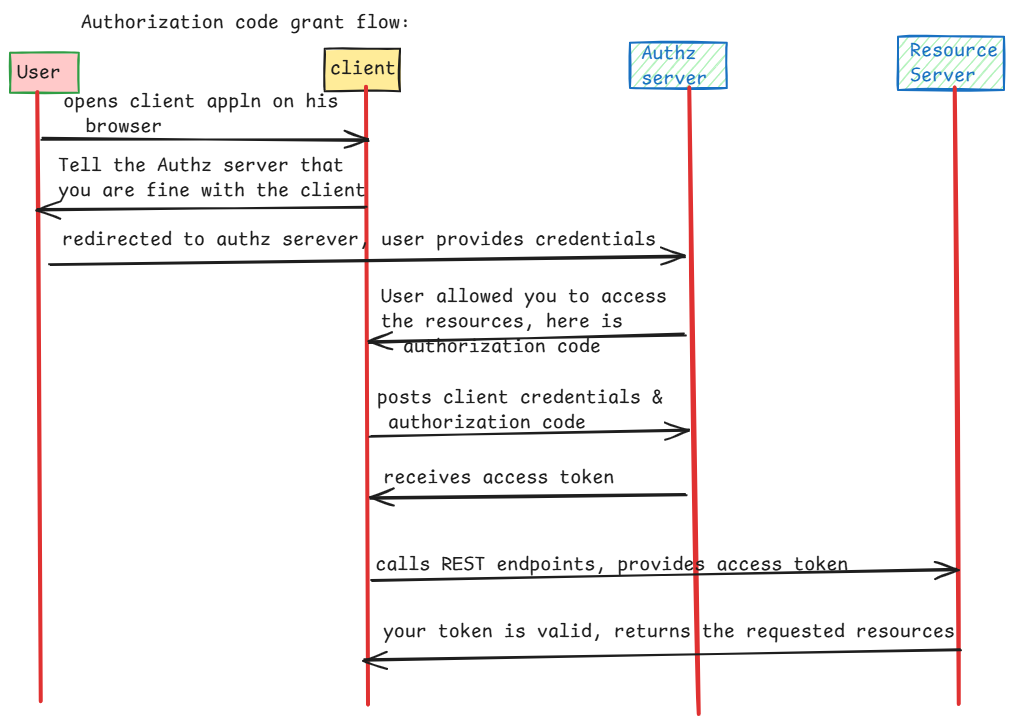
from the Google Resource server.

In this communication,

* 1. end user is the resource owner
  2. udemy.com is the client
  3. Google Authz server is the AuthZ server
  4. Google Resource server is the Resource server.
* Oauth2 framework has specified different grant types to allow the client to get access token, from Authz server.
* The two grant types that, we need to understand for implementing OAUTH2 in spring are,

. Authorization code grant type

. client credentials grant type



Note: Tell Authz server that you are fine with this client means,

user clicks on login with google button, on the client’s website.

<https://github.com/ShekherJava/SB-MVC-OAuth.git>

