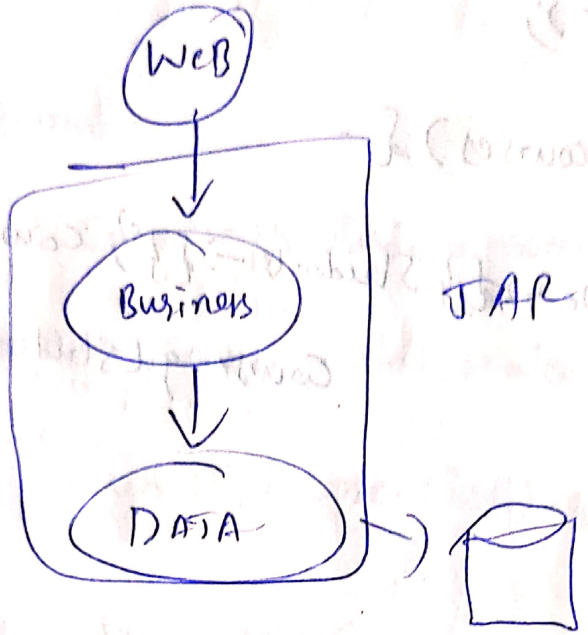
Web Services:

1. What is a webservice?

A service delivered over the Web.



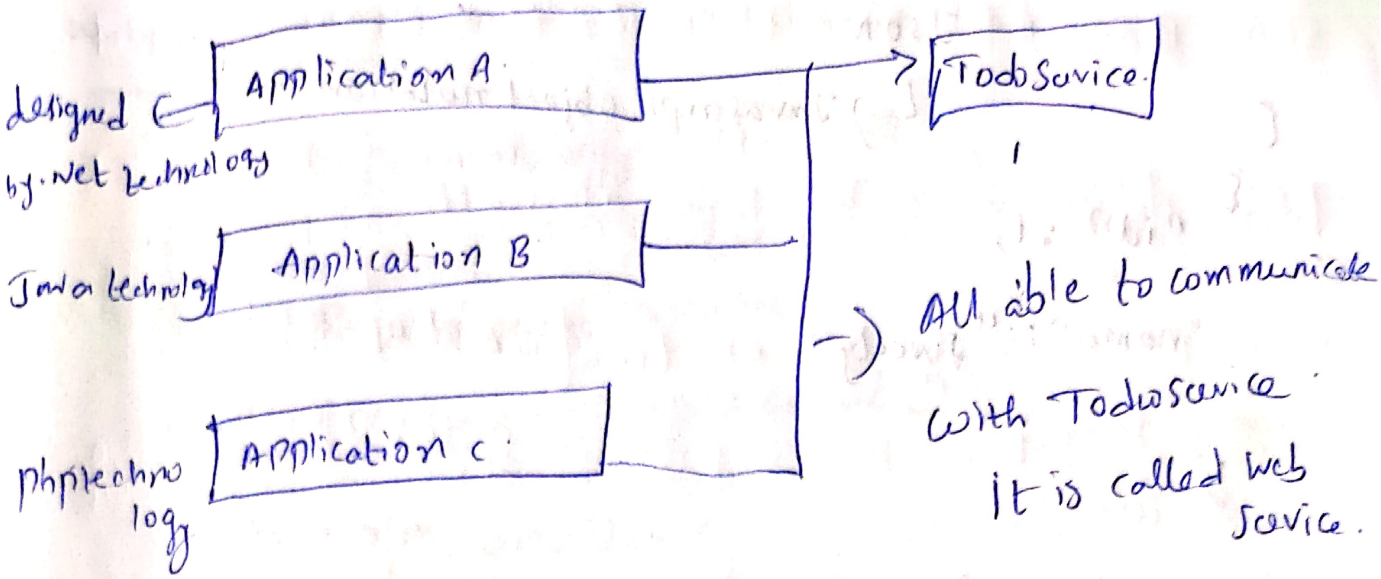
WEBSERVICE - W3C DEFINITION.

Software System designed to support interoperable machine-to-machine interaction over a network

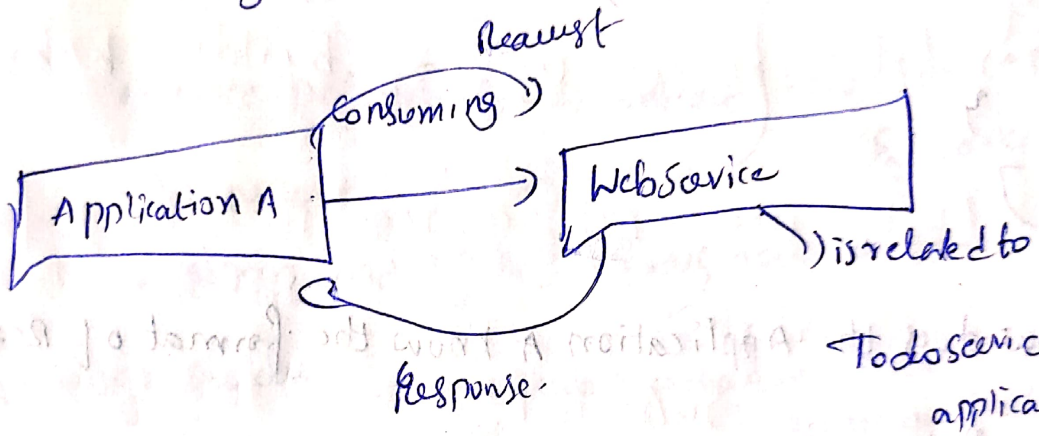
Webservices are designed for application to application interaction.

3 keys:-

- Designed for machine-to-machine (or application-to-application) interaction.
- Should be interoperable - not platform dependent.
- Allow communication over a network



How does data exchange b/w applications take place?



In

Webservice should be platform Dependent

→ when Request & response should be platform Independent

Two popular formats for request & response.

XML → eXtensible markup language.

XML.

```
<getCourseDetailsRequest>
    <id>Course1</id>
```

JSON.

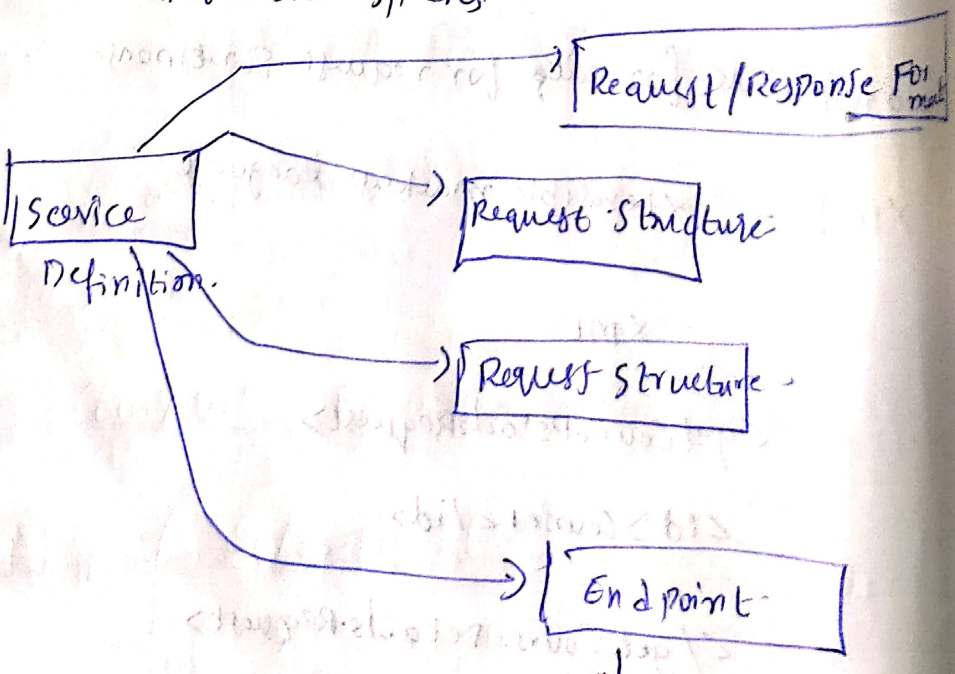
↳ Javascript object notation.

```
[
  {
    "id" : 1,
    "name" : "Tweety"
  },
  {
    "id" : 2,
    "name" : "madhu";
  }
]
```

How does the Application A know the format of Request or Response?

A: Every Webservice offers a service definition.

Service Definition specifies



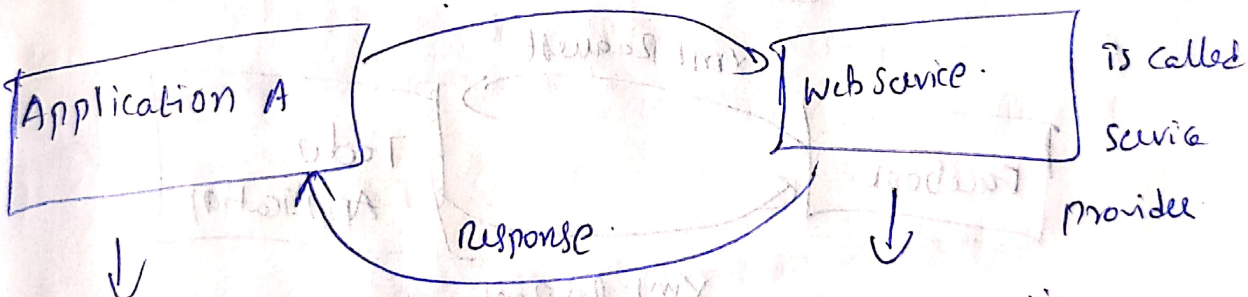
Web services - Key Terminology:

Key Terminology:

- Request and Response. (Input → output)
- Message Exchange Format → format of request and response
- XML and JSON.
- Service provider or server
- Service Consumer or client.
- Service Definition. → Contract } b/w service provider & service consumer
- Transport
- HTTP and MQ } Defines how a service is called.

Called by url or Queue

MQ → communication over a queue.



Consuming the Web Service application.

Introduction to SOAP web services:

Web services Groups

- SOAP-based.

SOAP & and REST are not really Comparable.

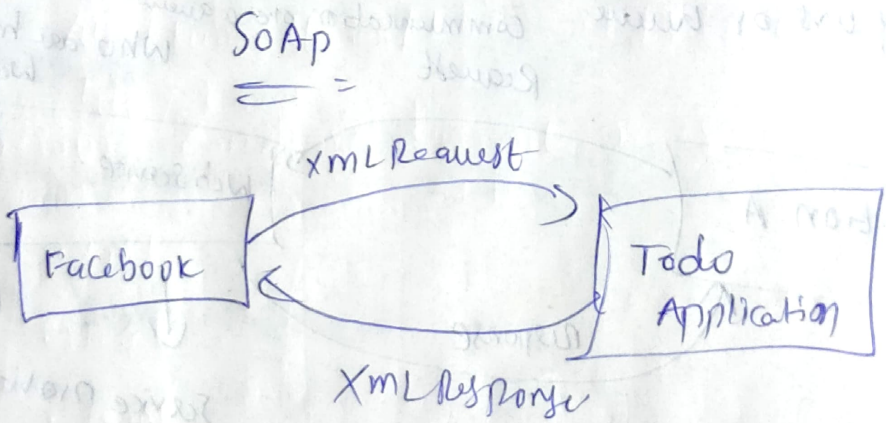
REST defines an architectural approach

SOAP possess restriction on the format of xml which is exchanged b/w service provider & service Consumer

SOAP → simple object Access protocol

↳ Define specific way of building web services

In SOAP we use XML as request exchange format

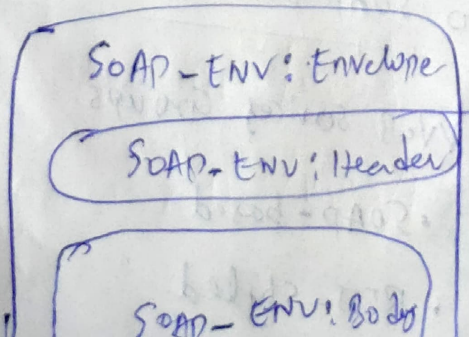


<getCourseDetailsRequest>

<id> Course1 </id>

</getCourseDetailsRequest>

To Create SOAP envelope we need to create



Header contains meta information like authentication and authorization

Body: We can Put Content of Request or response

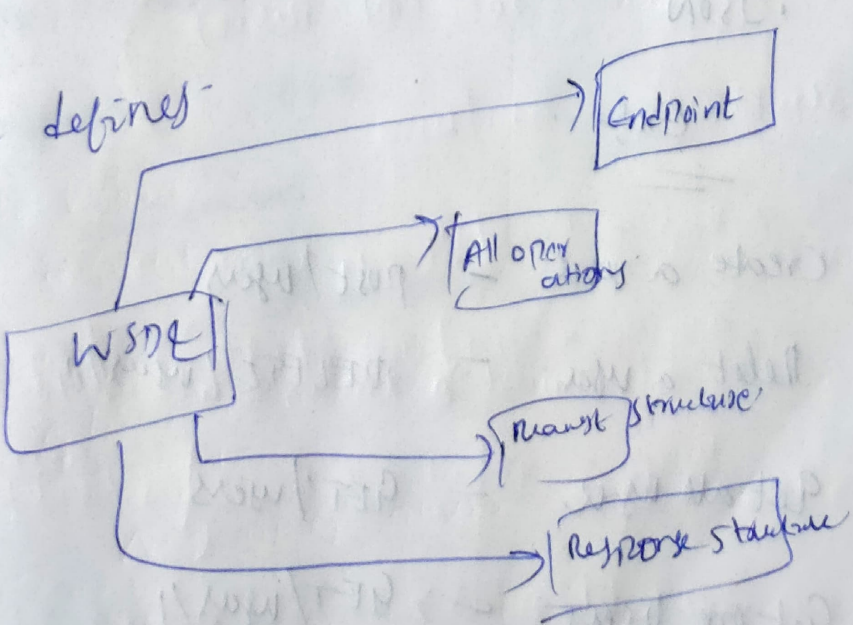
SOAP Header is optional.

SOAP:

- Format:
  - SOAP XML Request
  - SOAP XML Response
- Transport:
  - SOAP over MQ
  - SOAP over HTTP
- Service Definition
  - WSDL

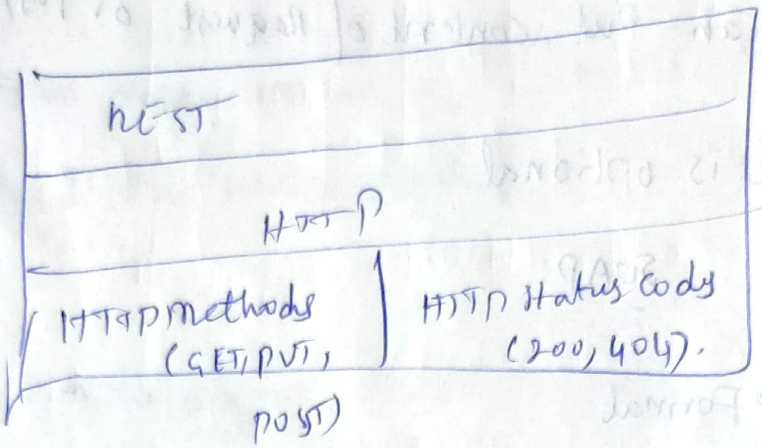
    Web Service Definition Language

WSDL defines



RESTFUL Webservices

REST stands for Representational State Transfer



Key ABSTRACTION – RESOURCE

- A resource has an URI (Uniform Resource Identifier)
  /users/kumar/todos/1
  /users/kumar
- A resource can have different representations.
  - XML
  - HTML
  - JSON

Eg:-

Create a user – POST /users

Delet a user – DELETE /users/1

Get all users – GET /users

Get one users – GET /users/1

REST:
===

make use of HTTP methods and

• Data Exchange format

• No restriction • JSON is popular

• Transport

• only HTTP

• Service Definition

• No Standard • WADL / Swagger

REST vs SOAP → WSDL

NO WSDL ∈

• Restrictions. vs Architural approach.

• Data Exchange Format.

• Service definition

• Transport Both.

HTTP

• Ease of implement

Better implement than SOAP.

Based on Jason

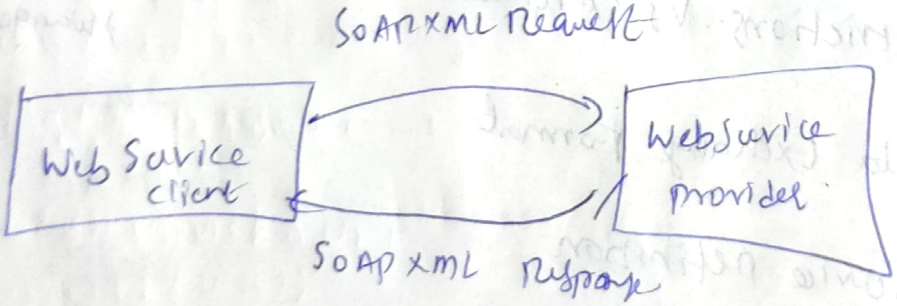
Web service Definite language

Swagger

SOAP Web Services

- Service definition
- XML
- XSD (XML schema definition)
- WSDL
- XML-Java Binding
- JAXB → Java API for XML Binding
- Framework Configuration
- Endpoint
- WSDL - Contract first
- SOAP Web Service client
- Wizdler chrome plugin

We need to create



XML structure:-

namespace will make xml unique

<GetCourseDetailsRequest xmlns="http://example.com/courses">

<id>123</id>

</GetCourseDetailsRequest>

↓ Name Space

<GetCourseDetailsResponse xmlns="https://example.com/Courses">

<courseDetails>

<id>123</id>

<name>Spring in 28 minutes</name>

<description>you would learn the base</description>

</courseDetails>

</GetCourseDetailsResponse>

XSD to validate file or not.

Create xmlschema file.

↳ to define how xml should look like

xmlns:tns= TargetNamespace used for which xml file it created

<element name="GetCourseDetailsRequest">

<complexType>

<sequence>

<element name="id" type="integer"></element>

</sequence>

</complexType>

</element>

<!-- --> Comments

<complexType> ↳ inside we can define sequence

Java API for xml Binding: JAXB

XSD Source folder

Jaxb2

RESTful Web Services:

Http methods

web, devtools, JPA, h2

REST Representation State Transfer

Rest is a style of S/W architecture for distributed hypermedia systems.

Make use of HTTP:

Key abstraction is resource.

Each resource has URI

Retrieve all users - GET /users

Create a user - POST /users

Retrieve one user - GET /users/{id} -> /user/1

Delete a user - DELETE /users/{id} -> /users/1

Retrieve all posts for a user - GET /users/{id}/posts

create a posts for a user - POST /user/{id}/posts

Retrieve details of a post - GET /~~posts~~users/{id}/post/{post_id}

---

what is dispatcherservlet?

---

logging.org.springframework = debug

```
@RestController
Public class HelloworldController {

@GetMapping(Path = "/helloWorld")
Public HelloBean helloBean()
{ return new HelloBean("hello World");
}

@GetMapping(Path = "/helloworld/{name}")
Public HelloBean hellopathVariable(@PathVariable String name)
{return new HelloBean(String.format("Hello %s", name));

}
```

Creating user Bean and user service

```
@Component
Public class UserDaoService {
  Private static int count=3;
Private Static List<User> users = new ArrayList<>();

  Static {
    users.add(new User(1, "kumar", new Date()));
    users.add(new User(2, "Sweety", new Date()));
    ules.add(new Usa(3, "madhu", new Date()));
  }

  Public List<Usa> findALL() {
    return users;

Public Usa Save(Usa user) {
  if (user.getId() == null) {
    usa.setId(++userCount);
  }
  users.add(user);

  return usa
}
Public User findone(int id) {
  for(Usa user: users) {
    if(user.getId()==id) {
```

spring.jackson.serialization.write-dates-as-timestamps=false

↓ Reading dates as dates not timestamps

```
Public class UserResource {

@Autowired
Private UserDaoService service;

  @GetMapping("/users/{id}")
  Public User retrieveUser(@PathVariable int id) {

    return service.findone(id);

  }

@PostMapping("/add")
ResponseEntity<Object>
Public void create User(@RequestBody User user)

{
User saved=Servica.save(user);

}
```

// How to return the status of created user back.

// how to set the URI of the created resource in to response.

```
URI location = ServletUriComponentsBuilder
        .fromCurrentRequest()
        .path("/{id}")
        .buildAndExpand(savedUser.getId()).toUri();

return ResponseEntity.created(location).build();
```

→ return current request URI (← appending id and)

```
Public class Resource
{
@GetMapping("/users/{id}")
Public User retrieveUser(@PathVariable int id)
    User usa = savice.findone(id);
    if(usa==null)
        throw new UserNotFoundException("id-"+id);
    return usa;
    }
@ResponseStatus(HttpStatus.Not-FOUND)
PublicClass UserNotFoundException extends RuntimeException{
{
    Public UsaNotFoundException(String message)
    { Super(message);
    }
}
Public Usa deleteById(int id) {
Iterator<Usa> iterator = users.iterator();
  while(iterator.hasNext()) {
  Usa usa = iterator.next();
if(usa.getId()==id) {
```

Validations @Valid.

@Valid, in post method

@Size(min=2)
private String name

@Past → only past is valid not future date

private Date birthDate;

Implementing HATEOAS for RESTful Web services.

Hypermedia As the Engine Of Application State.

H A T E O A S

SpringBoot hateoas starter enables us easy to add link using the method.

```
public class UserResource {

@GetMapping("/users/{id}")
Public ResoureEntity<User> retriveUser(@PathVariable int id)

{
  User user = service.findOne(id);

  if(user==null)
     throw new UserNotFoundException("id:"+id);

  ResourceEntity<User> resource = new ResourceEntity<User>(user); // creating
                                                                  resource for
```

ControllerLinkBuilder linkTo = linkTo(methodOn(this.get

↓ class().retrieveAllUsers());

This link indicates that we are creating link to methods.

resource.add(linkTo.withRel("all-users"));

↓ ↓ name refer to it

adding link to resource.

return resource;

}

Instead of returning data we are returning links and data.

@SpringBootApplication.

Public class RestfulWebServicesApplication {.

Public LocaleResolver localeResolver() {.

AcceptHeader ~~Session~~LocaleResolver localResolver = new ~~Session~~Locale

Resolver();

localResolver.setDefaultLocale(Locale.US);

return localeResolver;

}

Accept Header

src/main/resource -> Create messages.

message.properties.

good.morning.message = Good Morning

```
rm( ) { in @SpringBoot application
@Bean
Public ResourceBundleMessageSource bundleMessageSource() {
ResourceBundleMessageSource messageSource = new Resource
                    BundleMessageSource();
messageSource.SetBasename("messages");
return messageSource;
}
}

Public class HelloWorldController {
@Autowired
private MessageSource messageSource;

@GetMapping(path = "/hello-World-internationlized")
Public String helloWordInternationalized(@RequestHeader
                    (name = "Accept-Language", required
                    =false) Locale locale);
return messageSource.getMessage("good.morning.message",
                    null, locale);
}
}  Instead of (@RequestHeader(name = ("Accept-
                    , locale)
```

use

return messageSource.getMessage("good.morning.message", null, LocaleContextHolder.getLocale());

Content negotiation:

Content negotiation is the data formatted either in application/xml or application/Jason

Swagger:-

It is a documentation format for restful services.

```
@EnableSwagger2
@Configuration
Public class SwaggerConfig {

    @Bean
    Public Docket api() {

        return new Docket(DocumentionType.SWAGGER_2)
            .apiInfo(DEFAULT_API_INFO);
```

localhost:8080/v1/api-docs

localhost:8080/swagger-ui.html

```
@Configuration

@EnableSwagger2

public class SwaggerConfig {

public static final Contact DEFAULT_CONTACT = new Contact

    ("Ranga", "http://example.com", "example@gmail.com");

public static final ApiInfo DEFAULT_API_INFO = new Api

        Info("API Title", "API Documentation",

        "1.0", "urn:tos", DEFAULT_CONTACT,

        "Apache 2.0", "http://www.apache.org/licenses

                         /LICENSE-2.0");

Private static final Set<String> DEFAULT_PRODUCES_AND_CONSUMES
                 = new HashSet<String>("application/json",
@Bean                                   "application/xml");

public Docket api() {

    return new Docket(DocumentationType.SWAGGER_2)

            .apiInfo(DEFAULT_API_INFO)

            .produces(DEFAULT_PRODUCES_AND_CONSUMES);
  }
            .consumes(DEFAULT_PRODUCES_AND_CONSUMES);
}
```

@ApiModel(description = "All details about the user")
class User {

@ApiModelProperty(notes = "Name should have atleast 2 characters")
Private String name;

@Past
@ApiModelProperty(notes = "Birth date should be in the Past").
Private Date birthDate;

}

localhost:8080/actuator

management.endpoints.web.exposure.include=*

enabling web exposure.

Implementing static Filtering:

Class User {
@JsonIgnore → Ignoring this field.
Prive Date ~~dateOfBirth~~Date;

or

@JsonIgnoreProperties(value={"field1","field2"})
Class User {

Dynamic filtering:

```
Public class FilteringController {

@GetMapping("/filter")

 Public SomeBean someBean() {

SomeBean someBean = new SomeBean("value1", "value2", "value3");

SimpleBeanPropertyFilter filter = SimpleBeanPropertyFilter.filterOut
                                  AllExcept("field1", field2");

FilterProvider filter = new SimpleFilterProvider().addFilter
                        ("SomeBeanFilter", filter);

MappingJacksonValue mapping = new MappingJacksonValue(Some
                                                       Bean);

 mapping.setFilters(filters);

 return mapping;

}

@JsonFilter("SomeBeanFilter")
Class SomeBean {

}
```