11

Implementing a Multi-tier Design with RESTful Web Services

ORACLE

Objectives

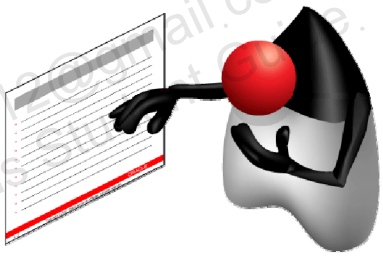
After completing this lesson, you should be able to:

- Compare the HenleyApp two-tier design and the HenleyApp three-tier design
- Describe a RESTful web service
- List the web services used in the HenleyApp application
- Describe how the RESTful web services were developed in the HenleyServer application

In this lesson, you will get a brief overview of Java web services with focus on RESTful web services. This lesson will not provide you with the details about how the web services technology works. The intent is to provide you with the big picture about the HenleyServer and BrokerToolServer applications without going into the implementation details.

Topics

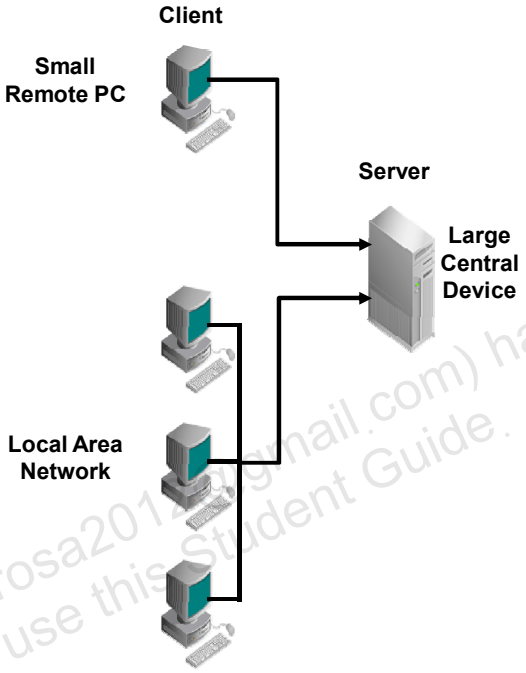
- Three-tier design versus two-tier design
- JAX-RS web services
- JAX-RS web services in the HenleyServer application



Two-Tier Architecture

Two-tier client/server architectures have two essential components:

- A client application
- A database server



Two-Tier Considerations

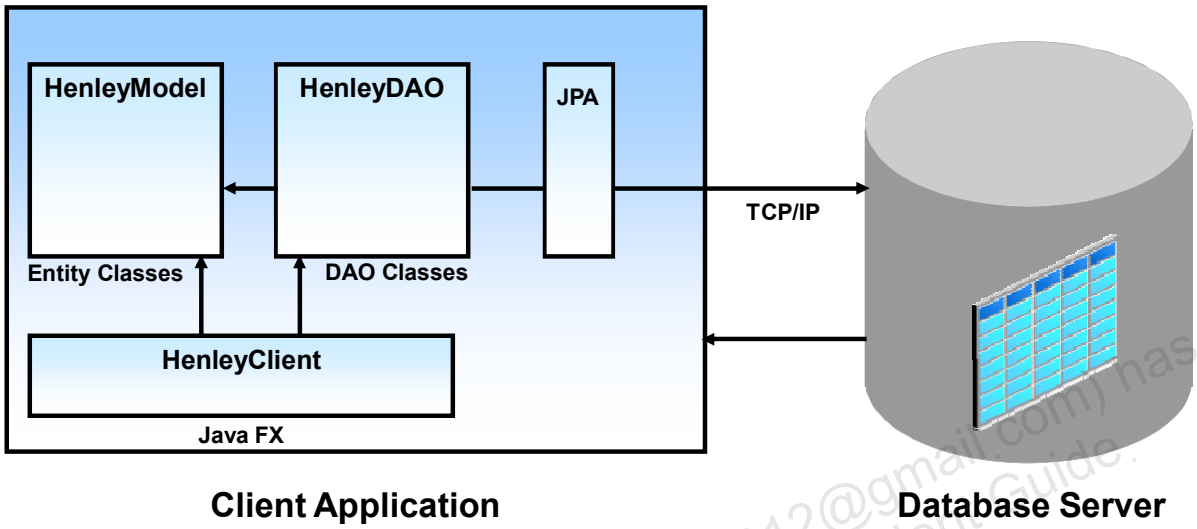
Client application accesses database directly.

- Requires a code change to port to a different database
- Potential bottleneck for data requests
- High volume of traffic due to data shipping

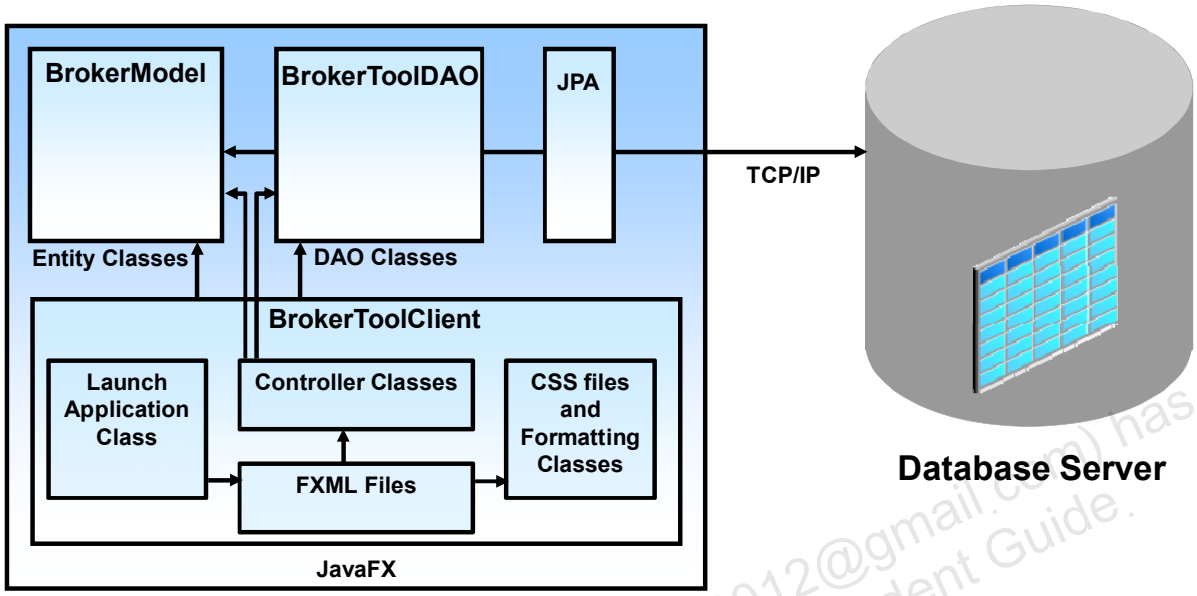
Client application executes application logic.

- Limited by processing capability of client workstation (memory, CPU)
- Requires application code to be distributed to each client workstation

HenleyApp Two-Tier Design



BrokerTool Two-Tier Design



Client Application

The client application as shown in the slide comprises BrokerToolClient, BrokerModel, BrokerToolDAO, and JPA components.

- **BrokerToolClient:** This component contains FXML files representing the front-end screens, corresponding controller classes that perform event handling, a class to launch the BrokerTool application, and CSS files, along with a few formatting classes that are used to format the front-end screens.
- **BrokerModel:** This component contains the entity classes that represent the tables of the BrokerTool database.
- **BrokerToolDAO:** This component contains the classes that perform the database operations using the BrokerModel classes.
- **JPA:** This component is the JPA 2.0 library that is used to connect to the database and perform database operations smoothly.

Database Server

Java DB is the database server that is used to store the tables of the BrokerTool database.

Advantages of Two-Tier Design

A two-tier design:

- Is more extensible than a one-tier design
- Combines presentation logic, business logic, and data resources into a single system
- Can have a client on any host as long as that host is connected by a network to the data-resource tier, unlike a one-tier design
- Has fewer points of failure than a three-tier design

Disadvantages of a Two-Tier Design

A two-tier design has the following disadvantages:

- Any changes to the business strategy result in changes in the business logic, which requires redeployment of all clients. This can be very costly and time-consuming.
- Each client requires a connection to the data resource.
- This design restricts or complicates the addition of mirroring, caching, proxy services, or secure transactions to the data resource.
- The business logic is in the client tier. Therefore, all the database data that is used by the application is exposed on the network.

Three-Tier Architecture

A three-tier client-server architecture has the following components:

- A client application
- An application server
- A database server

Three-Tier Architecture Considerations

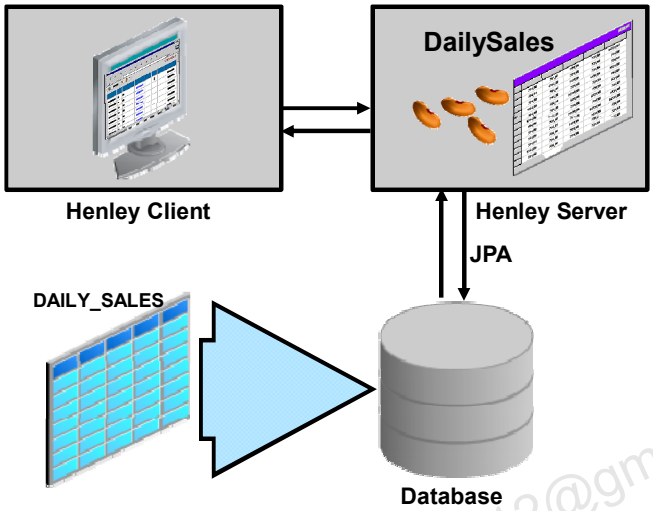
The client application contains presentation logic only; therefore:

- Less resources are needed for client workstation.
- No client modification is needed if the database location changes.
- There is less code to distribute to client workstations.

One server handles many client requests; therefore:

- More resources are available for the server application.
- Data traffic on the network is reduced.

HenleyApp Three-Tier Design



Extending HenleyApp to Three-Tier

- Develop a HenleyServer application as a web application (WAR file).
  - It will contain the business logic to perform CRUD operations with the HenleyApp database on the Java DB server.
  - Its components will be published as web services.
  - The web services will be developed using Jersey, which is a reference implementation of JAX-RS.
  - It will be deployed on the GlassFish application server and can be consumed over HTTP.

The list in this slide and the next provides an overview of how to extend the HenleyApp application to a three-tier application.

Create a HenleyApp Server application and deploy it on a GlassFish application server. The server application is a collection of RESTful web services that are available over HTTP. The web services use JPA to talk to the HenleyApp database stored on a Java DB database server.

GlassFish will provide infrastructure services for both JPA and JAX-RS.

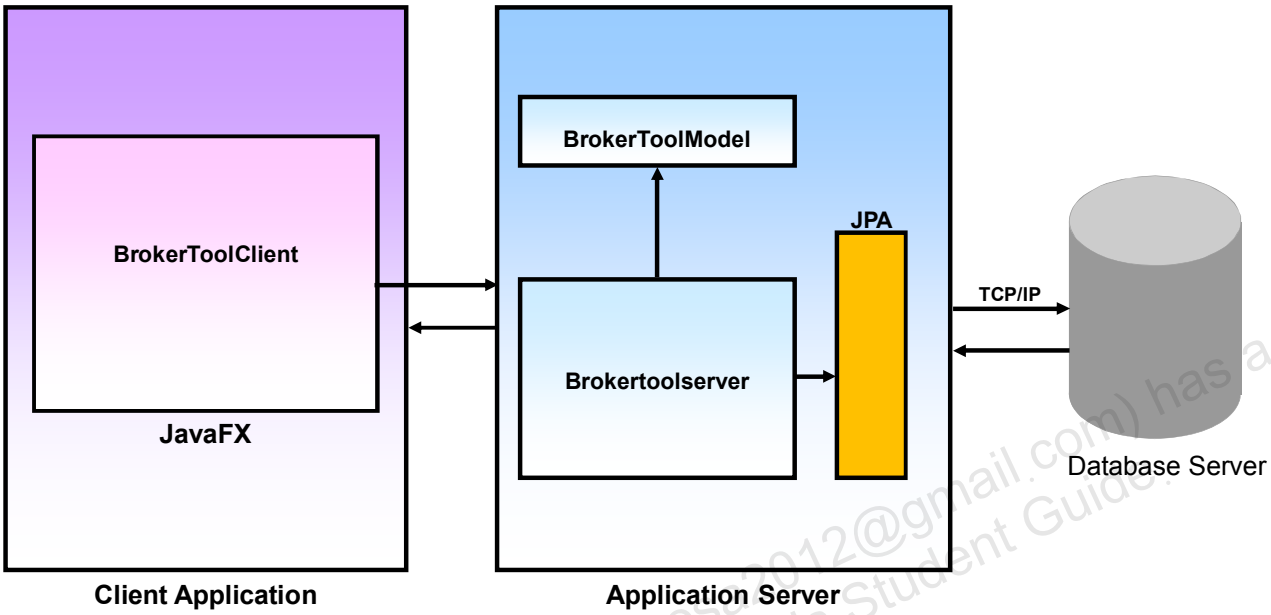
If the Java DB server is deployed in a separate address space, the three-tier architecture can be extended to a multi-tier one.

Subsequent slides discuss why JAX-RS was chosen as the web services API.

Extending HenleyApp to Three-Tier

- Separate the model and the view component into two tiers.
  - `HenleyClient` will comprise the front-end user interface and controller component developed in JavaFX.
  - `HenleyModel` will contain the Entity classes that will be used by JPA for database persistence.
  - `HenleyModel` classes will be used by HenleyServer. Therefore, HenleyModel's JAR will be added to the HenleyServer library.
- Modify the JavaFX front-end components of HenleyClient to consume the web services published by HenleyServer.

BrokerTool Three-Tier Design



Client Application

The client application as shown in the slide comprises BrokerToolClient. The BrokerToolClient is the front end of the BrokerTool application. It primarily comprises FXML files and their corresponding controller classes.

Application Server

The Application Server contains the BrokerToolModel, BrokertoolServer, and JPA components that represent the business logic of the BrokerTool application.

- **BrokerToolModel:** This contains the entity classes class that represent the tables of the BrokerTool database.
- **BrokertoolServer:** This component's classes provide server capability and use the BrokerToolModel classes.
- **JPA:** This component is the JPA 2.0 library that is used by the BrokertoolServer to connect to the database and perform database operations smoothly.

Database Server

The enterprise storage system is placed on the database server. Java DB is the database server that is used to store the tables of the BrokerTool database.

Advantages of Three-Tier Design

A three-tier design has the following advantages:

- It enables efficient use of data resource connections by using connection pooling.
- It is possible to change business logic without redeploying client software.
- It is architecturally better suited for scaling and load balancing than other designs.
- Scaling affects primarily the middle tier.

Disadvantages of Three-Tier Design

A three-tier design has the following disadvantages:

- This design has increased network traffic.
- This design has multiple points of failure.
- Business objects must be designed to manage transactional integrity.

Quiz

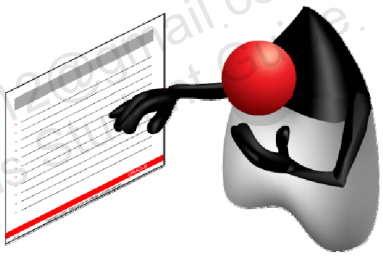
Select the advantages of three-tier design from the following list:

a. The business logic is in the client tier. Therefore, all the database data that is used by the application is exposed to the network.
b. It is possible to change business logic without redeploying client software.
c. It is architecturally better suited for scaling and load balancing than other designs.
d. It combines presentation logic, business logic, and data resources into a single system

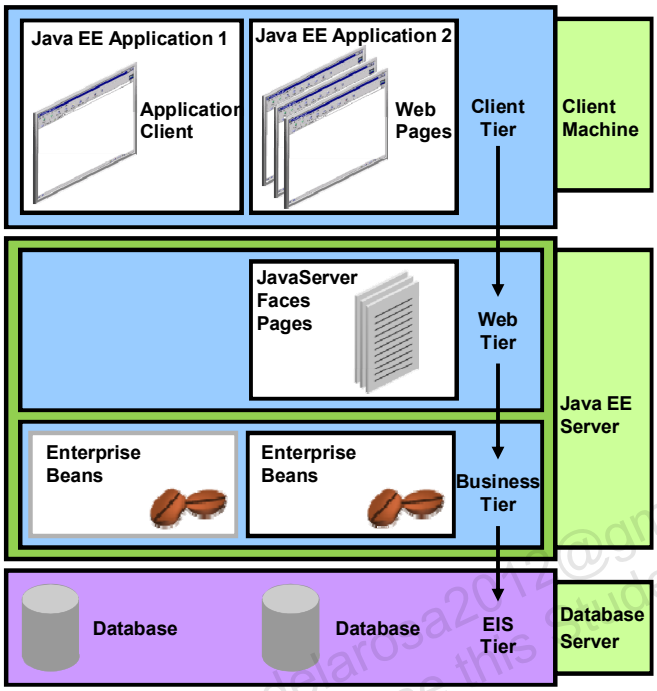
Answer: b, c

Topics

- Three-tier design versus two-tier design
- JAX-RS web services
- JAX-RS web services in the HenleyServer application



Implementing Three-Tier Design



Java EE Technologies Used to Implement Three-Tier Design

- **Java EE Web Components**
- **Java EE – EJB components**
- **Java EE web services**
- **Java EE server:** The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.
- **Enterprise JavaBeans (EJB) container:** Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.
- **Web container:** Manages the execution of web pages, servlets, and some EJB components for Java EE applications. Web components and their container run on the Java EE server.
- **Application client container:** Manages the execution of application client components. Application clients and their container run on the client.
- **Applet container:** Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.
- **EE EJB** is also designed for such things, but it can be a pretty heavy architecture for a simple client-server app.

Web Services

Web services:

- Are applications that communicate over the World Wide Web's (WWW) HyperText Transfer Protocol (HTTP)
- Provide a standard means of interoperating between software applications running on a variety of platforms and frameworks
- Are interoperable and extensible because of using XML
- Can be combined in a loosely coupled way to achieve complex operations

A web service is a software component provided through a network-accessible endpoint.

By using web services, your client server two-tier application can be modified to a three-tier networked application that can operate over the web. Thus your three-tier application will become extensible as the use cases expand and it will become interoperable with different kinds of client applications.

Types of Web Services

The two types of web services are Simple Object Access Protocol (SOAP)-based web services and Representational State Transfer (REST)ful web services:

JAX-WS	**API for SOAP-based web services**
JAX-RS	**API for RESTful web services**

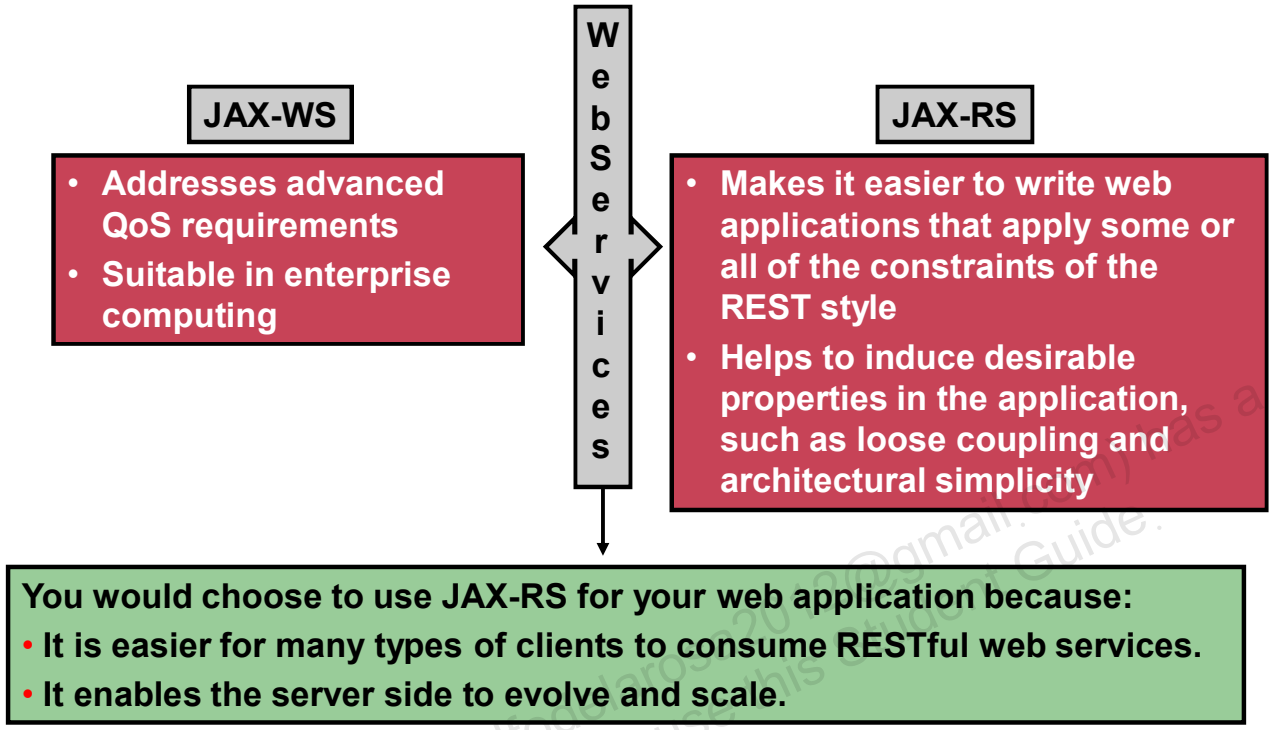
SOAP based web services use XML messages that follow the Simple Object Access Protocol (SOAP) standard, an XML language defining a message architecture and message formats. Such systems often contain a machine-readable description of the operations offered by the service, written in the Web Services Description Language (WSDL), an XML language for defining interfaces syntactically.

REST is well suited for basic, ad hoc integration scenarios. RESTful web services, often better integrated with HTTP than SOAP-based services are, do not require XML messages or WSDL service API definitions.

RESTful web services are based on the JSR-311 specification, JAX-RS API. Jersey is a reference implementation of JAX-RS .

RESTful web services use existing well-known W3C and Internet Engineering Task Force (IETF) standards (HTTP, XML, URI, MIME) and have a lightweight infrastructure that allows services to be built with minimal tooling. Developing RESTful web services is inexpensive and, therefore, has a very low barrier for adoption. You can use a development tool such as NetBeans IDE to further reduce the complexity of developing RESTful web services.

Which Type of Web Service to Use?



You would want to use RESTful web services for integration over the web and use SOAP-based web services in enterprise application integration scenarios that have advanced quality of service (QoS) requirements.

When compared to JAX-RS, JAX-WS makes it easier to support the WS-* set of protocols, which provide standards for security and reliability, among other things, and interoperate with other WS-*–conforming clients and servers.

You would choose to use JAX-RS for your web application because it is easier for many types of clients to consume RESTful web services while enabling the server side to evolve and scale. Clients can choose to consume some or all aspects of the service and mash it up with other web-based services, because it is easy to modify the server without breaking existing clients.

When to Use REST

A RESTful design may be appropriate when:

1. The web services are completely stateless.
2. A caching infrastructure can be leveraged for performance.
3. The service producer and service consumer have a mutual understanding of the context and content being passed along.
4. Bandwidth is particularly important, and limited.
5. Web service delivery or aggregation into existing websites can be enabled easily with a RESTful style.

This slide provides an analysis for using REST with design point of view. Developers must decide when this particular style is an appropriate choice for their applications. Each point in the slide is elaborated below:

1. A good test to check whether the web service is stateless is to consider whether the interaction can survive a restart of the server.
2. If the data that the web service returns is not dynamically generated and can be cached, then the caching infrastructure that web servers and other intermediaries inherently provide can be leveraged to improve performance. However, the developer must take care because such caches are limited to the HTTP GET method for most servers.
3. Because there is no formal way to describe the web services interface, both parties must agree on the schemas that describe the data being exchanged and on ways to process it meaningfully.
4. REST is particularly useful for devices such as PDAs and mobile phones, for which the overhead of headers and additional layers of SOAP elements on the XML payload must be restricted.
5. Rather than starting from scratch, services can be exposed with XML and consumed by HTML pages without significantly refactoring the existing website architecture.

Principles of a RESTful Web Service

RESTful applications are simple, lightweight, and fast because:

- **Resources** are identified by URIs, which provide a global addressing space.
- **A uniform interface** is used for resource manipulation.
- **Self-descriptive messages** or metadata about the resource is available and used.
- **Stateful interactions through hyperlinks** are based on the concept of explicit state transfer.

ORACLE

REST is a key design idiom that represents a stateless client-server architecture. A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

Resources are manipulated by a fixed set of four create, read, update, and delete operations: PUT, GET, POST, and DELETE.

Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, and JSON. Metadata about the resource is available and used, for example, to control caching, detect transmission errors, negotiate the appropriate representation format, and perform authentication or access control.

Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.

Web service clients that want to use these resources access a particular representation by transferring application content using a small, globally defined set of remote methods that describe the action to be performed on the resource.

Relationships Between SQL and HTTP Verbs

For resource manipulation, an analogy to operations in SQL is made, which also relies on a few common verbs, as shown in this table:

Action	SQL	HTTP
Create	Insert	PUT
Read	Select	GET
Update	Update	POST
Delete	Delete	DELETE

ORACLE

With RESTful web services, there is a natural mapping between the HTTP methods and most CRUD-like business operations that many services expose. Though there are no hard-and-fast rules, the following general guidelines are applicable for most cases:

- GET is used to retrieve data or to perform a query on a resource. The data returned from the web service is a representation of the requested resource.
- PUT is used to create a new resource. The web service may respond with data or a status indicating success or failure.
- POST is used to update existing resources or data.
- DELETE is used to remove a resource or data.

In some cases, the update and delete actions may be performed with POST operations as well (for example, when the services are consumed by browsers that do not support PUT or DELETE).

Sometimes, the following mapping might be applicable for PUT and POST:

- Create = PUT if you are sending the full content of the specified resource (URL)
- Create = POST if you are sending a command to the server to create a subordinate of the specified resource, using some server-side algorithm
- Update = PUT if you are updating the full content of the specified resource
- Update = POST if you are requesting the server to update one or more subordinates of the specified resource

Developing a RESTful Web Service with JAX-RS

- JAX-RS, a Java programming language API:
  - Is designed to make it easy to develop applications that use the REST architecture
  - Uses Java programming language annotations to simplify the development of RESTful web services
  - Uses annotations that are runtime annotations, and therefore, runtime reflection will generate the helper classes and artifacts for the resource
- In the Java programming language class files, you use JAX-RS annotations to define resources and the actions that can be performed on those resources.
- Project Jersey is the reference implementation for the JAX-RS specification.

Jersey implements support for the annotations defined in the JAX-RS specification, making it easy for developers to build RESTful web services with Java and the Java Virtual Machine (JVM).

A Java EE application archive containing JAX-RS resource classes will have the resources configured, the helper classes and artifacts generated, and the resource exposed to clients by deploying the archive to a Java EE server.

Detailed information about RESTful web services can be obtained from http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html.

RESTful Web Service with JAX-RS: An Example

```
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;
//The Java class will be hosted at the URI path
//"/hello"
@Path("/hello")
 public class Hello {
 // The Java method will process HTTP GET requests
 @GET
// The Java method will produce content identified by the
//MIME Media type "text/plain"
 @Produces("text/plain")
public String sayHello() {
// Return some textual content
return "Hello World";
 }}
```


The code sample in the slide is a very simple example of a root resource class that uses JAX-RS annotations.

The annotations used in the example can be explained as follows:

- The @Path annotation's value is a relative URI path. The Java class will be hosted at the URI path /helloworld.
  - This is an extremely simple use of the @Path annotation, with a static URI path.
  - Variables can be embedded in the URIs.
  - *URI path templates* are URIs with variables embedded within the URI syntax.
- The @GET annotation is a request method designator, along with @POST, @PUT, @DELETE, and @HEAD, defined by JAX-RS and corresponding to the similarly named HTTP methods. In the example, the annotated Java method will process HTTP GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
- The @Produces annotation is used to specify the MIME media types that a resource can produce and send back to the client. In this example, the Java method will produce representations identified by the MIME media type "text/plain".

- The @Consumes annotation is used to specify the MIME media types that a resource can consume that were sent by the client.
  - The example could be modified to set the message returned by the sayHello method, as shown in this code example:

```
@POST @Consumes("text/plain")
      public void sendMessage(String message)
      { // Store the message
       }
```

Quiz

What is a web service?

a. An application that communicates over the World Wide Web's (WWW) Hyper Text Transfer Protocol (HTTP)
b. An application that provides a standard means of interoperating between software applications running on a variety of platforms and frameworks
c. An application that is interoperable and extensible because of using XML
d. A software component provided in the client tier

Answer: a, b, c

Quiz

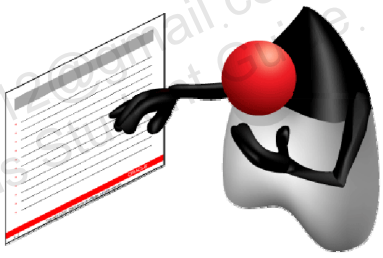
Identify the statements that are true about RESTful web services:

a. REST is a key design idiom that represents a stateful client-server architecture.
b. A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients.
c. Resources are defined as tables that provide a global addressing space for resource and service discovery.
d. Resources are manipulated by a fixed set of four create, read, update, and delete operations: PUT, GET, POST, and DELETE.

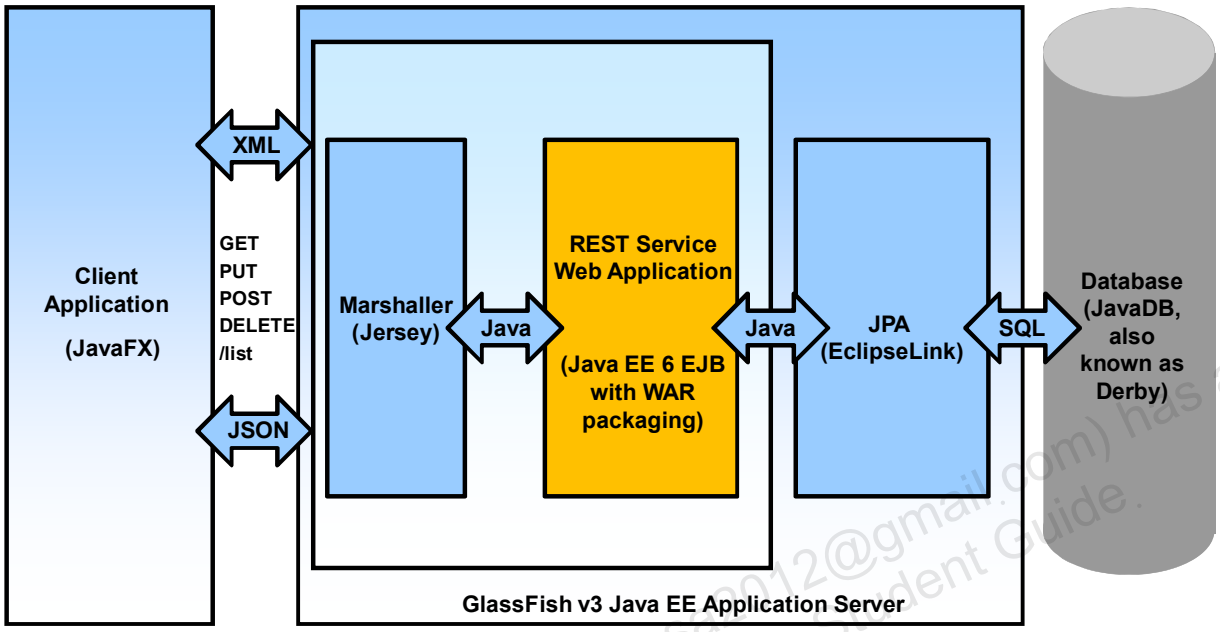
Answer: b, d

Topics

- Three-tier design versus two-tier design
- JAX-RS web services
- JAX-RS web services in the HenleyServer application



Three-Tier Architecture Using REST



Steps to Generate RESTful Web Services in NetBeans

1. Ensure that the prerequisites are met:
   - Add Jersey 1.8 to the project library.
   - Add JAX-RS 1.1 API to the project library.
   - Add the `HenleyModel` project with Entity classes to the `HenleyServer` project.
   - Ensure that the connection pool and JDBC data source has been created on the GlassFish server.
   - Ensure that the persistence unit has been created and configured in the project.
   - Ensure that the JAXB annotations are added to the JPA Entity classes.
2. Generate web services:
   - Create RESTful web services from the entity classes.
   - Validate the generated web service classes.
   - Validate the configuration in the `web.xml` file.

ORACLE

This slide provides an overview of the steps involved in generating RESTful web services using NetBeans.

JAXB annotations are added directly to JPA entity classes.

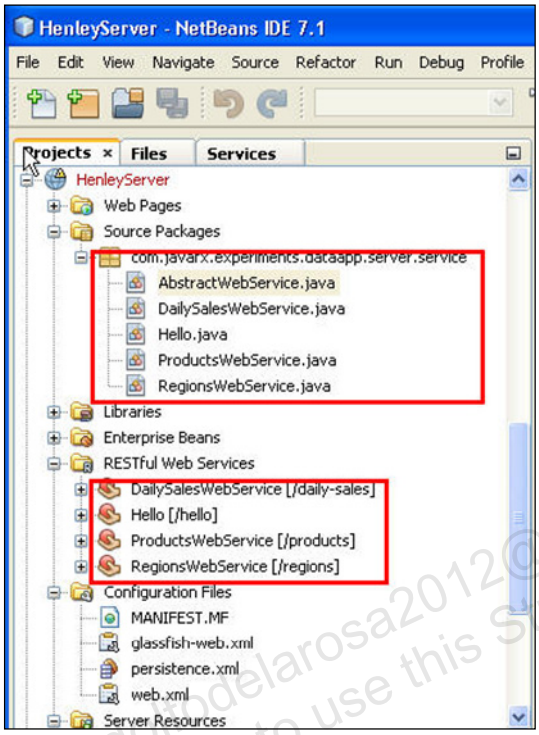
Services are generated as EJB session facades.

A session facade is a design pattern. As stated in the core J2EE pattern catalog, it attempts to resolve common problems that arise in a multi-tiered application environment, such as:

- Tight coupling, which leads to direct dependence between clients and business objects
- Too many method invocations between client and server, leading to network performance problems
- Lack of a uniform client access strategy, exposing business objects to misuse

A session facade abstracts an application's underlying business object interactions and provides a service layer that exposes only the required functionality. Thus, the session façade hides from the client's view the complex interactions between the objects. The session bean also manages the life cycle of business objects. The session bean creates, locates, modifies, and deletes objects as required by the workflow.

Examine the RESTful Web Services in HenleyServer



RESTful web services in Java rely on the JPA to communicate with a database. Specifically, RESTful web services rely on *entity classes* and a *persistence unit*, as defined in the Persistence API. Entity classes are Java classes that map to objects in a relational database.

The screenshot in the slide shows the list of RESTful web services generated in the HenleyServer application.

By creating web services as session facades, NetBeans IDE creates one service class for each entity class. The figure shows the project structure of a Java EE 6 RESTful service created from the entity classes, but with the service classes generated as session facades.

The other service classes extend the abstract session facade and call on its methods for a specific entity class. NetBeans IDE creates an abstract session facade class. This abstract class contains the code used by all service classes for managing entities.

Examining the DailySales Web Service Code

```
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
@Stateless
@Path("/daily-sales")
public class DailySalesWebService extends
AbstractWebService<DailySales> {
    public DailySalesWebService() {
        super(DailySales.class);
    }
    @GET
    @Override public List<DailySales> findAll() {
        return super.findAll();}
   @POST
    @Override public void create(DailySales entity) {
        super.create(entity);}
    }
```

The code in the slide shows the DailySalesWebService class. The use of session facades results in simpler code. Customizing and maintaining this code should also be much easier.

The DailySalesWebService class calls the methods of AbstractWebService, replacing the generic class parameters with references to the DailySales entity class.

The service class is easy to read, consisting of HTTP method-annotated method calls that refer back to the abstract class.

Testing DailySales RESTful Service

- Deploy and run the HenleyServer project.
- Test the web service URL in a web-browser:

```
http://localhost:8080/HenleyServer/resources/daily-sales
```

- The result of the `get` operation shows all the records of the Daily-Sales table in XML format.

Quiz

Select the correct statements about applying JAX-RS in your application:

a. You would choose to use JAX-RS for your web application because it is easier for many types of clients to consume RESTful web services while enabling the server side to evolve and scale.

b. RESTful web services in Java rely on HTTP to communicate with a database.

c. Jersey is the JAX-RS RI and is used to implement RESTful web services in your application.

d. RESTful web services rely on *entity classes* and a *persistence unit*, as defined in the Persistence API.

Answer: a, c, d

Summary

In this lesson, you should have learned how to:

- Compare the HenleyApp two-tier design and the HenleyApp three-tier design
- Describe a RESTful web service
- List the web services used in the HenleyApp application
- Describe how the RESTful web services were developed in the HenleyServer application

Practice 11: Overview

- Practice 11-1: Reviewing Basic Concepts of Java Web Services
- Practice 11-2: Examining `BrokerToolServer`'s Web Services