LearnQuest | 1601 Market Street
19th Floor | Philadelphia, PA 19103
Telephone: 610-206-0101 | www.learnquest.com

Web Services using REST

WDXM-361

Course Description:

This Web Services Using Rest course is designed to provide students with an introduction to Representational State Transfer architectural style of application development. The course presents the best practices for development of RESTful applications.

Course Objectives:

Upon completion of the course, students will be able to:

- Describe the RESTful application architectural style
- Describe the best practices with REST web services
- Design web services as RESTful style
- Contrast REST with SOAP and web services

Audience:

- Programmers

Prerequisites:

- Some understanding of programming concepts and program design

Duration:

- 1 Day

Course Topics:

I. Overview of Web Services

- Understand Web Services
- Characteristics of a good Web Service
- Web Service Standards and API's
    - Simple Object Access Protocol (SOAP)
    - Java API for XML Web Services (JAX-WS)
    - Java Architecture for XML Binding (JAXB) annotations
    - Java API for RESTful Web Services

II. Overview of REST

- What is REST?
- Why is it called Representational State Transfer?
- Understanding why REST is an Architectural Style and not a standard

III. Designing REST Web Services

- Designing REST Web Services
    - Principles and best practices with REST web services
- Examples of Services Designed in a RESTful Style
- Logical URLs vs. Physical URLs
- REST Web Services Characteristics
- Java Architecture for XML Binding (JAXB) annotations
- Creating a Root RESTful Resource Class
- JAX-RS Annotations

IV. Security, SOAP and REST

- REST Security and Validation
- Compare SOAP and REST
- Compare WSDL, WADL, and OpenAPI/Swagger

Web Services using REST

(WDXM-361)

Table of Contents

THE QUEST FOR KNOWLEDGE IS A LIFELONG JOURNEY

Module 1

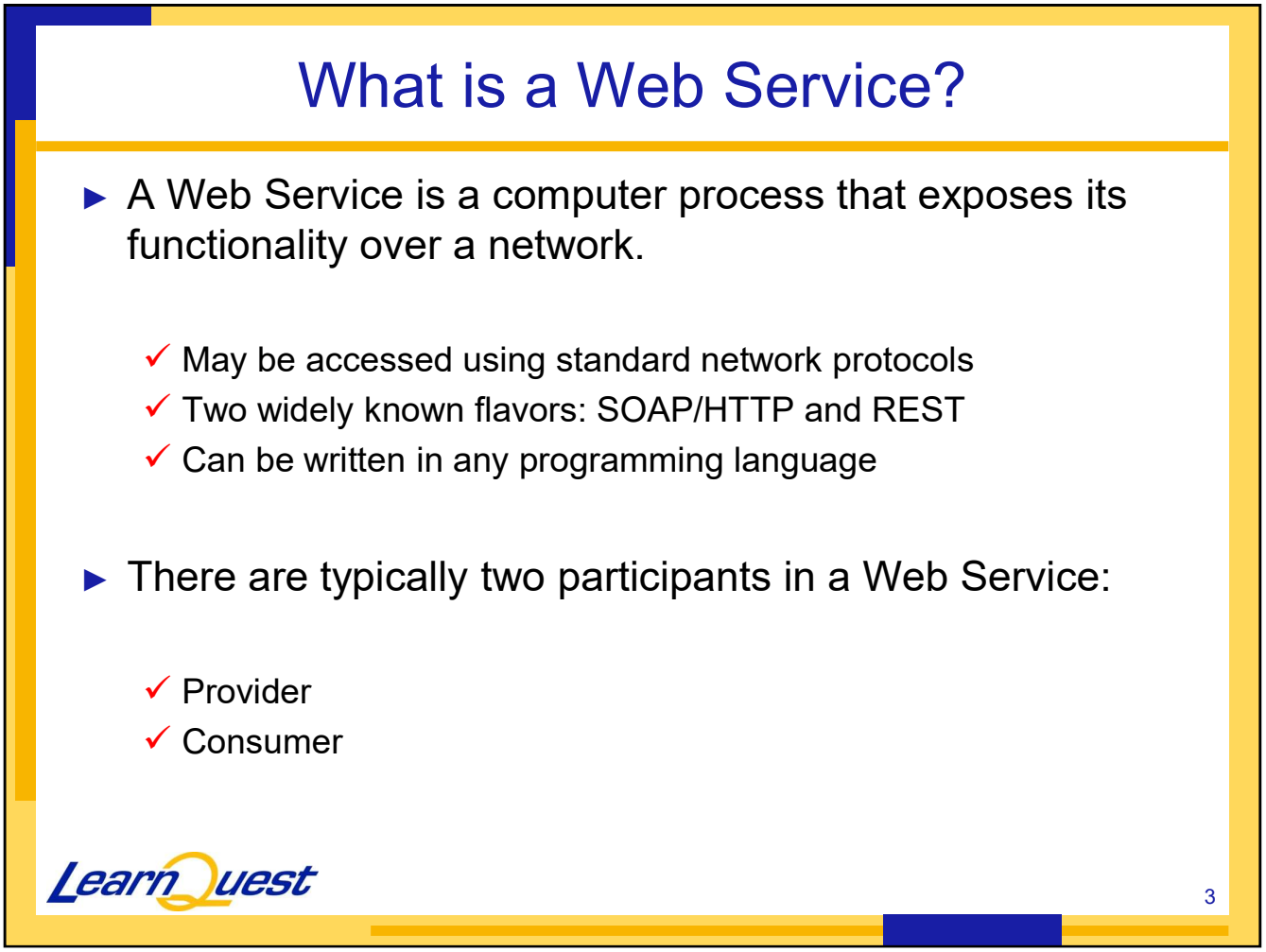
Overview of Web Services

Objectives

- In this module, participants will learn the following:
  - ✓ What is a Web Service?
  - ✓ Characteristics of a good Service
  - ✓ Standards and APIs
  - ✓ Apache CXF

LearnQuest

2



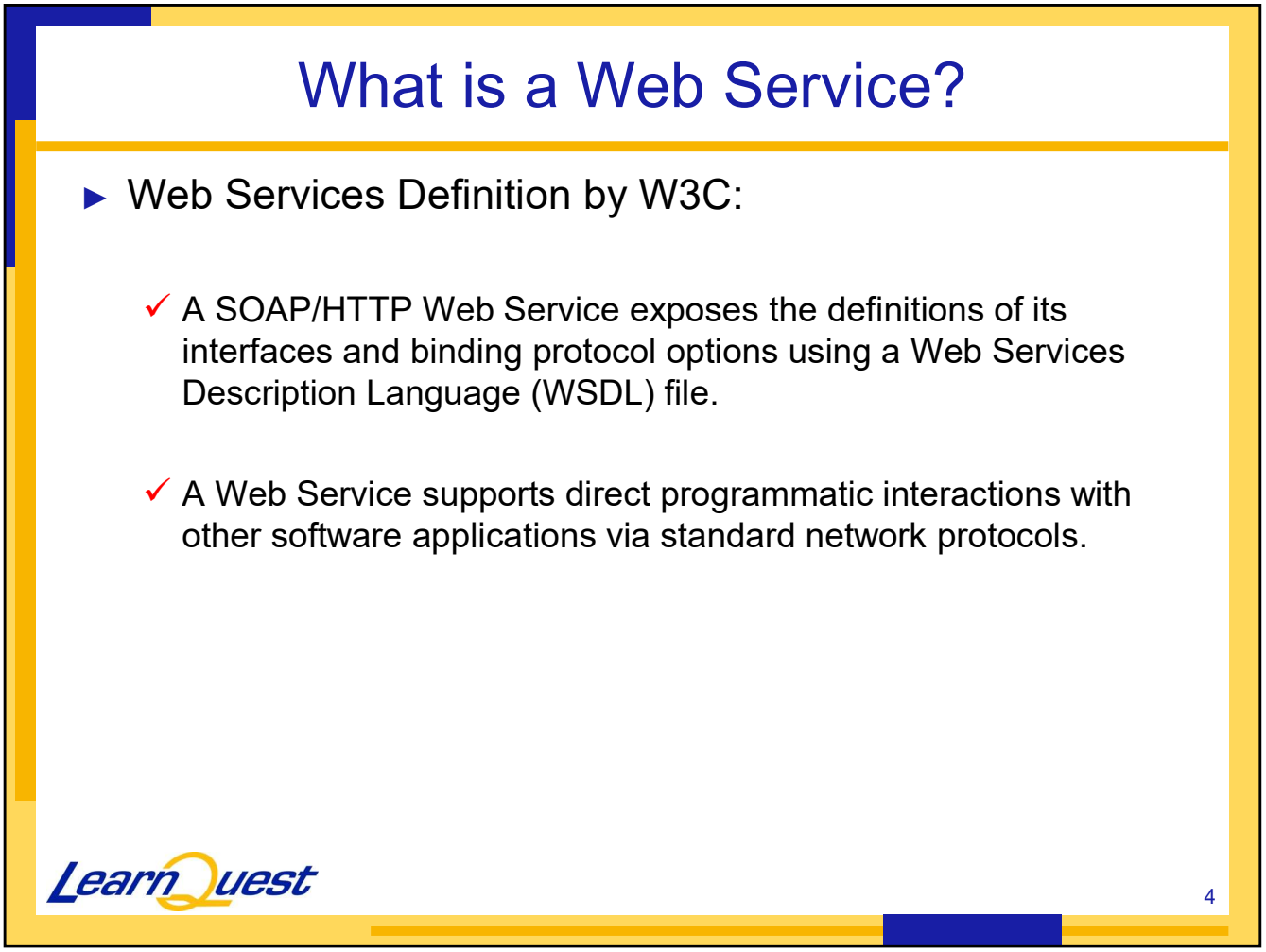
Web Services have two widely known flavors: SOAP/HTTP and REST. Programmatic clients must be able to read and/or write text streams. SOAP/HTTP Web Service interfaces are described using WSDL (Web Services Description Language).

There are typically two participants in a Web Service:

Provider – A provider creates the web and its interface definition

Consumer – A consumer obtains the interface definition and creates a client to access the service

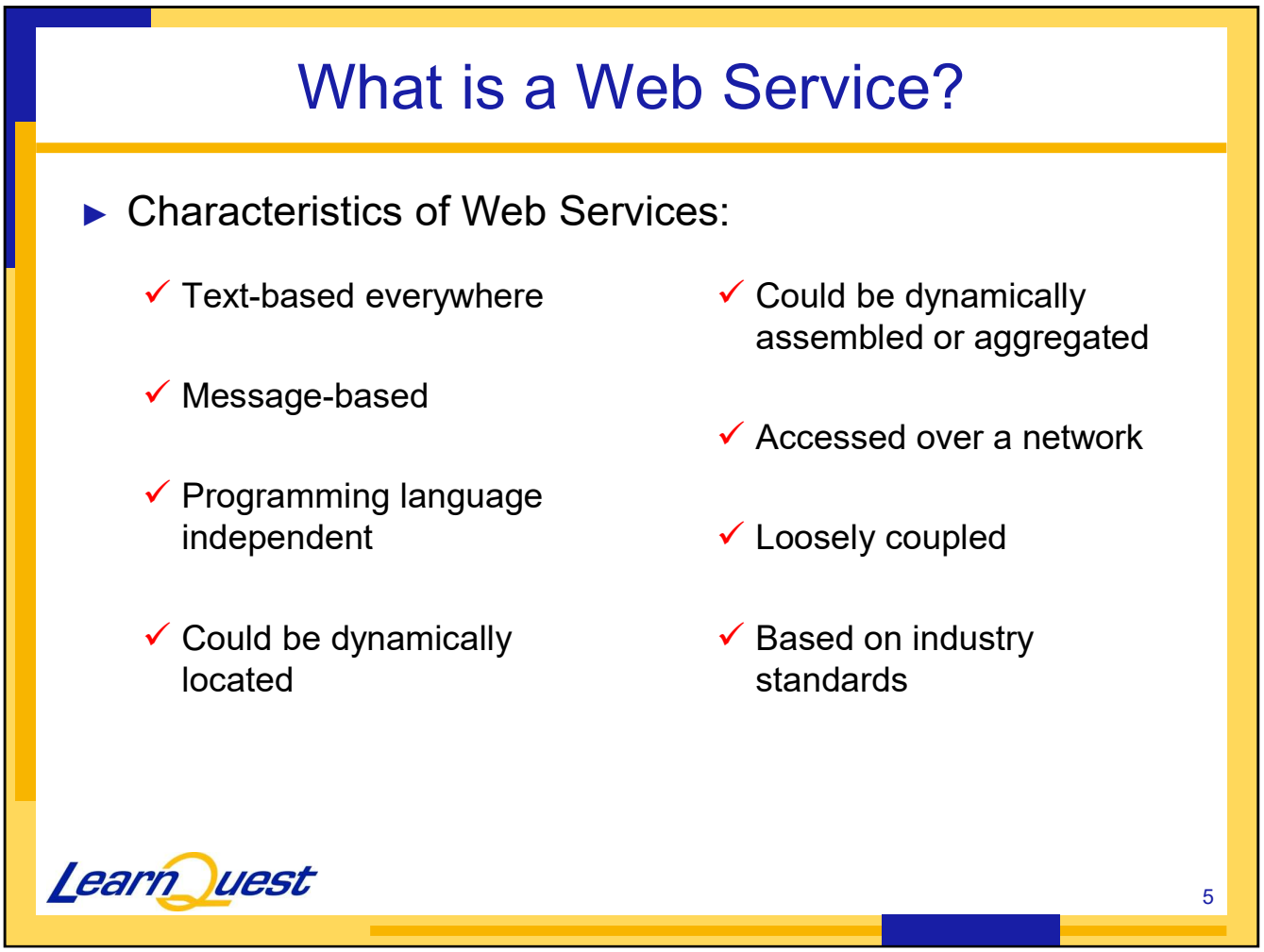
There can also be three participants—the third being a registry in which a provider can register the service. This is also known as the UDDI registry.



A Web Service is a software application identified by a URI.

W3C (World Wide Web Consortium): An international consortium of Member organizations, staff members, and the public users work together to develop Web standards. W3C's mission is as follows: "To lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web."

URI: A URI (Uniform Resource Identifier) is used to identify any type of resource from a given location, name, or both. The resource content can be text file, image file, audio file, or program. A commonly used form of URI is the webpage address. A webpage address is an extension format of URI called URL (Uniform Resource Locator). The URL format specifies that a file that can be accessed using a network protocol and location.



Text-based everywhere: Web Services can utilize plain text as the data representation and data transportation layers. Using plain text eliminates any networking specifics, operating system, or platform binding configuration.

Message-based: A message is considered a unit of communication within a Web service.

Programming language independent: Web Service implementation does not have to be implemented in any specific language.

Could be dynamically located: Web services allow change at runtime and accessibility to occur.

Could be dynamically assembled or aggregated: Web services can be assembled at runtime or compile time.

Accessed over a network: Web Services can be accessed over standard networking protocols.

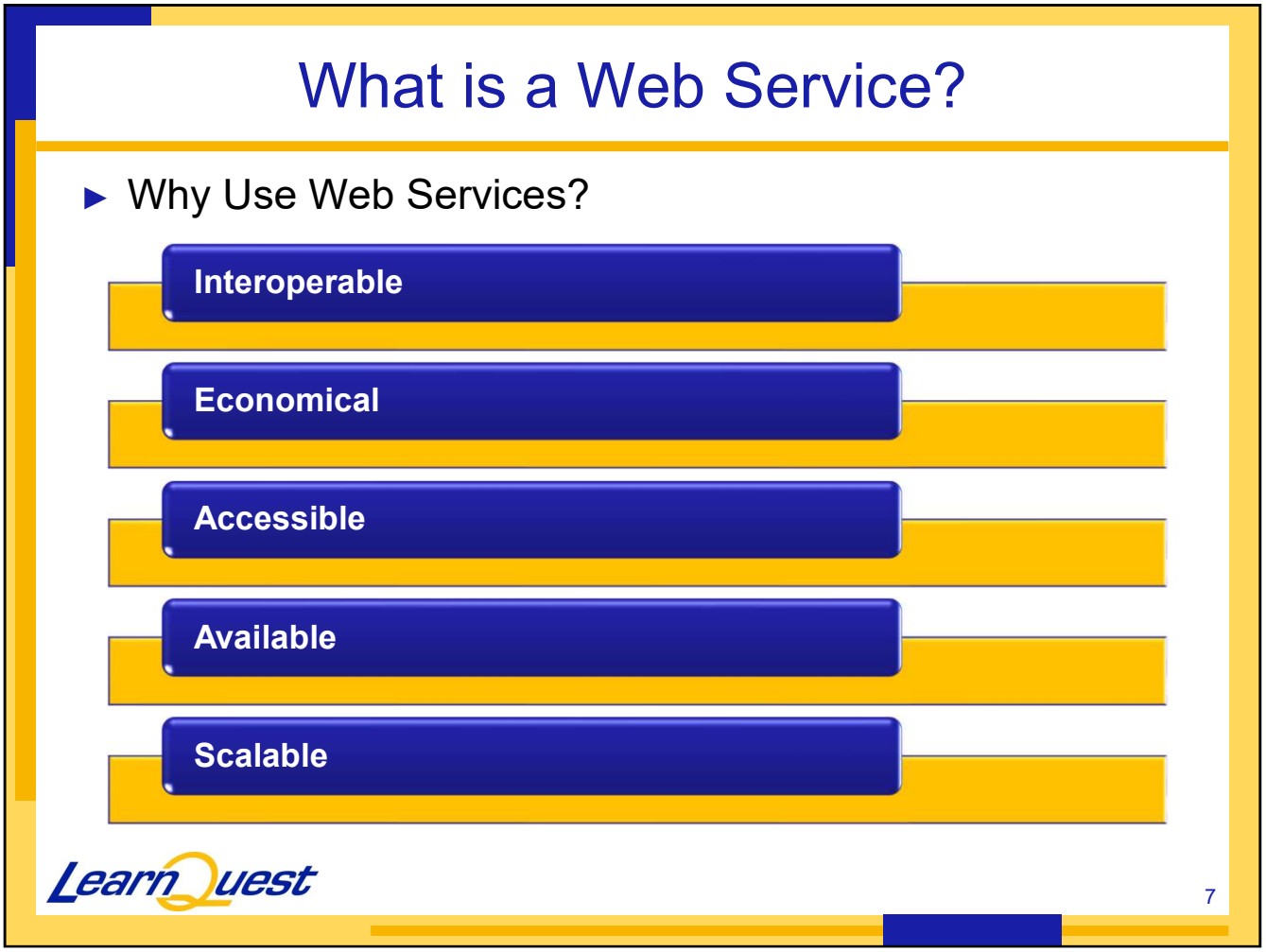
Loosely coupled: The web service interface can change over time without affecting the consumer of a web service.

Based on industry standards: Web Services are defined from general IT members from an industry perspective as opposed to a vendor specific implementation or need.

Coarse grained: Web Service operations should perform multiple tasks. The advantage of the result will be that clients can avoid multiple calls over the network to perform one task.

Quality of service:

- **Fault reporting**: Caller should be notified if there is an exception during request processing.
- **Execution time**: Reasonable response time is necessary to the success of a web service. This can impact the size of a service.
- **Security**: If required, security aspects should be incorporated.



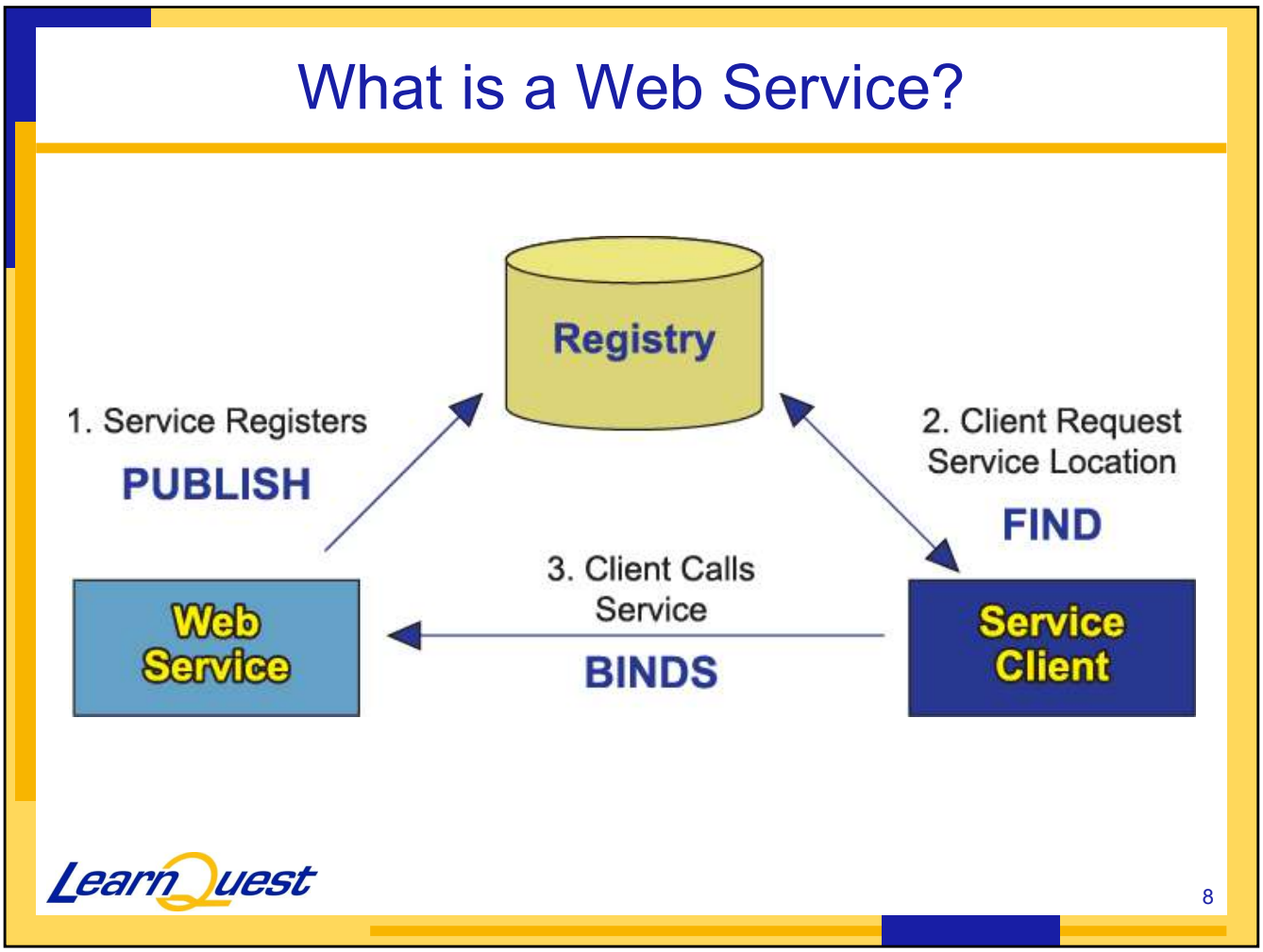
Interoperable: Connect across heterogeneous networks using web-based standards

Economical: No installation needed

Accessible: Legacy assets and applications may be exposed and made accessible using a network protocol

Available: Services on a variety of devices are accessible anywhere and any time

Scalable: No limits on scope, resources, or number of applications



Software developers use the registry's API to publish services (put information about the services in the registry) and query the registry to discover services matching various criteria.

The owners of Web Services publish them to the registry. Once published, the registry maintains references to the Web Service description and to the service. Clients search the registry, find the service need, and retrieve information about the service. These details include the service invocation, and possibly other information to help identify the service and its capabilities.

Initially, vendors offered open UDDI Servers to the general public to register their web services. Currently, most companies configure and maintain their own internal servers, few of which use UDDI, one of the industry's more spectacular failures.

For Service-Oriented Architectures (SOA) that have not embraced Microservices, it is typical to have an Enterprise Service Bus (middleware) perform the client lookup, rather than expect each client to do it. The Microservice philosophy encourages embedding that behavior in each client.

SOAP/HTTP Standards and APIs

- **SOAP** provides the definition of the XML-based information that can be used for exchanging information between peers in a distributed environment.
- It relies on:
  - ✓ XML for its message format
  - ✓ Some protocol, usually HTTP, for message transmission

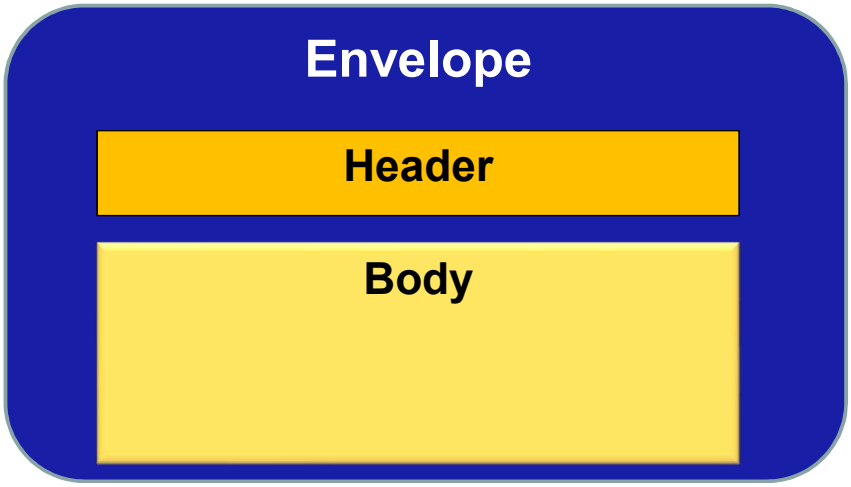
LearnQuest

9

A SOAP message is sent to a web service, usually in the body of an HTTP request. The Web Service responds either with a SOAP message or a SOAP fault. Regardless of the response content, the receiving application is expected to parse the response and translate the response.

SOAP/HTTP Standards and APIs

- A SOAP message consists of an envelope, which encloses an optional **header** and a required **body**.



LearnQuest

10

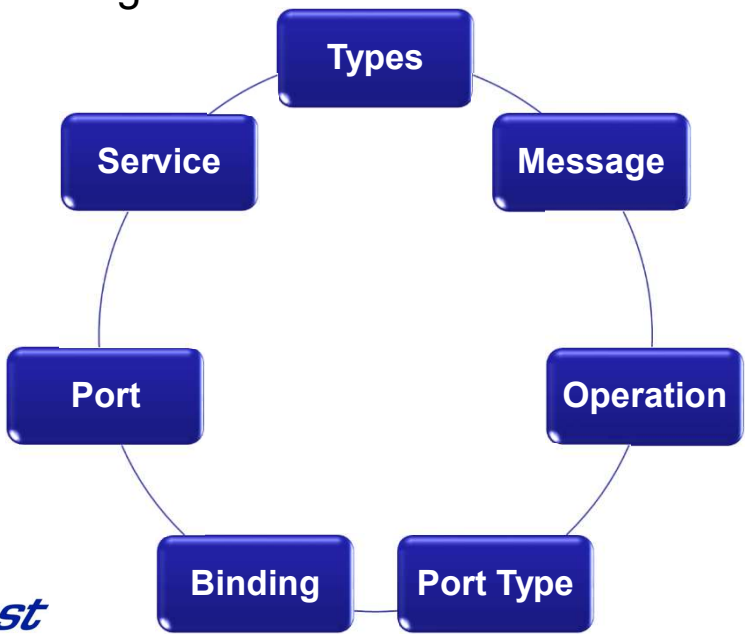
The message sent from a client contains instructions on which function to call and what arguments to provide. The message returned to a client may contain data returned from the function call.

Header: optional section to add information not part of the payload

Body: mandatory section that contains the main payload

SOAP/HTTP Standards and APIs

- **WSDL** provides the ability to describe communications in a structured way (i.e. as an XML document). It consists of the following seven sections:



LearnQuest

11

Types– a container for data type definitions

Message–definition of the data being communicated

Operation–description of an action supported by the service

Port Type–set of operations supported by endpoints

Binding– a concrete protocol and data format specification for a particular port type

Port– a single endpoint defined as a combination of a binding and a network address

Service– a collection of related endpoints

SOAP/HTTP Standards and APIs

- **JAX-WS** stands for *Java API for XML Web Services*
  - ✓ Uses annotations to simplify development and consumption of web services
  - ✓ Is a successor to **JAX-RPC**, an API to work with Remote Procedure Call based web services
  - ✓ Uses **JAXB** for data binding (for converting XML to Java objects and vice-versa)

LearnQuest

12

API is defined in packages:

- javax.xml.ws
- javax.jws
- javax.jws.soap

JAX-WS ships as part of Java SE 6.

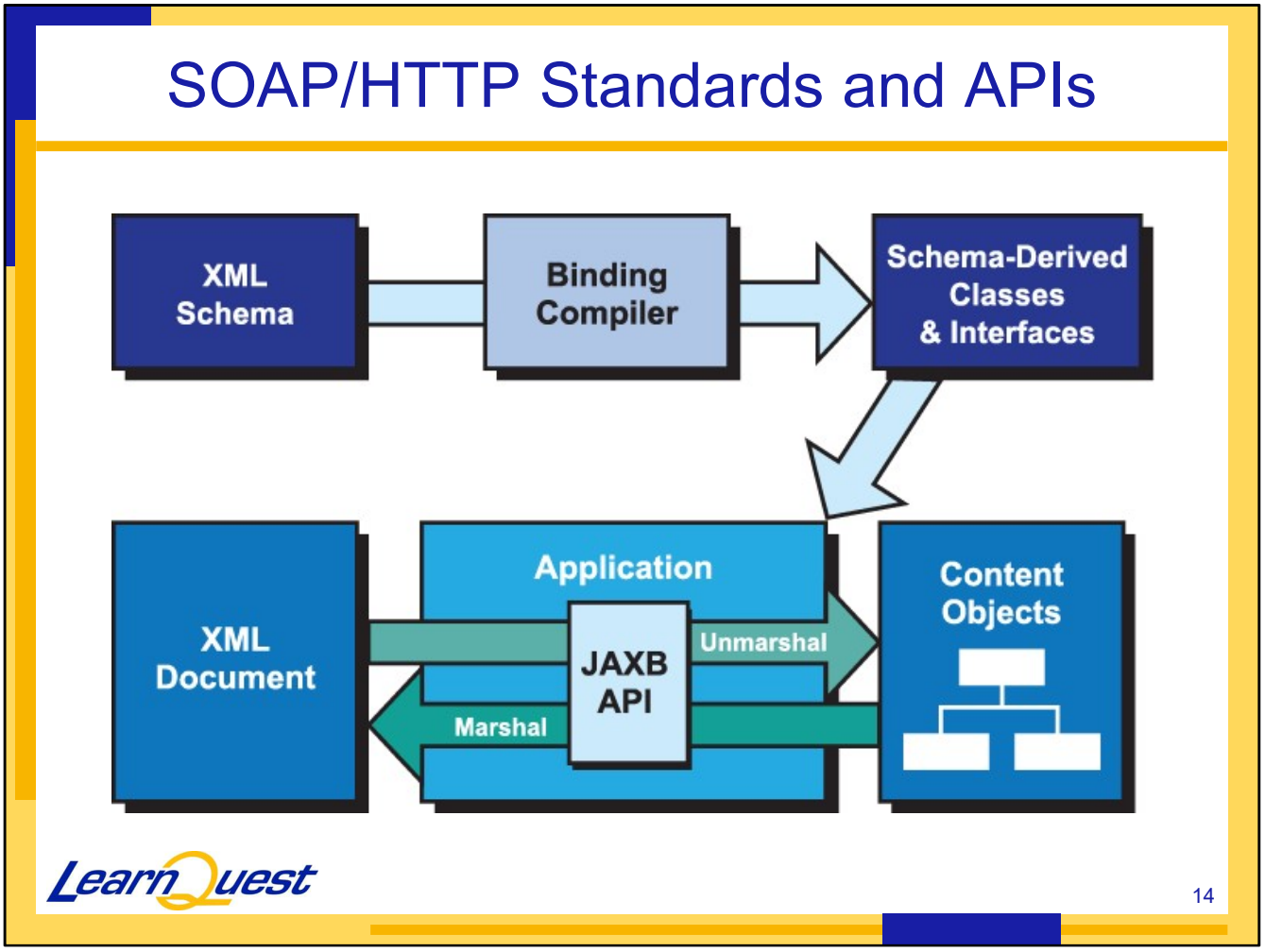
SOAP/HTTP Standards and APIs

- **JAXB** stands for *Java Architecture for XML Binding*
  - ✓ Makes it easier to read and write XML documents
  - ✓ You can create an XML schema, and JAXB can generate Java classes from it.

LearnQuest

13

Objects of Java classes can be marshaled to XML, or XML files can be un-marshaled to Java objects. JAXB is part of Java SE 6 and defined in package javax.xml.bind.



@XMLRootElement annotation

- The @XMLRootElement annotation on the class name declares it as a JAXB entity
  - ✓ Indicates the class that represents an entity managed by a web service
  - ✓ This tells a serialization mechanism to marshal and un-marshal the properties of the entity
    - Using XML as the serialization format
  - ✓ The annotations can drive the generation of XML Schemas
    - Schemas may be used to generate JAXB annotated Java classes

LearnQuest

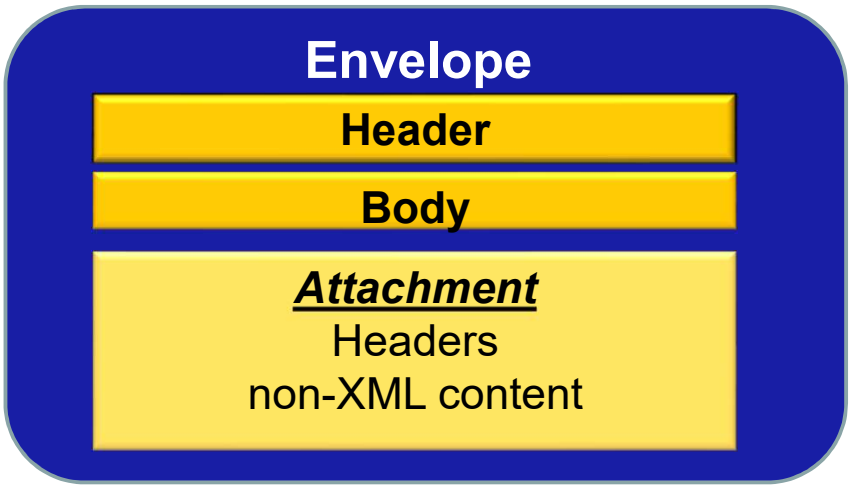
15

Entities have data that is persisted. JAXB provides a way to use XML as the content structure to be passed and Java classes as the application representation of that data. Using XML allows us to use many transport mechanisms that can easily carry XML payloads and web services.

It is possible to annotate the fields and methods within the class to customize the serialization (the bindings).

Standards and APIs

- **SAAJ** stands for *SOAP with Attachments API for Java*
  - ✓ Used to send non-XML data with SOAP messages in the form of attachments



LearnQuest

16

SAAJ is part of Java SE 6 in the package javax.xml.soap.

SOAP with Attachments (SwA) is a non-standard technology. The WS-I approved mechanism adopted industry-wide is MTOM (https://en.wikipedia.org/wiki/Message_Transmission_Optimization_Mechanism) with XOP (https://en.wikipedia.org/wiki/XML-binary_Optimized_Packaging). Support for MTOM/XOP is built into many vendor SOAP stacks, including JAX-WS (Java's standard for SOAP-based Web Services).

Standards and API's

- **JAX-RS** stands for *Java API for RESTful Web Services*
  - ✓ Used to create/consume services according to REST style

LearnQuest

17

REST: Representational State Transfer is a concept in which the request URL/URI represents a resource in a particular state.

JAX-RS is part of the Java EE 6 API and javax.ws package.

Although technically one could code REST-based services using plain Java Servlets or JAX-WS, it would be sub-optimal. JAX-RS is a purpose-built specification for doing so. It is the de jure standard for REST using Java, although many Spring developes eschew JAX-RS for Spring Web MVC or Spring WebFlux.

REST

- A RESTful web service (also called a RESTful web API) is a simple web service implemented using HTTP and the principles of REST, which is an *Architectural Style*.
- Such a service is viewed as a collection of resources, with three defined aspects:
  - ✓ A URI (Uniform Resource Identifier) for each resource
  - ✓ A standard, constrained, set of operations supported by the web service using HTTP methods: POST, GET, PUT, and DELETE
  - ✓ A representation of the resource, using MIME types, exchanged between the client and the service

LearnQuest

18

Aspects of REST:

- Hierarchical resources, each with a *URI*, *e.g.*, http://www.parkingrus.com/cars and http://www.parkingrus.com/cars/1HGBH41JXMN109186
- REST endpoints exchange (transfer) *representations* of the resource using data described using MIME types. This is often JSON, XML, or YAML but can be any other valid MIME type.
- The set of HTTP methods (*e.g.,* POST, GET, PUT, or DELETE) supported by the web service for any given resource, and the interpretation of what that means.

Summary

- In this module, participants learned the following:
  - ✓ What a Web Service is
  - ✓ Characteristics of a good Service
  - ✓ Standards and APIs

LearnQuest

19

Questions

LearnQuest

20

THE QUEST FOR KNOWLEDGE IS A LIFELONG JOURNEY

Module 2

Overview of REST

Objectives

- In this module, participants will learn the following:
  - ✓ What is REST?
  - ✓ Why is it called Representational State Transfer?
  - ✓ Understanding why REST is an Architectural Style and not a standard

LearnQuest

2

What is REST?

- **Representational State Transfer** (**REST**) is a style of software architecture for distributed systems, such as the World Wide Web.
  - ✓ REST differs from other network-based architectures in its emphasis on a uniform interface between components.

LearnQuest

3

Representational State Transfer (REST) was introduced by Roy Fielding. The REST architecture consists of clients and servers. Initially, a client makes a request to a server. A server will process the request and return an appropriate response.

“The motivation for developing REST was to create an architectural model for how the Web should work, such that it could serve as the guiding framework for the Web protocol standards. REST has been applied to describe the desired Web architecture, help identify existing problems, compare alternative solutions, and ensure that protocol extensions would not violate the core constraints that make the Web successful."

-Roy Fielding

REST: Representational State Transfer

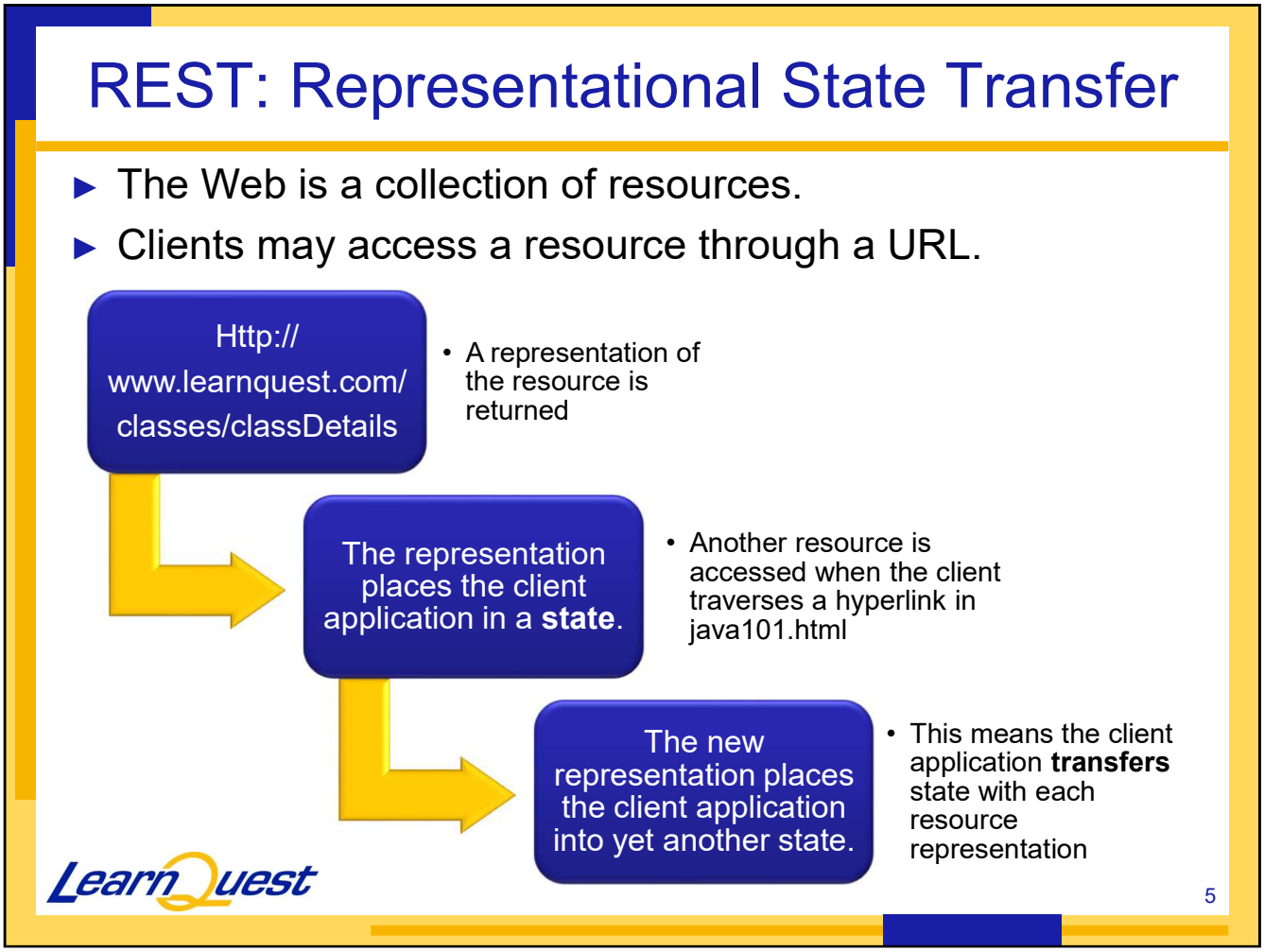
- A RESTful web service (also called a RESTful web API) is a simple web service implemented using HTTP and the principles of REST:
  - ✓ A web request and response is built based on the transfer of resources that can be represented in a variety of formats.
  - ✓ Typically, a representation of a resource is a document that captures the state of a resource.

LearnQuest

4

REST is a aggregation of resources with three core components of REST:

- The base URI for the web service, such as http://example.com/resources/
- A reference of the MIME type of the data supported by the web service. (Examples: JSON, XML)
- HTTP methods support by the web service (e.g., POST, GET, PUT, or DELETE).



Clients may access a resource through a URL:

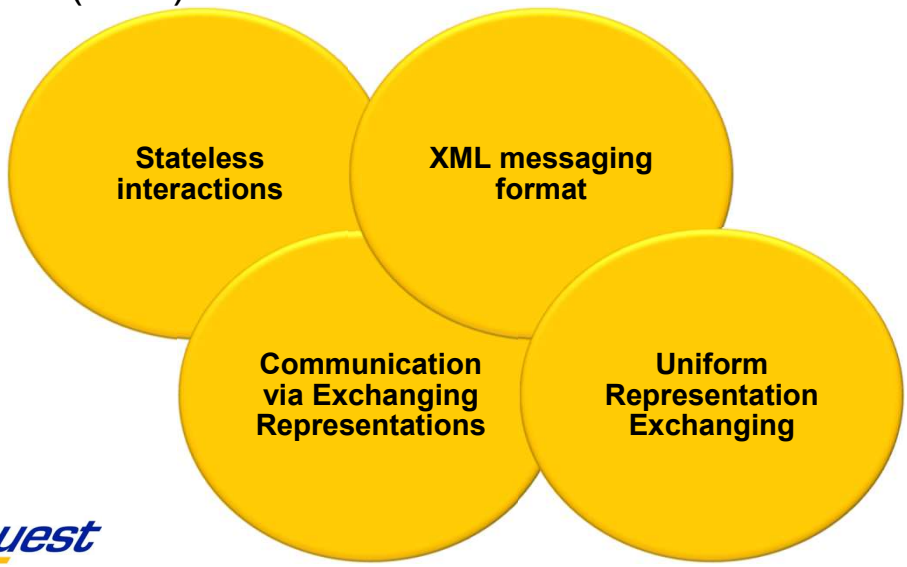
- Http://www.learnquest.com/classes/classDetails
- A representation of the resource is returned (e.g. java101.html).
- The representation places the client application in a **state**.
- The result of the client traversing a hyperlink in java101.html is that another resource is accessed.
- The new representation places the client application into yet another state. This means the client application **transfers** state with each resource representation, resulting in Representational State Transfer!

"Representational State Transfer is intended to evoke an image of how a well-designed Web application behaves: a network of webpages (a virtual state-machine) in which the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for his or her use."

-Roy Fielding

REST: Representational State Transfer

- Core component of the style is a resource
- Each resource accessible on the web has a unique identifier (URL).



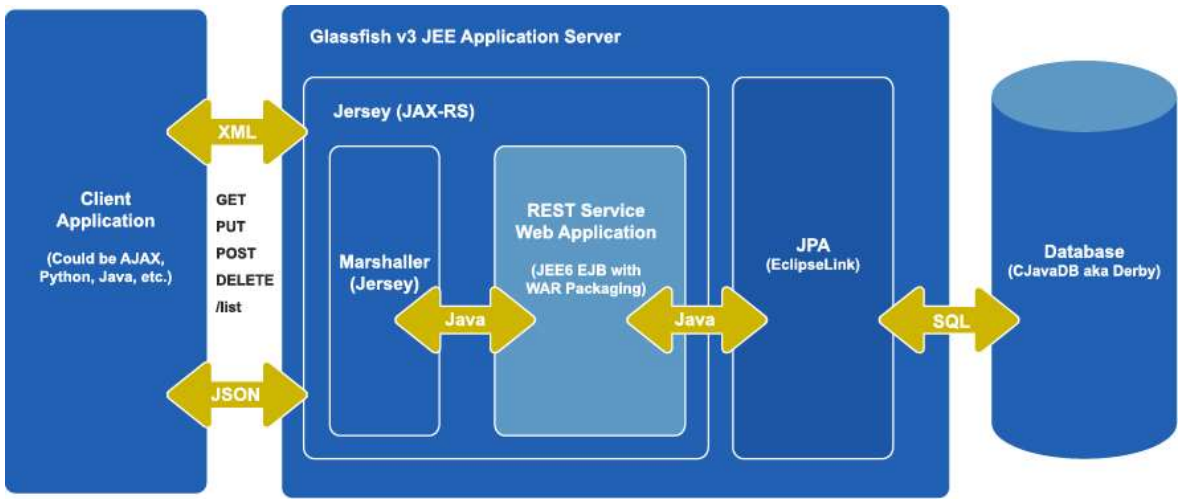
LearnQuest

6

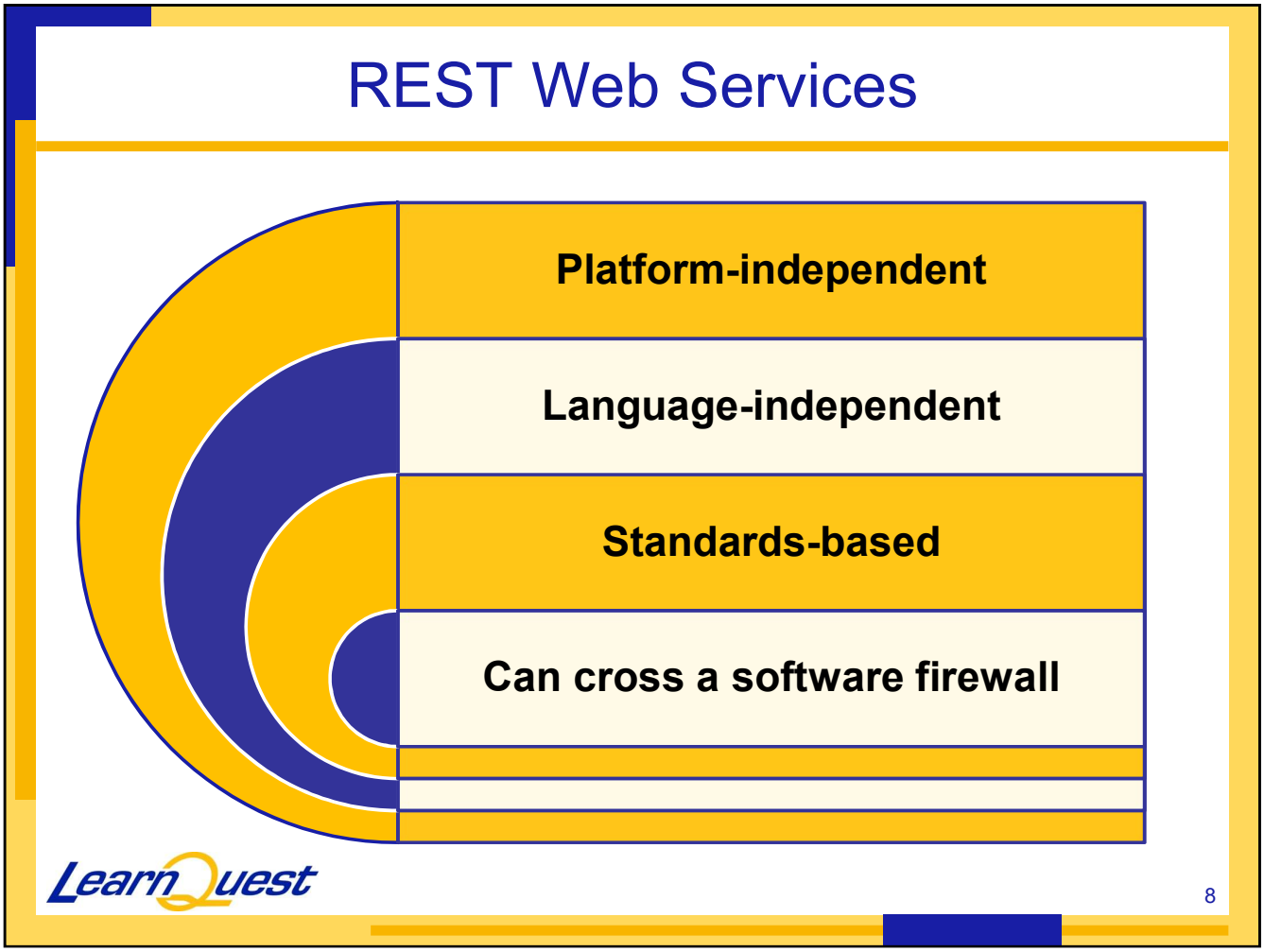
- Components communicate by exchanging representations of the resource rather than performing operations on the resource.
- Each resource exposes a uniform set of methods for exchanging representations:
    - GET, POST, PUT and DELETE
- All interactions with the resources are stateless.
- XML may be used as the messaging format.

REST Web Services

- REST architectural style:
  - ✓ Based on W3C's web architecture
  - ✓ Resource oriented and stateless



REST Architecture style is derived from the traditional client-server architecture. It represents a network of systems. The main difference with the REST architecture style compared to the traditional client-server architecture is the focus on a uniform interface between components.



Standards-based (runs on top of HTTP)

REST Web Services

- REST offers no built-in:
  - ✓ Security features
  - ✓ Encryption
  - ✓ Session management
- REST can be used on top of HTTPS for encryption.
- REST is a lightweight alternative to RPC (Remote Procedure Calls) and standards based web services (SOAP/HTTP)

LearnQuest

9

The design advantage of REST is that REST does not implement any vendor specific security. REST does not have built-in support for security at all but can use the standard secure networking protocol for encryption.

What REST is Not

- REST is not a standard.
  - ✓ It uses standards, such as:
    - HTTP
    - URL
    - XML/HTML/GIF/JPEG
    - Text/xml
    - Text/html
    - Image/gif
    - Image/jpeg

LearnQuest

10

The W3C does not define the REST specification. Vendors do not have a REST developer's toolkit. REST is an architectural style. You can only understand it and design your Web services in that architecture style.

Simon Says

- "*WS was the reaction of the CORBA crowd to the web: complex and over-engineered. I believe [REST] is all people will be using in a few years, apart for custom build infrastructure within bounded environments.*

  The problem with remote method invocation across business boundaries is it requires a level of functional interdependencies between business units that is unreasonable to assume. The more semantics associated with a data object the riskier it gets. Even in trading communities the semantic richness needs to be watched.

  It's very, very hard to get people to agree data structures, let alone their semantics and to get them to agree all that and codify it as an object architecture seems brittle beyond belief.

  My prescription back in 1997 for objects inside organisations and XML between them has become WS- within a business function and REST between business functions.*" – Simon Phipps

What REST Is (And Is Not)

- REST is **not** “the web way of doing RPC”
- REST requires you to rethink the nature of the communication between your endpoints.
  - ✓ REST uses a hierarchical structure to refer to resources.
  - ✓ REST uses a **common** interface to resources.
- REST **is** about creating and moving “representations” of resources across the network.
  - ✓ The client indicates what type of representation it is sending, and what type(s) it will accept as a response.
  - ✓ The server indicates what type of representation it is providing as a response.

LearnQuest

A Car at REST

- A car has properties
  - ✓ Color, VIN, Mileage, Manufacturer, *etc.*
- A car has subcomponents
  - ✓ Radio, Engine, Tires, Gas Pedal, Break Pedal, *etc.*
  - ✓ Each of these subcomponents also has properties.
- With REST, each (sub-)component has a URI, and can receive commands to set its properties, get its properties, add a sub-component, and be deleted.
- The command interface is uniform across all resources.

LearnQuest

Car Resources

- A possible hierarchical mapping:
  - ✓ Cars: `/cars`
  - ✓ A Car: `/cars/{id}`
  - ✓ A Car component: `/cars/{id}/<component>`
- The common interface:
  - ✓ **C**reate
  - ✓ **R**etrieve
  - ✓ **U**pdate
  - ✓ **D**elete

LearnQuest

A Bank Example

- A possible hierarchical mapping
  - ✓ `/account/{id}`
  - ✓ `/user/{id}/accounts/<type>`
  - ✓ `/account/{id}/transactions`
  - ✓ `/account/{id}/transactions/{id}`
  - ✓ `/account/{id}/transactions/batches/{id}`
- The common interface:
  - ✓ **C**reate
  - ✓ **R**etrieve
  - ✓ **U**pdate
  - ✓ **D**elete

What **Is** REST?

- **REST** = **Re**presentational **S**tate **T**ransfer
- REST is the invention of Dr. Roy T. Fielding, Ph.D., Chief Scientist of Day Software, Principal Author of the HTTP 1.1 Specification, Co-Founder and Past Chairman of The Apache Software Foundation, Chairman of the Apache HTTP Server Project, Designer of Waka (http://en.wikipedia.org/wiki/Waka_%28protocol%29), Member of the initial OpenSolaris Board, …, OK you can stop reading now.
  - ✓ REST is the subject of Roy's Ph.D. Dissertation
    - http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
  - ✓ REST is a Software Architectural Style. REST is **not** a Software Architecture.
    - Huh? OK, we'll let Roy explain…

LearnQuest

Software Architecture Research

- *A **software architecture** is an abstraction of the run-time elements of a software system during some phase of its operation. A system may be composed of many levels of abstraction and many phases of operation, each with its own software architecture.*
  - ✓ *A software architecture is defined by a configuration of architectural elements — components, connectors, and data — constrained in their relationships in order to achieve a desired set of architectural properties.*
  - ✓ *A configuration is the structure of architectural relationships among components, connectors, and data during a period of system run-time.*
- *An **architectural style** is a coordinated set of architectural constraints that restricts the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style.*
  - ✓ *A style can be applied to many architectures*
  - ✓ *An architecture can consist of many styles*

Slide courtesy of Roy T. Fielding, used with permission.

LearnQuest

The Web's Architecture

- Roy refers to architecture as being one level of abstraction above implementation.
- The architecture of the Web consists of the following elements:
  - ✓ Universal Resource Indicator (URI): **one** identifier standard for **all** resources
  - ✓ HyperText Transfer Protocol (HTTP): a standard protocol
  - ✓ HTML, XML, *etc.*: common representation formats for resources

LearnQuest

Architectural Styles

- *Architectural Styles are one abstraction level above Architecture*
  - ✓ *two abstraction levels above implementation*
  - ✓ *that's one too many for most folks*
- *An architectural style is a set of constraints*
  - ✓ *unfortunately, constraints are hard to visualize*
- *Consider the real world and Civil Engineering…*
  - ✓ *Impl: There is water next to the conference hotel, all the time.*
  - ✓ *Arch: The Sihl river flows next to the conference hotel*
  - ✓ *Style: Channel water into stable river flows for flood control*
- *In Software, our "laws of physics" are information and social theory*
  - ✓ *Which can make it very hard to decide on constraints*
  - ✓ *And very easy for a fool to break them by mistake*

Slide courtesy of Roy T. Fielding, used with permission.

LearnQuest

REST Constraints

- REST is an architectural *style*, with the following constraints documented by Roy:
  - ✓ *Uniform Interface*
    - *Resources are identified by only one resource identifier mechanism.*
      - *simple, visible, reusable, stateless communication*
    - *Access methods (actions) mean the same for all resources (universal semantics).*
      - *layered system, cacheable, and shared caches*
    - *Manipulation of resources occurs through the exchange of representations.*
      - *simple, visible, reusable, cacheable, and stateless communication*
    - *Actions and representations are exchanged in self-descriptive messages.*
      - *layered system, cacheable, and shared caches*
  - ✓ *Hypertext as the engine of application state*
    - *Each response contains a partial representation of server-side state*
    - *Some representations contain directions on how to transition to the next state*
    - *Each steady-state (page) embodies the current application state*
      - *simple, visible, scalable, reliable, reusable, and cacheable network-based applications*

REST Rationale

- *Maximizes reuse*
  - ✓ *Uniform resources having identifiers*
  - ✓ *Visibility results in serendipity*
- *Minimizes coupling to enable evolution*
  - ✓ *Uniform interface hides all implementation details*
  - ✓ *Hypertext allows late-binding of application control-flow*
  - ✓ *Gradual and fragmented change across organizations*
- *Eliminates partial failure conditions*
  - ✓ *Server failure does not befuddle client state*
  - ✓ *Shared state is recoverable as a resource*
- *Scales without bound*
  - ✓ *Services can be layered, clustered, and cached*
- *Simplifies, simplifies, simplifies*

Slide courtesy of Roy T. Fielding, used with permission.

LearnQuest

Benefits and Tradeoffs of REST Constraints

- Client-Server Model
  - ✓ Pro:
    - Decouple UI from Server
    - Simplify Server
- Stateless
  - ✓ Pro:
    - Simplify Server
    - Improve Scalability and Reliability
  - ✓ Con:
    - Less Efficient
- Optional Caching
  - ✓ Pro:
    - Improves Performance
    - Improves Efficiency
    - Improves Scalability
  - ✓ Con:
    - Less Reliable Content
- Uniform Interface
  - ✓ Pro:
    - Decouples Implementations
    - Improves Ability to Evolve
    - Improves Visibility (don't need special code to access)
  - ✓ Con:
    - Less Efficient
- Layering
  - ✓ Pro:
    - Improves Caching
    - Facilitates Load Balancing
    - Improves Scalability
    - Simplifies Clients
  - ✓ Con:
    - Increases Latency
- Extend Client w/ Downloaded Code
  - ✓ Pro:
    - Simplifies Clients
    - Improves Extensibility
  - ✓ Con:
    - Reduces Visibility (need the code)

LearnQuest

Summary of REST Benefits

- Maximizes Reuse of Resources
- Minimizes Coupling
  - ✓ One common interface to resources
  - ✓ Messages describe provided and acceptable representation type(s), facilitating evolution
- Facilitates caching, scalability and fail-safety.

Adoption and Challenges

- IBM and the rest of the industry have come to recognize the true value of REST, which has increasing hype of its own.
- **But** … REST has challenges, too.
  - ✓ For almost 7 years, Roy stayed relatively silent on REST, allowing a lot of misleading if not outright wrong information to be associated with his brainchild, including by its advocates, much less its detractors.
  - ✓ The toolset for REST development is not yet as mature as for simple WS-I compliant Web services.
  - ✓ Doing REST is not the easiest approach, and it requires re-learning for many programmers, since it differs from their daily techniques.
- So we're going to try to lay out what REST is in simple(r) terms, and try to get you started correctly.
- But keep in mind that REST is neither, in Roy's own words, "the only architectural style" nor about "the one true architecture."
  - ✓ Remember the constraints that define REST

LearnQuest

REST – It's How the Web Works

- REST is based on simple principles:
  - ✓ We address resources by URI
    - URI elements should be **nouns** (resources) not **verbs** (actions)
  - ✓ We use HTTP 1.1 (RFC 2616) *properly*
    - One consistent resource interface: CRUD
    - But I have never *seen* a CGI script that does what POST is *supposed to do*, and in fact, seen `doPost` and `doGet` documented as similar, when they are fundamentally *different!*
    - So perhaps it is more proper to say that REST is how the web is *supposed* to work.

LearnQuest

HTTP 1.1 And CRUD Mapping

- There is a direct mapping between the HTTP 1.1 methods and the classic CRUD operations:
  - ✓ HTTP POST **C**reate
  - ✓ HTTP GET **R**etrieve
  - ✓ HTTP PUT **U**pdate
  - ✓ HTTP DELETE **D**elete
- This is almost directly out of RFC 2616, but so very few people have ever even read it.
- Keep in mind that we are dealing with *resources*, not mere database records.
- There are other HTTP methods that you might use, but these are the basics, and the ones that we shall discuss.

LearnQuest

HTTP POST

- HTTP POST is our means to create new resources.
- The HTTP Specification documents the correct behavior:
  - ✓ *The POST method is used to request that the origin server accept the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI.*
  - ✓ *The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status, depending on whether or not the response includes an entity that describes the result.*
  - ✓ ***If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header.***
  - ✓ So … anyone seen this behavior actually done?

LearnQuest

HTTP PUT

- HTTP PUT is our means to update an existing resource, although its behavior isn't entirely pure, as noted below.
- The HTTP Specification documents the correct behavior:
  - ✓ *The PUT method requests that the enclosed entity be stored under the supplied Request-URI.* ***If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version*** *of the one residing on the origin server.* ***If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI****. If a new resource is created, the origin server MUST inform the user agent* via *the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request.*

LearnQuest

HTTP POST *vs*. HTTP PUT

- Since the HTTP PUT method is not purely an update, and can result in resource creation, what is the difference between it and the HTTP POST method?
- RFC 2616: "*The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI. The URI in a POST request identifies the resource that will handle the enclosed entity. In contrast, the URI in a PUT request identifies the entity enclosed with the request.*" (slightly redacted from the actual text)
- This is a fairly significant difference. The URI used with POST identifies a handler resource, whereas the URI used with PUT identifies the resource, itself.

LearnQuest

HTTP PUT & HTML <FORM>

- Oddly, the HTML 4 `<form>` element only accepts `GET` and `POST` for methods.
  - ✓ So I asked:
    - Q: W*hy isn't there a* `method=PUT` *for a form that is intended to modify an entity?*
    - A: There was, but unfortunately it was removed from HTML4 due to concern that it might be too hard to document. It is specified again in Web Forms 2: http://www.whatwg.org/specs/web-forms/current-work/
  - ✓ The point is that there really is sound theory about how the web is supposed to work. It isn't a conceptually difficult theory, but we need to understand that theory, and how things are supposed to work, in order to gain the true benefits of the web. REST requires a mind-shift, and even people working on web standards can make mistakes. The mistake with HTML 4 makes it hard to use a browser as a fully functional REST client. We need help to send the missing methods.

LearnQuest

HTTP GET

- HTTP GET is our means to retrieve resources.
- The HTTP Specification documents the correct behavior:
  - ✓ The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI.
  - ✓ Some behavior is designed to reduce traffic and support client caching of the content:
    - The semantics of the GET method change to a "conditional GET" if the request message includes an `If-Modified-Since`, `If-Unmodified-Since`, `If-Match`, `If-None-Match`, or `If-Range` header field.
    - The semantics of the GET method change to a "partial GET" if the request message includes a `Range` header field.
    - A partial GET requests that only part of the entity be transferred, as described in section 14.35 of RFC 2616.

LearnQuest

HTTP DELETE

- HTTP DELETE is our means to delete a resource.
- The HTTP Specification documents the correct behavior:
  - ✓ The DELETE method requests that the origin server delete the resource identified by the Request-URI.
  - ✓ A successful response SHOULD be 200 (OK) if the response includes an entity describing the status, 202 (Accepted) if the action has not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include an entity.
- This is perhaps the simplest of the methods to understand and use.

LearnQuest

URI and Query Strings

- Consider these two URIs:
  1. http://mydomain.tld/stuff?parameter=values
  2. http://mydomain.tld/stuff/parameter/values
  - ✓ Although they might semantically identify the exact same underlying object, the URIs are different, and they may not be treated the same.
  - ✓ Implementations that map using only URI path components, but not query parameters, may dispatch to a different resource handler.
  - ✓ Many HTTP caches avoid caching when the URI has query parameters.
- Consider the semantic structure represented by the URI:
  - URI path components provide a hierarchical structure
  - Query strings are unordered, non-hierarchical
  - ✓ For these reasons, using query strings where you mean to represent a hierarchical structure, as in the following examples:
    - http://localhost/users?UID=Noel
    - http://localhost/news?group=HTTP

  is a poor URI design. Instead, the hierarchies could be represented as:
    - http://localhost/users/Noel
    - http://localhost/news/newsgroup/HTTP

LearnQuest

URI and HTML Forms

- Another issue is caused by using query strings with HTML `<form>`.
- Recall that the `method` attribute can have values of `GET` and `POST`. When the `GET` method is used with a form, all form fields are encoded as query string parameters. When the `POST` method is used, the parameters are encoded in the body of the request.
- Consider the implications, since the resulting URI identifies the *resource* according to REST.

Representations

- The requester and responder have to agree on the representation(s) of the resource they will exchange.
  - ✓ The `Accepts` header describes the acceptable representation formats in preference order.
  - ✓ The `Content-Type` header describes representation being transmitted.
- As you come to understand REST, you will come to understand some of the portions of the HTTP Specification that have previously seemed unused, such as STATUS codes 201 (Created), 406 (Not Acceptable), 409 (Conflict), etc.

LearnQuest

The Car Example - Create

- `POST /cars`
  - ✓ Request body provides a representation of the new car.
  - ✓ Response provides a 201 Status and a Location header providing the URI of the new car resource.
- `Car car = CarFactory.newCar(…);`

Modify Radio

- `PUT /cars/{id}/radio`
  - ✓ Request body provides a representation of the specified car's (new) radio.
  - ✓ Response provides a 201 Status and a Location header providing the URI of a new radio, or 200/204 if the radio was only modified.

```
if (car.getRadio()
!= null) {
   car.getRadio().s
etStation(…);
   car.getRadio().s
etVolume(…);
} else {
   car.setRadio(new
Radio(…));
}
```

LearnQuest

Give It The Gas

- `PUT /cars/{id}/gaspedal`
  - ✓ Request body provides a representation that sets the gas pedal state.
  - ✓ Response provides 200 or 204 status.
- `car.getGasPedal().set(…);`

Where Is It Now?

- `GET /cars/{id}`
  - ✓ Response body provides a representation of the car.
    - We'll look in that representation for the "location" property.
  - ✓ Repeated GET requests might provide updated state, since the gas pedal is still depressed.
  - ✓ REST does not represent a hierarchical database. The application has semantically linked the state of the gas pedal with the location of the car.
- `car.getLocation();`

LearnQuest

A Few Other Examples

- `GET /cars`
  - ✓ Response body provides a representation of the known cars, including a URI for each.
- `GET /cars/{id}`
  - ✓ Response body provides a representation of a specific car.
- `GET /cars?color=blue`
  - ✓ Response body provides a representation of cars whose color is blue, including a URI for each.

- `Car[] cars = …?`
- What is the entity that would know about cars? REST is predicated on a resource hierarchy.

SOA versus REST?

- The fact that this question comes up so often, really indicates widespread misunderstanding of the core rationales for SOA and REST. Brought to their essences, rather than hype and unconstrained excess, SOA and REST are compatible and complementary.
- SOA (Service Oriented Architecture) really needs to be understood as a Business Process Oriented Architecture. The **Service** in SOA is a Business Service.
  - ✓ Business Processes, formally described and executable, should be the focus when establishing SOA in the enterprise.
  - ✓ Business Processes orchestrate interactions across Services.
- The real issue isn't SOA versus REST, but SOAP versus REST. Today, BPEL understands Services to be WS-* (SOAP) Web services. As BPEL evolves, *e.g.*, to support WSDL 2.0, REST can easily become part of the Business Process environment.
  - ✓ See: *"Developing Web Services Choreography Standards – The Case of REST vs. SOAP" by Michael zur Muehlen, Jeffrey V. Nickerson, and Keith D. Swenson.*
  - ✓ REST is about a uniform CRUD interface to resources
  - ✓ WS-* is about domain-specific interfaces to services

LearnQuest

Sanjiva Speaks

- *"REST has clearly given a great architectural foundation for the Web. However, [a] question comes whether REST provides a better foundation for application to application interaction than what WS-* does.*

  People have of course used the Web for application to application integration for many years. But is that really REST or just using the infrastructure of the Web? It's of course the latter: The reality is that it's mostly people shipping XML documents back and forth over HTTP or, in less complex cases, using HTTP GET to send data and get responses back. That's just not REST as there's no well-designed resource structure."
  – Sanjiva Weerawarana (Co-author: WSDL, BPEL4WS, WS-Addressing, WS-RF, WS-Eventing, Apache SOAP, Apache Axis, …)
- Fair Point. We don't get REST just by using HTTP, POX and URLs. REST is an architectural *style*.

Roy Fields The Issues

- *"The fact of the matter is that most CGI scripts are not HTTP compliant. Most CGI scripts, in fact, provide interfaces to applications that suck. [People should realize] that if the CGI script is written such that it behaves like a proper Web interface, then it won't suck. The point is to describe to developers the ways in which an interface can be better implemented on the Web.* ***REST is not the easiest way to provide an interface on the Web. In fact, doing it right requires fairly sophisticated software. CGI makes life easy for programmers. REST-like operation makes life easy for users and content maintainers.***

  The problem with SOAP is that it tries to escape from the Web interface. It deliberately attempts to suck, mostly because it is deliberately trying to supplant CGI-like applications rather than Web-like applications. It is simply a waste of time for folks to say that "HTTP allows this because I've seen it used by this common CGI script." If we thought that sucky CGI scripts were the basis for good Web architectures, then we wouldn't have needed a Gateway Interface to implement them.

  In order for [SOAP] to succeed as a Web protocol, it needs to start behaving like it is part of the Web. That means, among other things, that it should stop trying to encapsulate all sorts of actions under an object-specific interface. It needs to limit its object-specific behavior to those situations in which object-specific behavior is actually desirable. If it does not do so, then it is not using URI as the basis for resource identification, and therefore it is no more part of the Web than SMTP."
  – Roy T. Fielding, Inventor of REST and Principal Author of HTTP 1.1.

Sanjiva Admits WS Flaws and REST Benefits

- *"There are key features of HTTP as a transfer protocol that are very valuable. The current crop of WS-* middleware hasn't done a superb job of figuring out how to maximize using those. If you look at WSDL 2.0 you'll see that we've sorted out the idempotent & safety aspects. The REST folks should look at the HTTP binding of WSDL 2.0 that really does give a very clean way to describe RESTful (and non-RESTful!) HTTP services. WADL gets many of its ideas from WSDL 2.0 and, other than for NIH, I can't think of many reasons why [WADL advocates] wouldn't use and improve WSDL 2.0.*

  A lot of the criticism of WS- comes from people who continue to look at SOAP as a distributed object communication mechanism. SOAP 1.1 certainly had those traits (with SOAP-RPC and SOAP Encoding) but SOAP 1.2 has lost those totally and same for WSDL – 1.1 had RPCness built in but with 2.0 it's not in the least.*

  The real question is whether resource oriented architectures and service oriented architecture are one and the same. I assert that they're not: given a distributed systems problem one can develop solutions using either approach and the artifacts that result would be radically different. True REST applications are resource oriented. WS- is used to implement service oriented architectures. So it's not that one's wrong and the other is right, but rather that they're different."* – Sanjiva
  - ✓ See also: http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327/#http-binding

Is REST Really in Use?

- "*Amazon.com provides both REST and WS-* interfaces, and finds that 85% of developers use the REST interface, whereas only 15% choose the SOAP interface.*" (http://www.oreillynet.com/pub/wlg/3005)
- Yahoo:
  - ✓ http://developer.yahoo.com/search/
    - http://developer.yahoo.com/python/python-rest.html
    - http://developer.yahoo.com/dotnet/howto-rest_cs.html
- Flickr:
  - ✓ Look for REST at http://www.flickr.com/services/api/
- YouTube:
  - ✓ http://www.youtube.com/dev_rest
- Pete Lacey documents that REST is even "totally sweeping Microsoft", its products and development offerings.
  - ✓ http://wanderingbarque.com/nonintersecting/2007/05/14/the-beginning-of-the-end/
- Subversion repositories everywhere!

But Are They Really REST?

- Again, a fair question, reflecting back not only on Sanjiva's and Roy's comments, but also on others.
  - ✓ "Non-RESTful POX services are more accessible than SOAP services, but they don't exhibit the desirable characteristics associated with RESTful resources." – Anne Thomas Manes, *How NOT to do RESTful Web Services* (http://atmanes.blogspot.com/2007/06/how-not-to-do-restful-web-services.html)
    - Previously, Manes has been a pro-WS-* analyst (Pete Lacey is the pro-REST analyst) at the Burton Group.
  - ✓ Analysis of aforementioned interfaces is left as an exercise for the reader, but let's examine the principles of REST.
  - ✓ As previously noted, Roy has expressed concern about there being a lot of incorrect information about REST being disseminated, even by those who claim to understand it. Please try to make sure that whatever *you* are doing matches the concepts presented here, particularly those directly quoting Roy. If they don't, find out why not.

LearnQuest

Coding Your REST Service

- Java
  - ✓ JAX-RS was developed to provide a Java API for RESTful Web services.
  - ✓ Spring Web MVC and Spring WebFlux
  - ✓ Java Servlets – one problem that we have to work around is that the Servlet API conflates request body parameters with those in the query string. Remember: those must be treated differently!
  - ✓ AXIS2: http://ws.apache.org/axis2/0_94/rest-ws.html
  - ✓ "Restlet": http://www.restlet.org/
- JavaScript (NodeJS + Express)
- Microsoft.NET
- Python
  - ✓ http://flask.pocoo.org/
  - ✓ http://webpy.org/
  - ✓ http://www.cherrypy.org/
- It's HTTP! Roll your own.
- And the beat goes on …

LearnQuest

Summary

- In this module, participants learned the following:
  - ✓ What REST is
  - ✓ Why it is called Representational State Transfer
  - ✓ Understanding why REST is an Architectural Style and not a standard

LearnQuest

48

Lab Exercise

- Lab: Design a REST Service
  - ✓ *A Public Library desires to expose its catalog of borrowable movies using REST. Movies have titles, ratings, actors, genre. For each movie, they have physical copies in DVD, BluRay and BluRay 4K. Each physical copy identifies the customer who has borrowed it, and the date is is due back (and thus expected to be available upon).*
  - ✓ *Consumers may use the service to view the catalog, borrow a movie (to be picked up at the library), and add themselves to a wait list for the movie*
  - ✓ *Librarians may use the service to manage the catalog.*
  - ✓ *For this lab exercise, design:*
    - *Hierarchy of resources, expressed in URL format*
    - *List of properties for each resource*
    - *Operations to support on each resource*

LearnQuest

49

Questions

LearnQuest

50

THE QUEST FOR KNOWLEDGE IS A LIFELONG JOURNEY

Module 3

Designing REST Web Services

Objectives

- In this module, participants will learn the following:
  - ✓ Designing REST Web Services
  - ✓ Examples of Services Designed in a RESTful Style
  - ✓ Logical URLs vs. Physical URLs
  - ✓ REST Web Services Characteristics

LearnQuest

2

Designing REST Web Services

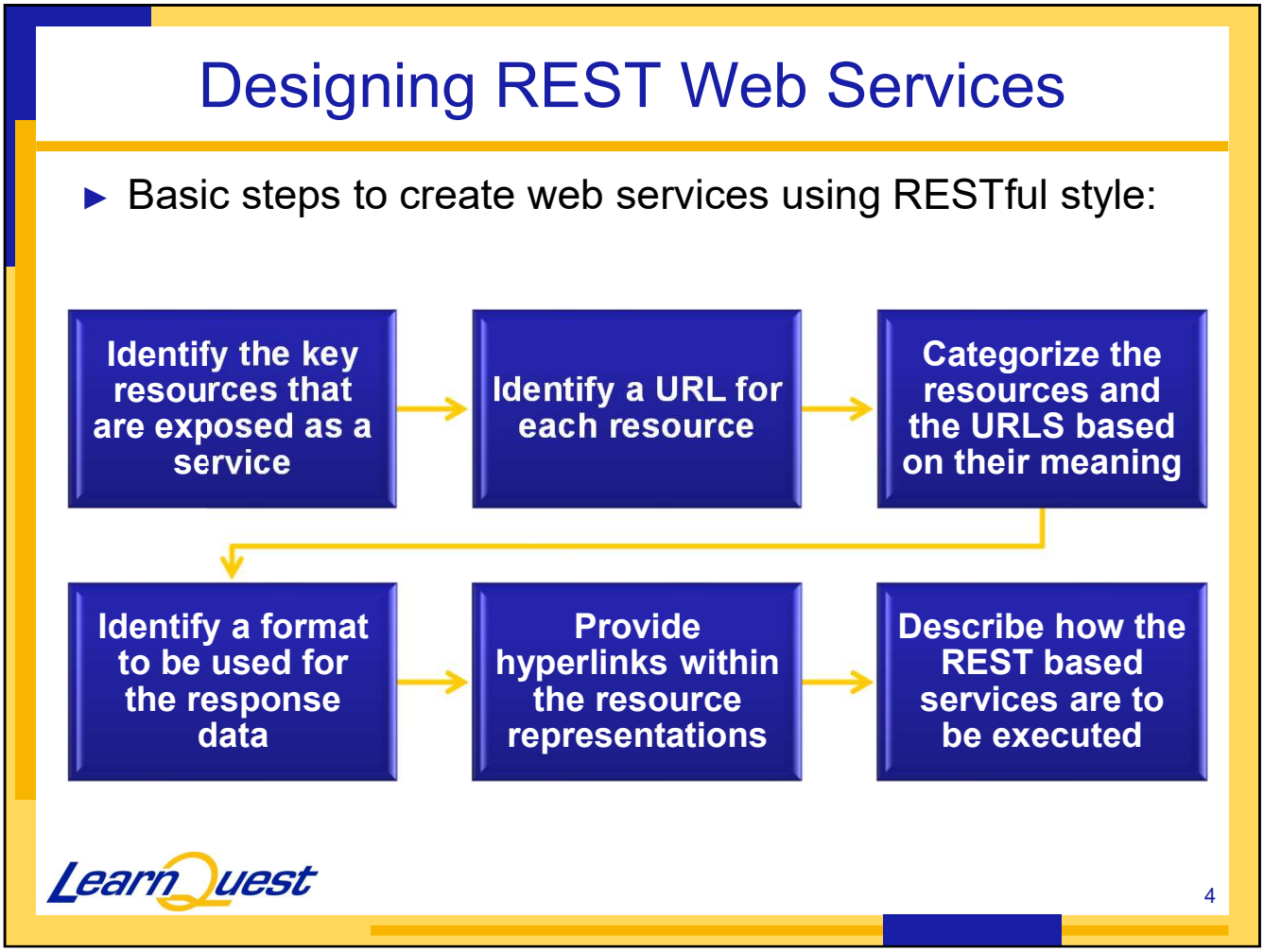
- Classic REST system:
  - ✓ The Web is a REST system.
  - ✓ REST focuses on the overall design, not the implementation details.

LearnQuest

3

The Web is a REST system. Typical examples of web based systems are online ordering services, search services, and online dictionary services.

REST focuses on the complete architectural design of the web-based system, not only the implementation details. Many developers have been using REST to build RESTful services and did not even realize it!



1. Identify the key resources that are exposed as a web-based service.
    - For example: List of training classes and class details
2. Identify a URL for each resource.
    - For example:
        - http://www.learnquest.com/classes
        - http://www.learnquest.com/classes/classid
        - http://www.learnquest.com/classDetails
3. Categorize the resources and the URLS based on their meaning.
    - Enable the client to view only the resource.
        - Utilization through HTTP, GET
    - Enable the client to update, delete, and add to a resource.
        - Utilization through HTTP, PUT, POST, and DELETE

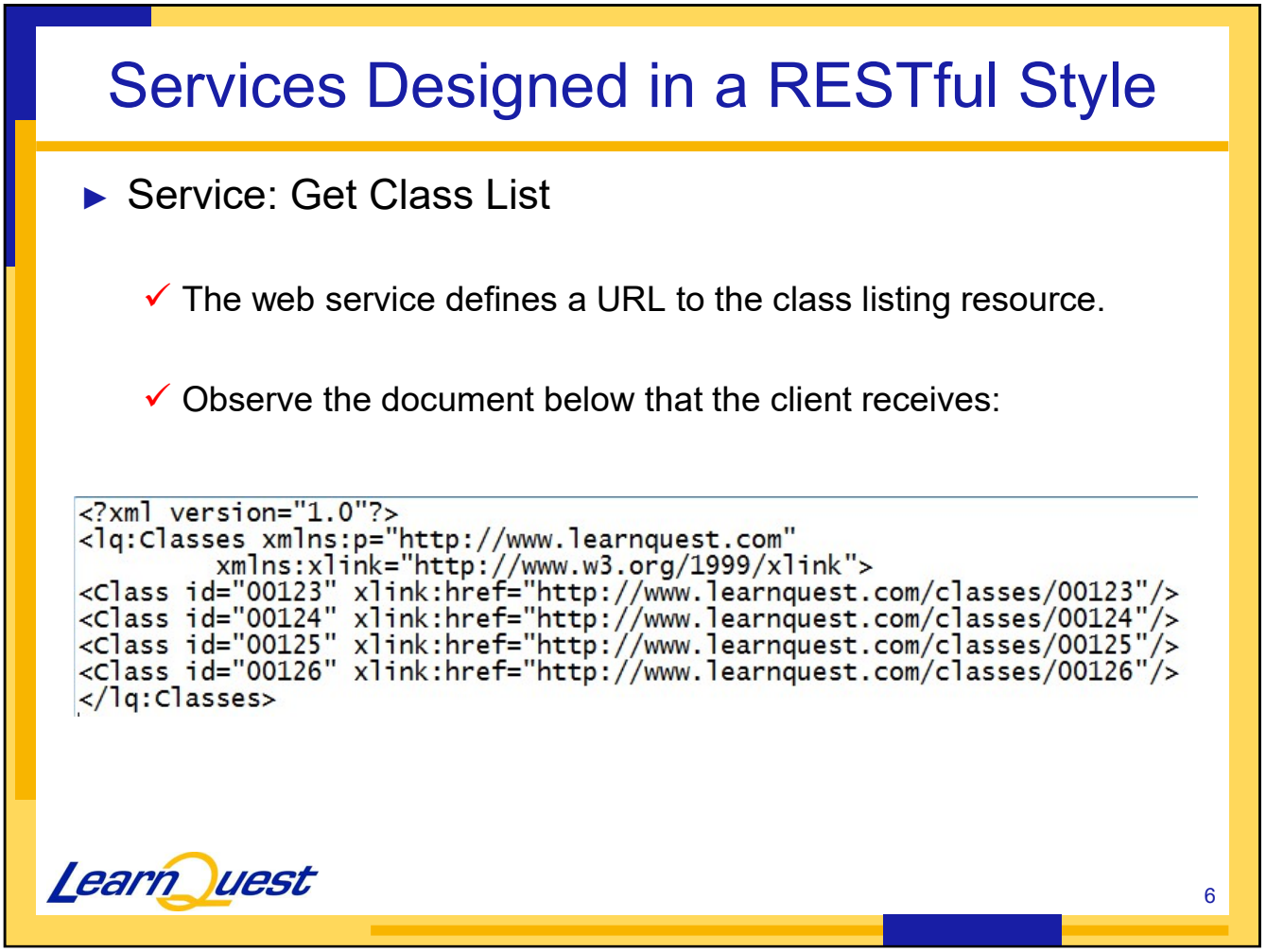
Services Designed in a RESTful Style

- LearnQuest Web Services deployed some web services to enable its customers to:
  - ✓ Get a list of training classes
  - ✓ Get detailed information about a particular class
  - ✓ Submit class registration requests

LearnQuest

5

The LearnQuest web service is made available through a URL, the class list resource. For example, a client would use this URL to get the class list: http://www.learnquest.com/classes. Note that "how" the web service generates the class list is completely transparent to the client. The implementation is transparent to clients; Learnquest.com can modify the implementation of this resource without impacting clients.



For example, a client would use this URL to get the class list: http://www.learnquest.com/classes.

Observe the reference list of classes that can be used to retrieve the information about the different class offerings. The client's initial request triggers the current state to change to another state when a specific URL is selected from the list in the response document.

Services Designed in a RESTful Style

- Service: Get Detailed Class information
  - ✓ The web service makes available a URL to each class resource.
  - ✓ Observe the document below that the client receives:

```
<?xml version="1.0"?>
<lq:Classes xmlns:p="http://www.learnquest.com"
         xmlns:xlink="http://www.w3.org/1999/xlink">
      <Class-ID>00123</Class-ID>
      <Name>Fundamentals of Java</Name>
      <Description>Learn programming using the Java programming language</Description>
      <Specification xlink:href="http://www.learnquest.com/classess/00123/specification"/>
      <Duration>10</Duration>
</l1:Classes>
```

LearnQuest

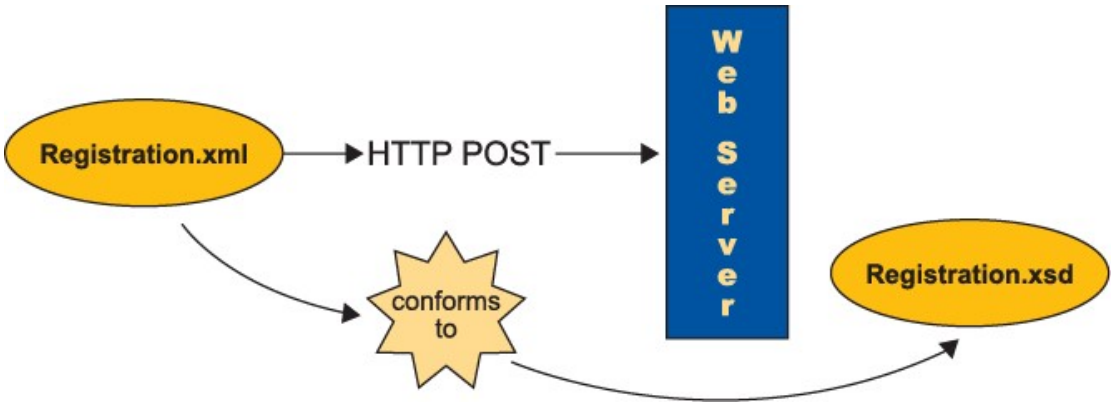
7

For example, here's how a client requests class 00123:
http://www.learnquest.com/classess/00123.

The specification for this class is exposed through the hyperlink. Each response document allows the client to take a deeper dive to collect more detailed information about a class topic.

Services Designed in a RESTful Style

- Service: Class Registration Submission
  - ✓ The web service makes available a URL to submit a class registration request.



LearnQuest

8

The web service makes available a URL to submit a class registration request. The client creates a registration instance document that conforms to the registration schema that LearnQuest has designed. The client submits registration.xml as part of the request using an HTTP POST. Next, the registration service responds to the HTTP POST with a URL to the submitted registration request.

Logical URLs versus Physical URLs

- A resource is a conceptual entity.
  - ✓ This URL: http://www.learnquest.com/classess/00123 is a logical URL, not a physical URL.
  - ✓ URLs should not expose the implementation technique used.
  - ✓ Enables the ability to change implementation details without impacting clients or having misleading URLs

LearnQuest

9

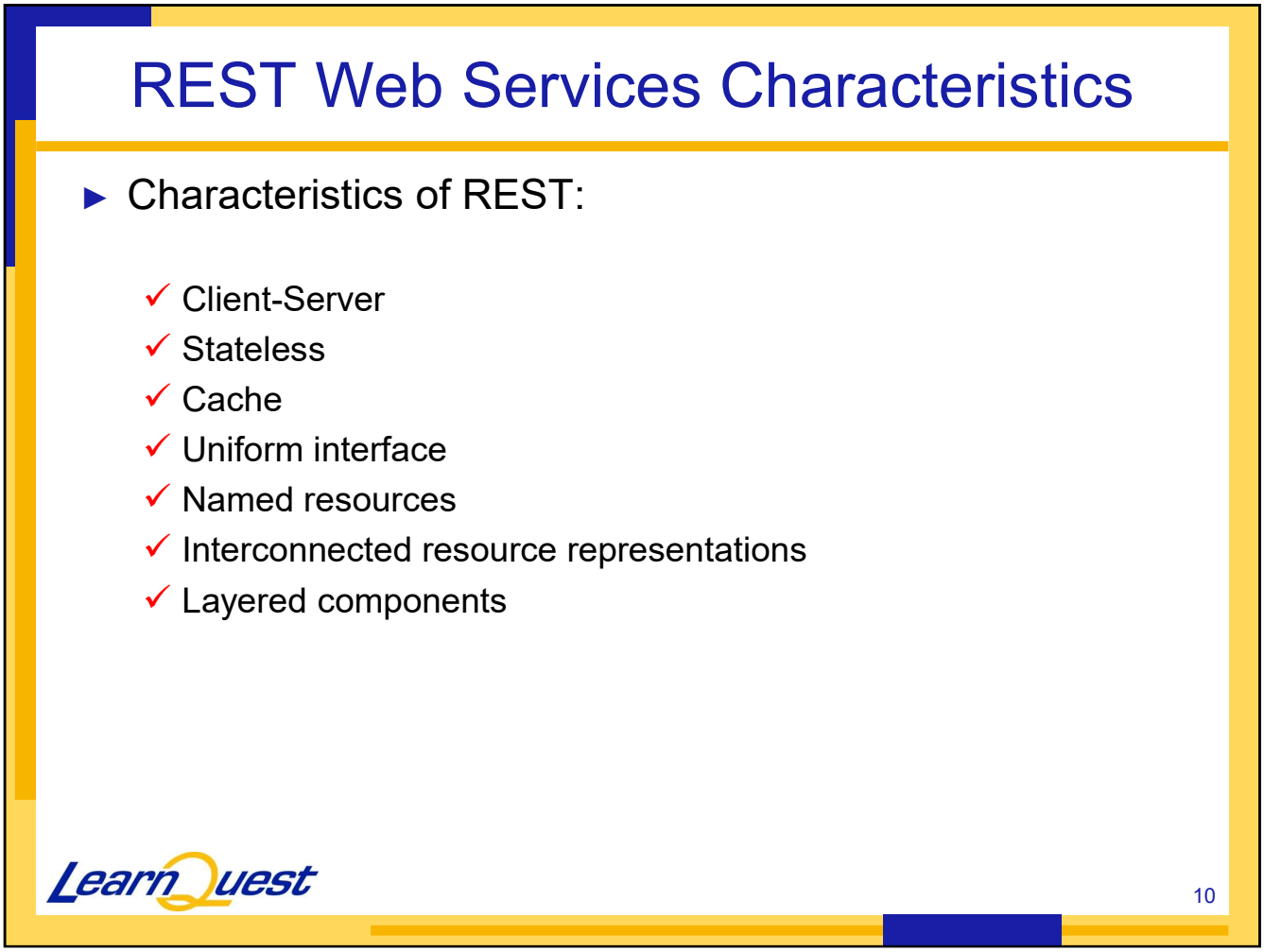
Learnquest.com will store all class information in a database. Implementation at the LearnQuest website will receive each logical URL request, parse it to determine which class is being requested, query the database, and generate the part response document that is returned to the client.

- http://www.learnquest.com/classess/000123
- http://www.learnquest/classess/000124

Contrast the above logical URLs with these physical URLs:

- http://www.learnquest.com/classess/000123.html
- http://www.learnquest/classess/000124.html

The above URLs are pointing to physical (HTML) pages.



Client-Server: An extension of the traditional client-server architecture that enables a pull-based interaction style.

Stateless: A request from client to server must contain required information to understand the request and does not have the ability to take advantage of any stored context on the server.

Cache: Improves network efficiency, and responses must be capable of being labeled as cacheable or non-cacheable.

Uniform interface: Resources are accessed with HTTP methods (e.g., GET, POST, PUT, DELETE).

Named resources: System resources are accessible from a given name using a URL.

Interconnected resource representations: Resources exposed in a variety of different formats are interconnected by using URLs. This enables a client to progress from one state to another.

Layered components: Proxy servers, cache server, and gateways are used to interface between clients and resources to support performance and security capabilities.

JAX-RS

- **JAX-RS** stands for *Java API for RESTful Web Services*
  - ✓ Used to create/consume services according to REST style
- Over the next several slides, we will look at a simple REST implementation using Java's JAX-RS API.
- Understanding of Java is optional; we will not delve into the details, but will focus on the concepts, highlighting parts of the code that implement some aspect of REST behavior.
- JAX-RS is not the only means to implement REST in Java. In our lab, we will briefly review the source for a different example, and spend time exercising it with a generic tool.

LearnQuest

REST: Representational State Transfer is a concept in which the request URL/URI represents a resource in a particular state.

The specification of JAX-RS is defined within the Java EE 6 API and javax.ws package.

Converting a Representation <-> Java

- **JAXB** stands for *Java Architecture for XML Binding*:
  - ✓ Makes it easier to read and write XML documents
  - ✓ You can create an XML schema, and JAXB can generate Java classes from it.
  - ✓ Used by many other Java APIs, including JAX-RS to convert between wire level representation and Java objects.
  - ✓ Because there is a well-defined mapping between XML and JSON, several JAXB implementations automatically support JSON.

LearnQuest

12

Objects of Java classes can be marshaled to XML, or XML files can be un-marshaled to Java objects. The JAXB API is defined within Java SE 6 specification and defined in package javax.xml.bind.

JAXB Annotation

- The @XMLRootElement annotation on the class name declares it as an JAXB entity:
  - ✓ Provides the ability to explicitly configure JAXB metadata for a class to be converted.
    - JAXB applies defaults, so this annotation is not always necessary.
  - ✓ This tells a serialization mechanism to marshal and un-marshal the properties of the entity
    - Using XML as the serialization format
  - ✓ The annotations can drive the generation of XML Schemas
    - Schemas may be used to generate JAXB annotated Java classes

LearnQuest

13

The @XMLRootElement annotation on the class name declares it as an JAXB entity. Using this annotation allows for the property values of the Java entity to be converted to XML format. Fields and methods can be annotated within the class to customize the serialization.

JAXB Annotation

- Observe the following code sample of a entity class that uses JAXB annotation.
  - ✓ @XMLRootElement

```
@XmlRootElement
public class Employee {

    private String firstName;
    private String lastName;
    private int id;
```

LearnQuest

14

Define the root element for the XML file that will be produced when using the @XmlRootElement JAXB annotation. The name of the root XML element is derived from the class name.

Creating a RESTful Root Resource Class

- **Root resource classes**
  - ✓ POJOs that are either annotated with @Path, or have at least one method annotated with @Path, or a **request method designator**, such as @GET, @PUT,@POST, or @DELETE
- **Resource methods**
  - ✓ Methods of a resource class annotated with a request method

LearnQuest

15

Creating a RESTful Root Resource Class

- Observe the following code sample of a resource class that uses JAX-RS annotations.

```
@Path("/customers")
public class CustomerResource {
    private Map<Integer, Customer> customerDB =
            new ConcurrentHashMap<Integer, Customer>();
    private AtomicInteger idCounter = new AtomicInteger();
    private OutputStream os;

    @GET
    @Path("{id}")
    @Produces("application/xml")
    public StreamingOutput getCustomer(@PathParam("id") int id) {
        final Customer customer = customerDB.get(id);
        if (customer == null) {
            throw new WebApplicationException(Response.Status.NOT_FOUND);
        }
        return new StreamingOutput() {
            public void write(OutputStream outputStream) throws IOException,
                    WebApplicationException {
                outputCustomer(outputStream, customer);
            }
        };
    }
```

LearnQuest

16

- The @Path annotation's value is a relative URI path. In the preceding example, the Java class will be hosted at the URI path /helloworld. Variables can be contained in the URIs.
- The JAX-RS @GET annotation is a request method, which has corresponding named HTTP method options: @POST, @PUT, @DELETE, and @HEAD.
- The @Produces annotation specifies the MIME media types that a resource can produce and send back to the client.
- The @Consumes annotation specifies the MIME media types that a resource can consume when sent by the client.

HTTP_GET Request Method

- Code sample of a resource class that uses JAX-RS @GET annotation:

```
@GET
@Path("{id}")
@Produces("application/xml")
public StreamingOutput getCustomer(@PathParam("id") int id) {
    final Customer customer = customerDB.get(id);
    if (customer == null) {
        throw new WebApplicationException(Response.Status.NOT_FOUND);
    }
    return new StreamingOutput() {
        public void write(OutputStream outputStream) throws IOException,
                WebApplicationException {
            outputCustomer(outputStream, customer);
        }
    };
}
```

LearnQuest

17

The @Path annotation's value is a relative URI path. In the preceding example, the Java class was hosted at the URI path /customers. Variables can be contained in the URIs. This example shows that a path parameter is expected that will carry the ID of the customer to be retrieved.

Note the use of the @PathParam annotation. This directs JAX-RS to marshal the ID from the content and inject it in the method parameter at runtime.

The URI to retrieve the customer that has an ID of "1" would be:

GET /customers/1

HTTP_POST Request Method

- Code sample of a resource class that uses JAX-RS @POST annotation:

```
@Path("/customers")
public class CustomerResource {

    private Map<Integer, Customer> customerDB =
            new ConcurrentHashMap<Integer, Customer>();
    private AtomicInteger idCounter = new AtomicInteger();
    private OutputStream os;

    @POST
    @Consumes("application/xml")
    public Response createCustomer(InputStream is) {
        Customer customer = readCustomer(is);
        customer.setId(idCounter.incrementAndGet());
        customerDB.put(customer.getId(), customer);
        System.out.println("Created customer " + customer.getId());
        return Response.created(URI.create("/customers/" + customer.getId()))
                .build();
    }
```

LearnQuest

18

The post annotation is used to create resources. Content to add is part of the body of the request. The @Consumes annotation tells us that this method will extract data from an XML document.

The @Path annotation's value is a relative URI path. In the preceding example, the Java class will be hosted at the URI path /customers. Variables can be contained in the URIs.

The URL for this request would be as follows:

POST /customers

The HTTP Status code for this request would be a 201 (Created), and a Location: header will be returned to the client containing the URI of the newly created subordinate resource.

HTTP_PUT Request Method

- Code sample of a resource class that uses JAX-RS @PUT annotation:

```
@PUT
@Path("{id}")
@Consumes("application/xml")
public void updateCustomer(@PathParam("id") int id, InputStream is) {
    Customer update = readCustomer(is);
    Customer current = customerDB.get(id);
    if (current == null)
        throw new WebApplicationException(Response.Status.NOT_FOUND);

    current.setFirstName(update.getFirstName());
    current.setLastName(update.getLastName());
    current.setStreet(update.getStreet());
    current.setState(update.getState());
    current.setZip(update.getZip());
    current.setCountry(update.getCountry());
}
```

LearnQuest

19

The put annotation is used to update resources. Content to update is part of the body of the request. The @Consumes annotation tells us that this method will extract the needed data from an XML document.

Note the similarities to a GET. The @PathParam annotation is used to “tell” our application which customer is targeted for the update.

The URI for this method would be:

PUT /customers/1

The HTTP Status code for this request would be a 204 (No Content), not a 200 as you might have expected.

HTTP_DELETE Request Method

- Code sample of a resource class that uses JAX-RS @DELETE annotation:

```
@DELETE
@Path("{id}")
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public void remove(@PathParam("id") int id) {
    customerDB.remove(id);
}
```

LearnQuest

20

The @Delete annotation is used to remove resources. Note the use of the MediaType enum to specify what mime types will be generated by the response.

A DELETE is similar to a GET. The @PathParam annotation is used to "tell" our application which customer is targeted for the deletion.

The URI for this method would be:

DELETE /customers/1

The HTTP Status code for this request would be a 204 (No Content), not a 200 as you might have expected.

HTTP_HEAD Request Method

- Code sample of a resource class that uses JAX-RS @HEAD annotation:

```
@HEAD
public void getHeaders(@Context HttpHeaders ui) {

    MultivaluedMap<String, String> headerParams = ui.getRequestHeaders();
    Map<String, Cookie> pathParams = ui.getCookies();

    // review and log headers, examine cookies, etc...
}
```

LearnQuest

21

The @Head annotation is used to return the start line and header for the HTTP frame. No body content is provided.

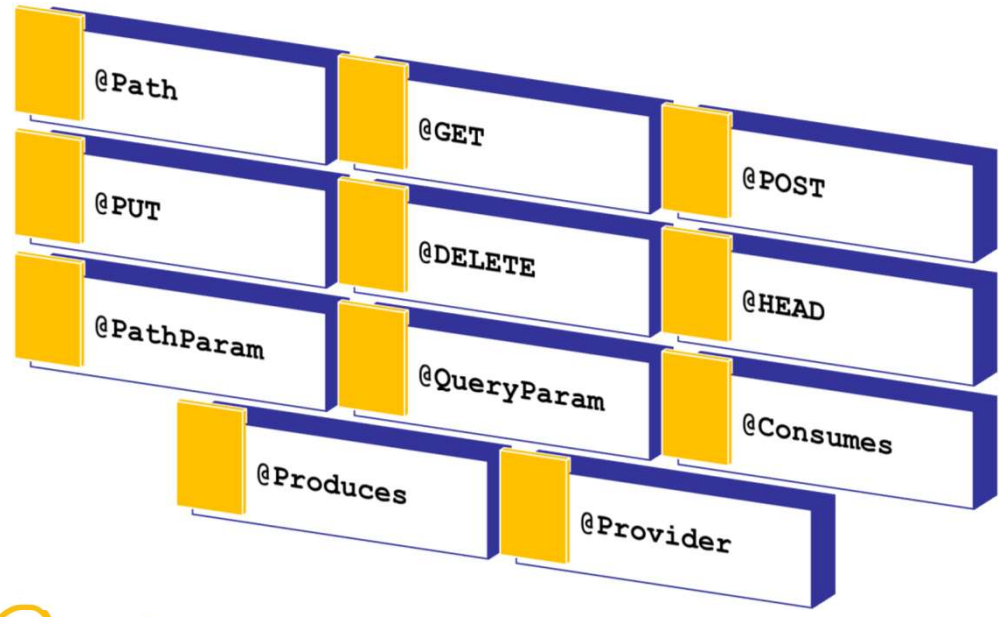
Note the use of the @Context annotation to return the context object for the application.

The URI for this method would be:

HEAD /customers

Summary of JAX-RS Annotations

- Annotate Java resource classes with JAX-RS to create RESTful web services.



LearnQuest

22

JAX-RS is a Java programming language API designed to develop applications that use the REST architecture.

The JAX-RS API uses Java programming language annotations to ease the development of RESTful web services. Developers enhance the Java programming language class files with JAX-RS annotations to define resources. JAX-RS annotations are used as runtime annotations. This allows runtime reflection to generate the class code automatically and additional artifacts for the resource. At deployment time, a Java EE application contains JAX-RS resource classes, the helper classes, and additional artifacts. The resources are exposed to clients through the Java EE server.

Hyper-Text Application Language - HAL

- REST is largely bereft of standards regarding the content of representations.
- There are myriad ways in which one could define content and relationships between resources/sub-resources.
- HAL represents one approach to providing a common convention for defining self-describing hypermedia.
  - ✓ *"Instead of using ad-hoc structures, or spending valuable time designing your own format; you can adopt HAL's conventions and focus on building and documenting the data and transitions that make up your API."*
- MIME Type: application/hal+json or application/hal+xml

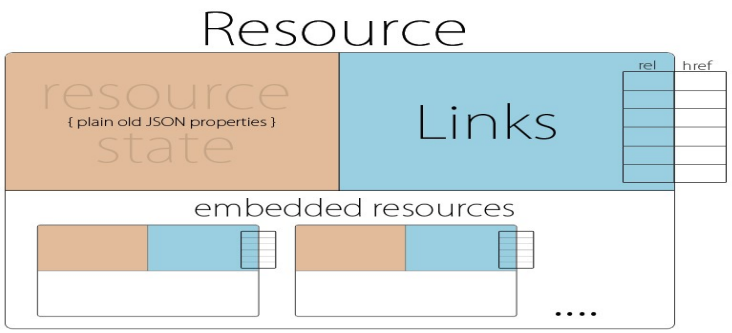
LearnQuest

23

The HAL Specification is http://stateless.co/hal_specification.html, with an Internet Draft (proposed standard) at https://tools.ietf.org/html/draft-kelly-json-hal-08

Hyper-Text Application Language - HAL

- HAL representations consist of
  - ✓ State (the properties of your resource)
  - ✓ Embedded (as opposed to linked) resources
  - ✓ Linked resources
    - Target URI
    - A name for the relationship
    - Optional properties, *e.g.*, content-type



Summary

- In this module, participants learned the following:
  - ✓ Design of REST Web Services
  - ✓ Examples of Services Designed in a RESTful Style
  - ✓ Logical URLs vs. Physical URLs
  - ✓ REST Web Services Characteristics

LearnQuest

25

Lab Exercise

- Lab: Interact with a RESTful service
  - ✓ With your Instructor, review the Java Source for a provided REST service. There is very little code, since the bulk of the work is actually done for us by a full-featured runtime, with which we will spend a lot of time later in the curricula.
  - ✓ The runtime provides a REST interface using HAL.
  - ✓ *Using POSTMAN, interact with the service, exercise all of the CRUD operations, and seeing the HTTP interactions.*

LearnQuest

26

Questions

LearnQuest

27

THE QUEST FOR KNOWLEDGE IS A LIFELONG JOURNEY

Module 4

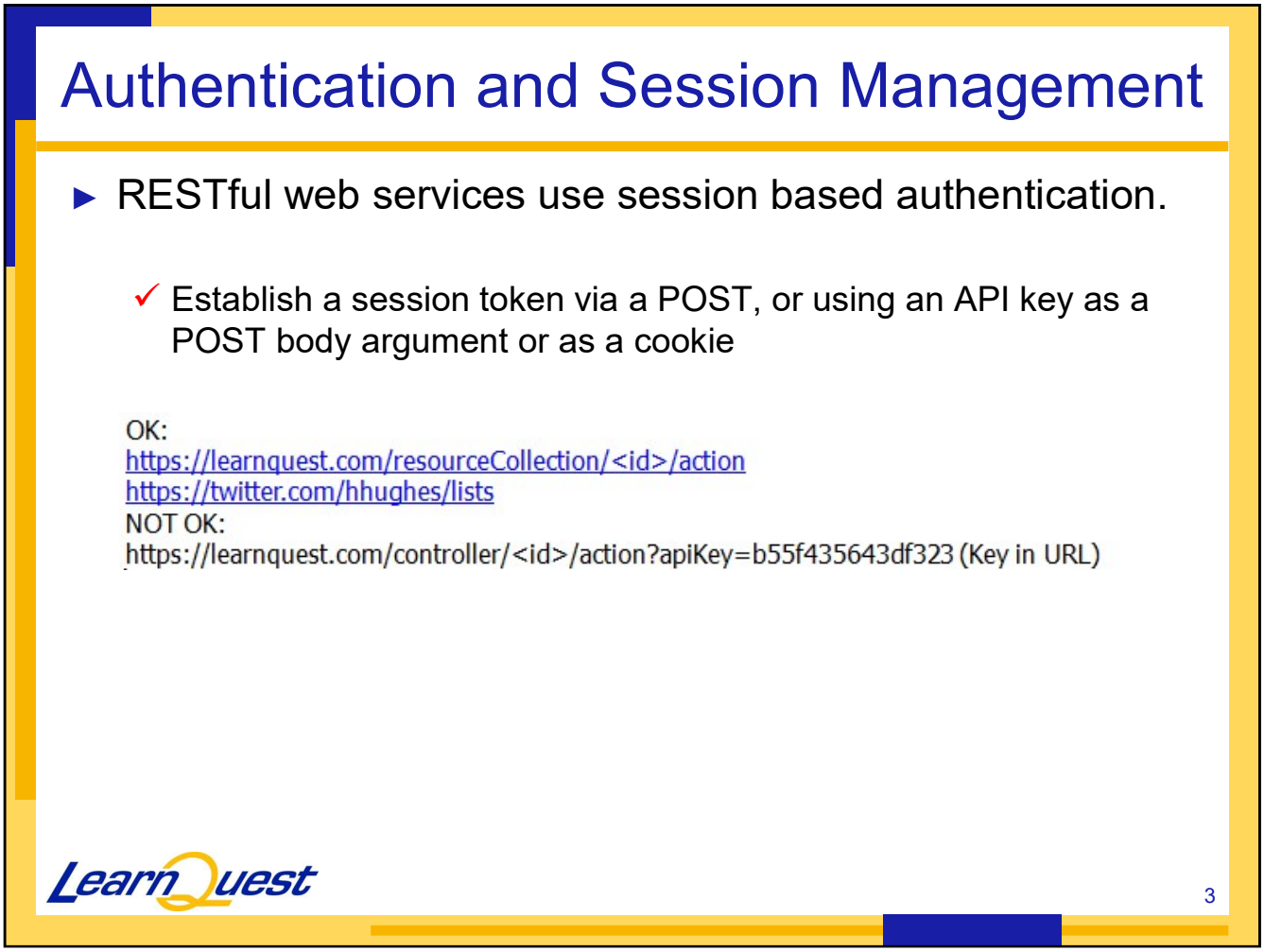
Security, SOAP, and REST

Objectives

- In this module, participants will learn the following:
  - ✓ REST Security and Validation
  - ✓ Compare SOAP and REST
  - ✓ Compare WSDL and WADL

LearnQuest

2



It is bad practice to include usernames and passwords, session tokens, and API keys within the URL. If authentication information is provided within the URL, the authentication information can be captured in web server logs and makes them intrinsically vulnerable.

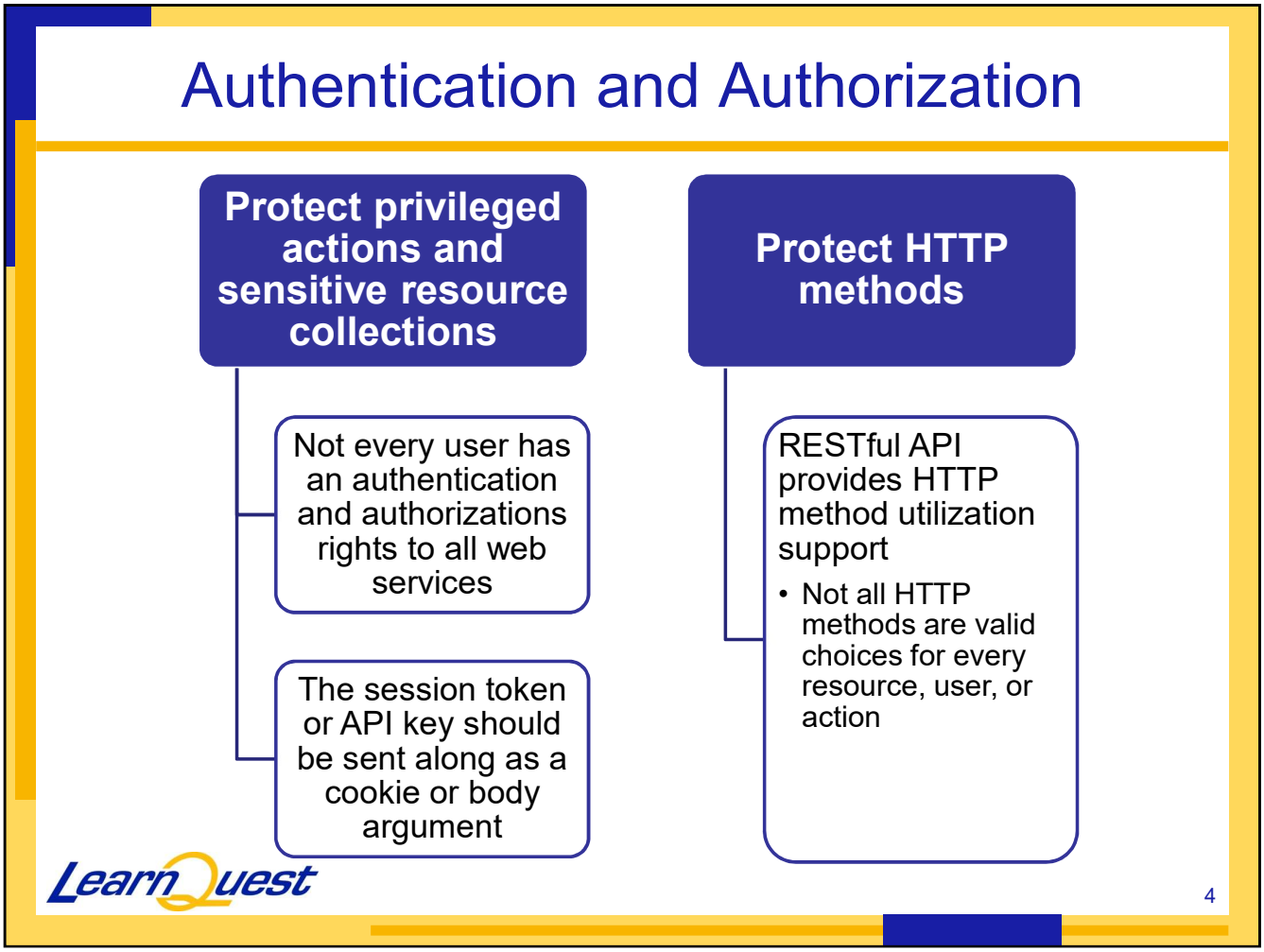
OK:

https://learnquest.com/resourceCollection/<id>/action

https://twitter.com/hhughes/lists

NOT OK:

https://learnquest.com/controller/<id>/action?apiKey=b55f435643df323 (Key in URL)

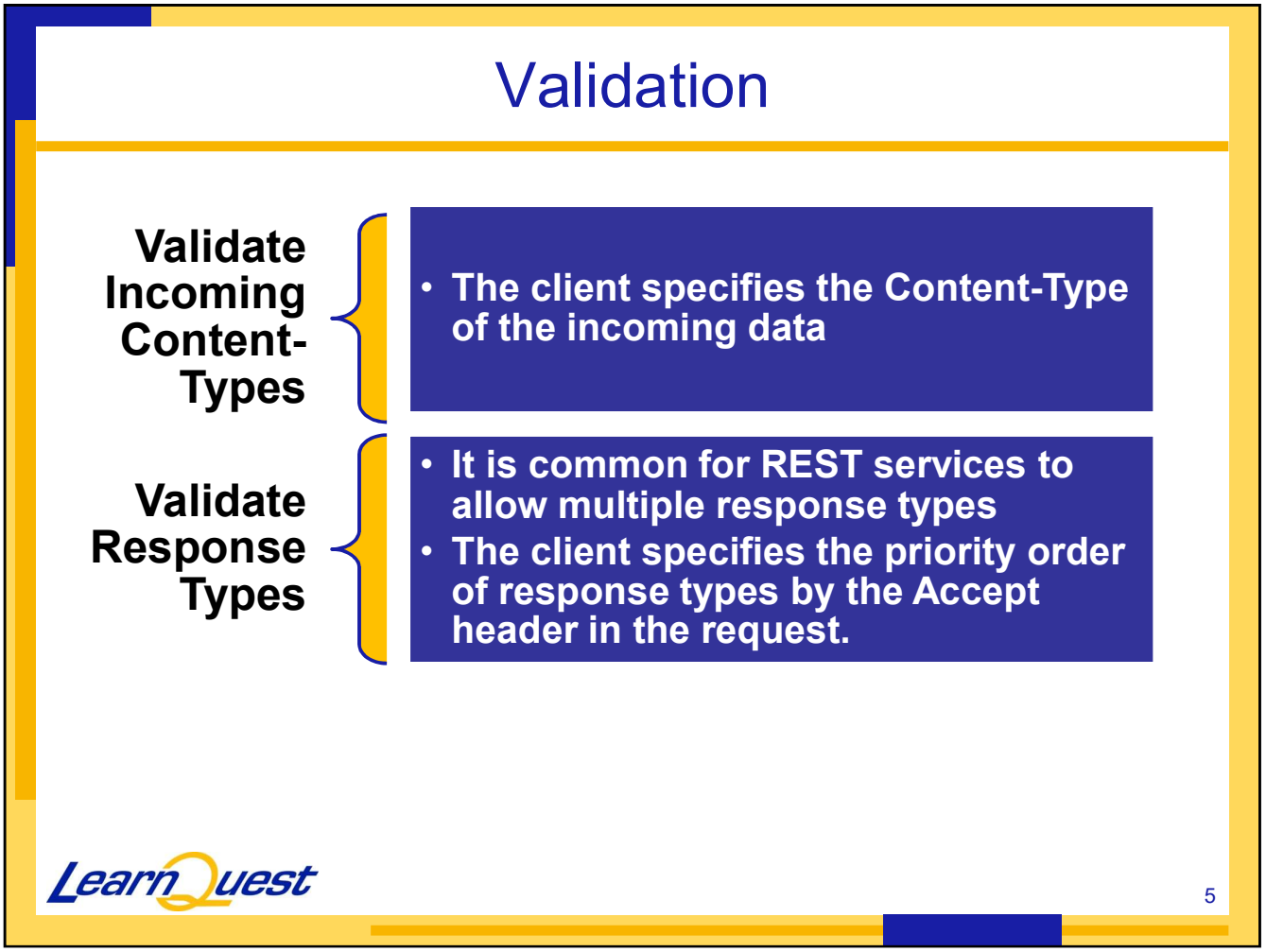


Protect privileged actions and sensitive resource collections.

Web service authentication and authorization rights should not be given to every user. It is proper practice to send the session token or API key as a cookie or body argument to ensure that privileges or actions are properly granted from unauthorized use.

Protect HTTP methods.

RESTful API provides HTTP method utilization support, such as GET (read), POST (create), PUT (replace/update), and DELETE (to delete a record). Not all HTTP methods are valid choices for every resource, user, or action. All incoming requests should be validated through a session token , using an API key or a cookie for a session based authentication with any POST method invocation.



Validate Incoming Content-Types.

The client specifies the Content-Type (e.g., application/xml or application/json) of the incoming data when it is sent as a post or put method. The client should always validate that the Content-Type header and the content is of the same type.

Validate Response Types.

It is a best practice for REST services to allow multiple response types (e.g., application/xml or application/json). The client can specify the priority order of response types by the Accept header in the request. A request can be rejected if the Accept header does not specifically contain one of the allowable types.

SOAP/HTTP Standards and APIs

- **SOAP** provides the definition of the XML-based information that can be used for exchanging information between peers in a distributed environment.
- It relies on:
  - ✓ XML for its message format
  - ✓ HTTP (for WS-I compliance) for message transmission

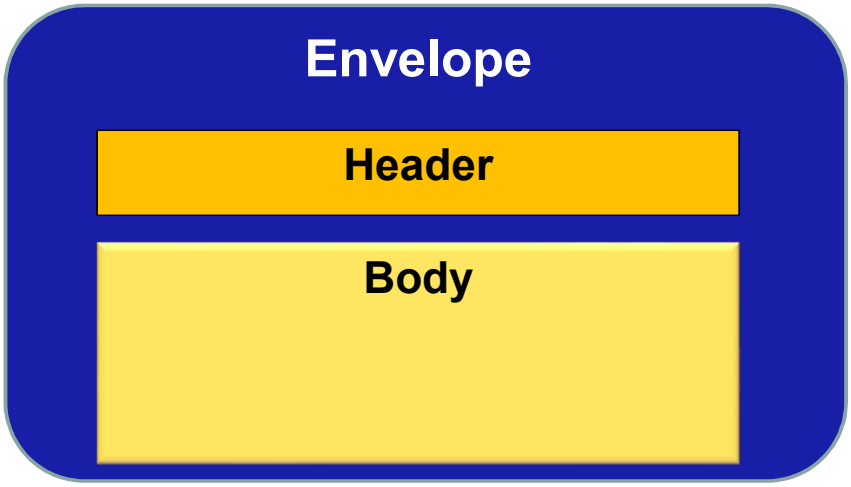
LearnQuest

6

A SOAP message is sent to a web service, usually in the body of an HTTP request. The Web Service responds either with a SOAP message or a SOAP fault. Regardless of the response content, the receiving application is expected to parse the response and translate it for its own purposes.

SOAP/HTTP Standards and APIs

- A SOAP message consists of an envelope, which encloses an optional **header** and a required **body**.



LearnQuest

7

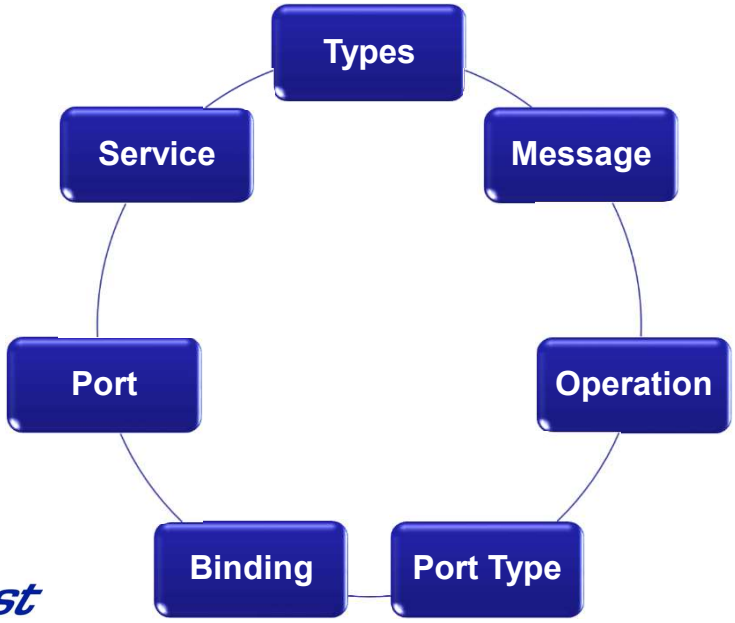
The message sent from a client contains instructions on which function to call and what arguments to provide. The message returned to a client may contain data returned from the function call.

Header: optional section to add information not part of the payload

Body: mandatory section that contains the main payload

SOAP/HTTP Standards and APIs

- **WSDL** provides the ability to describe communications in a structured way (i.e. as an XML document). It consists of the following seven sections:



LearnQuest

8

Types– a container for data type definitions

Message–definition of the data being communicated

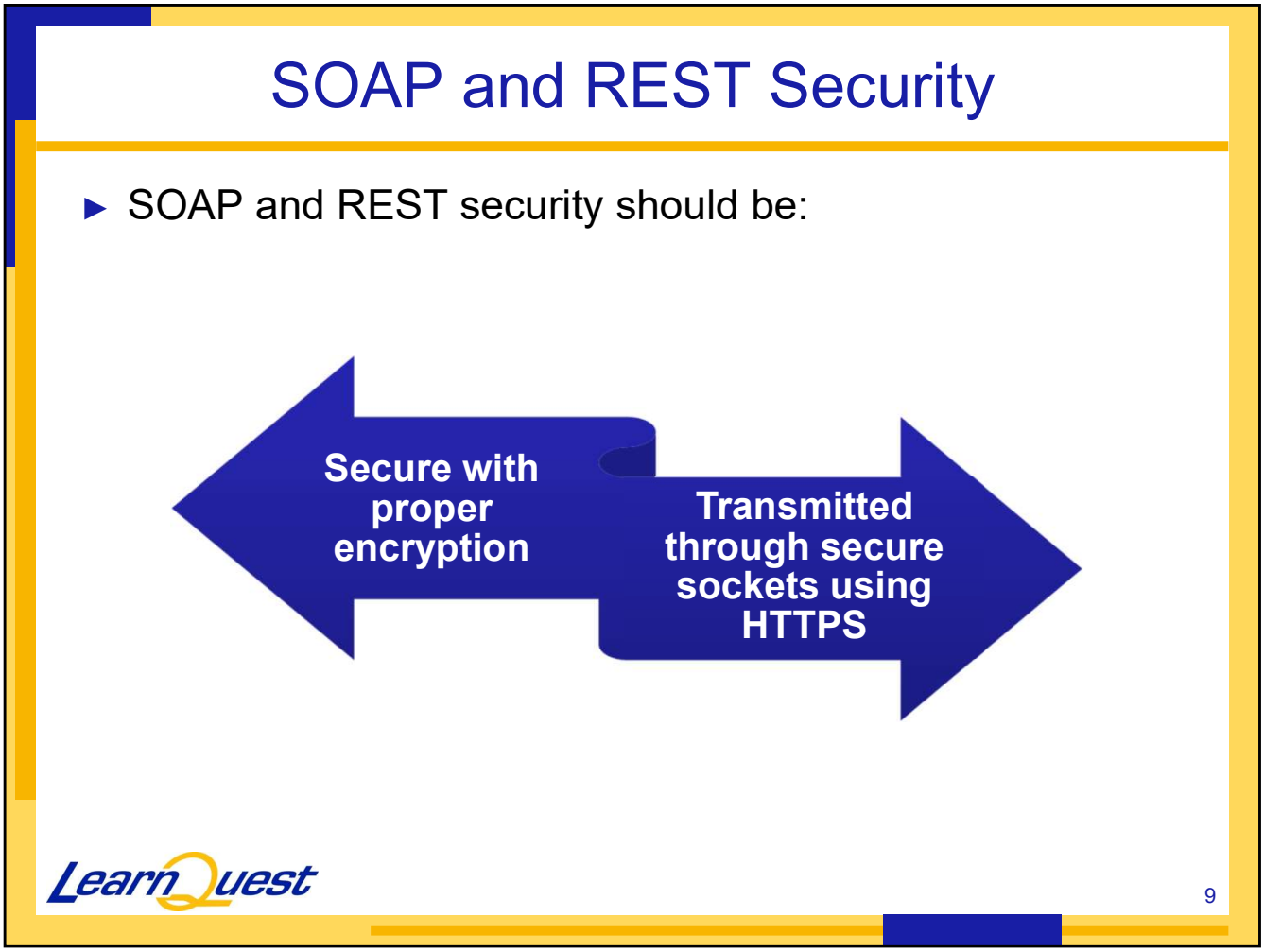
Operation–description of an action supported by the service

Port Type–set of operations supported by endpoints

Binding– a concrete protocol and data format specification for a particular port type

Port– a single endpoint defined as a combination of a binding and a network address

Service– a collection of related endpoints



Security for both SOAP and REST can be transmitted through secure sockets using HTTPS. They are also equally secure with proper encryption, and the content can be encrypted using any mechanism.

Web Services and SOAP

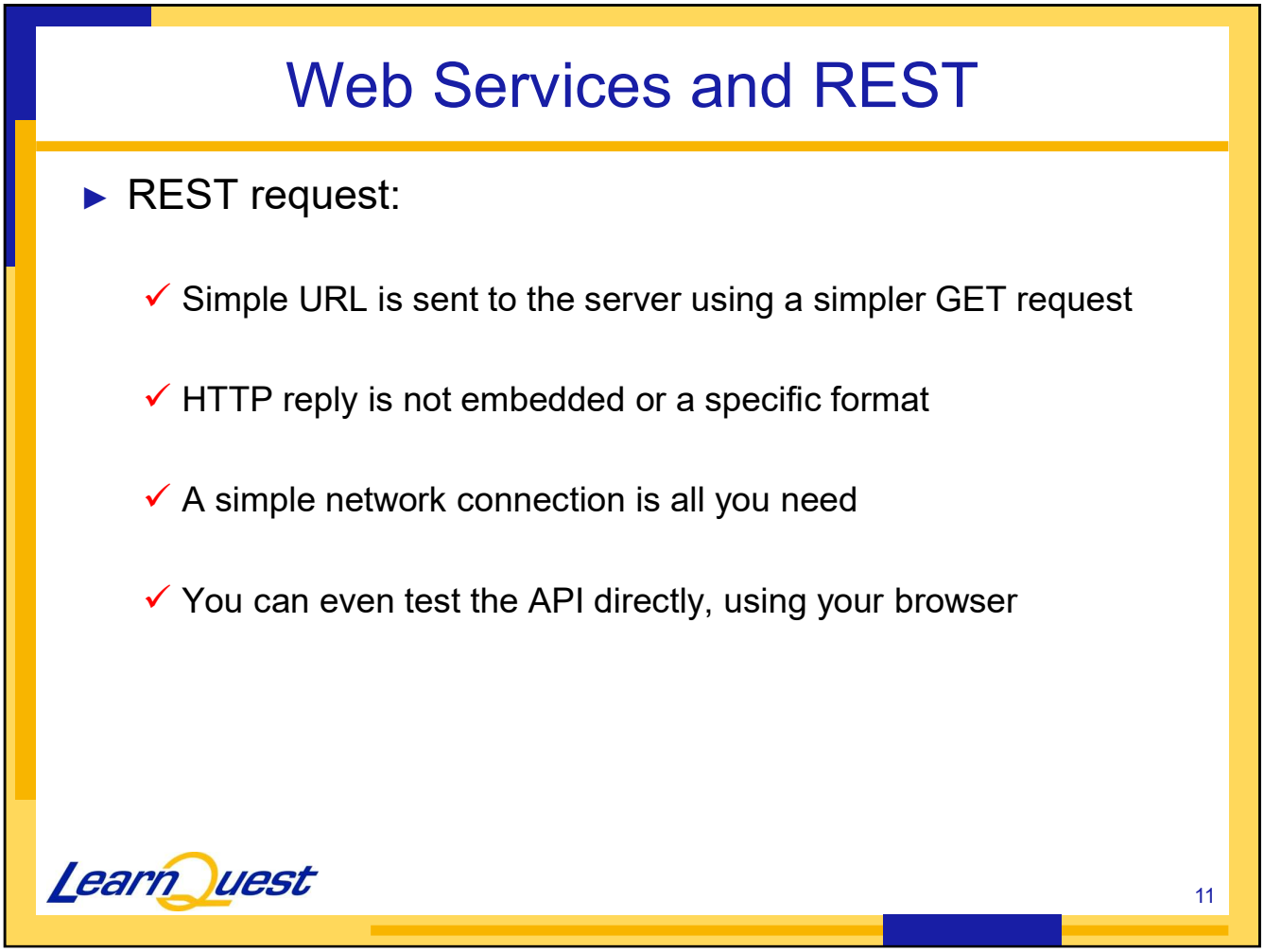
- SOAP request:
  - ✓ The entire formal SOAP envelope has to be sent (using an HTTP POST request) to the server.
  - ✓ Typically, the result is an XML file, but it will be embedded as the "payload" inside a SOAP response envelope.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
 <soap:body pb="http://www.acme.com/phonebook">
  <pb:GetUserDetails>
   <pb:UserID>12345</pb:UserID>
  </pb:GetUserDetails>
 </soap:Body>
</soap:Envelope>
```

LearnQuest

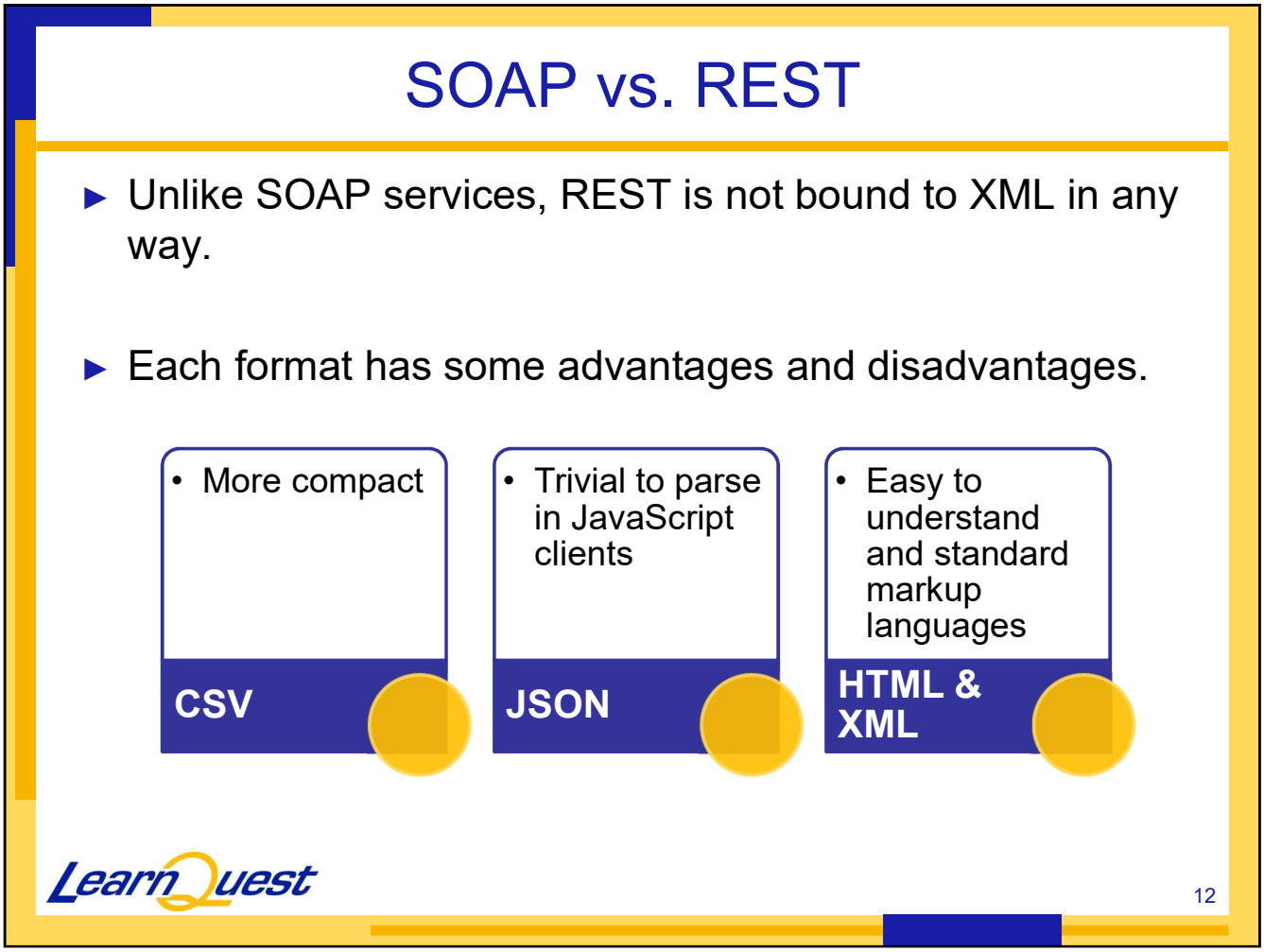
10

Web Services are often used with libraries that create the SOAP/HTTP request and send it over and then parse the SOAP response.



It is a common convention in REST design to use nouns rather than verbs to denote simple resources. REST can handle single or multiple parameter requests. In most cases, you'll just use HTTP GET parameters in the URL. You use a GET request to read data and a POST request to post a comment, where multiple parameters are required. REST services might use XML in their responses as a way of organizing structured data. One advantage of using XML is type safety. However, in a stateless system like REST, you should always verify the validity of your input.

Simple URL: http://localhost:8080/StockQuoteLab/services/stockquotes/IBM



- **CSV** (comma-separated values) is more compact.
- **JSON** (JavaScript Object Notation) is trivial to parse in JavaScript clients.
- **HTML** (HyperText Markup Language) and XML are easy to understand and standardized.

Complications using SOAP

- There are several major problems with using SOAP in the Web environment:
  - ✓ The resource being targeted is not known simply from the URL.
  - ✓ The method being invoked is not known from the HTTP method.
  - ✓ The set of methods is completely arbitrary.
  - ✓ Every SOAP application is free to define its own set of methods.
- **BUT…** SOAP-based services have a **much** more robust set of enterprise specifications, including security (WS-Security), than REST.

LearnQuest

13

In the web environment, the resource being targeted is not known simply from the URL; it is hidden within the SOAP message. The method being invoked is not known from the HTTP method; it is also hidden within the SOAP message. The set of methods is completely arbitrary. Also, every SOAP application is free to define its own set of methods.

SOAP messages cannot be utilized by proxy servers, cache servers, etc. The lack of URLs within SOAP messages is a limitation.

SOAP's grouping of resources behind a single URI is contrary to the Web vision.

WSDL

- **WSDL** (Web Services Description Language):
  - ✓ Files exist only for SOAP web services, not for REST based services
    - WSDL 2.0 supports REST, but is rarely adopted.
  - ✓ Flexible in service binding options

LearnQuest

14

W3C recommendation is commonly used to spell out in detail the services offered by a SOAP server. WSDL did not originally support any HTTP operations other than GET and POST.

WSDL files exist only for SOAP web services, not for REST based services. REST services have a WADL file.

WADL

- **WADL** (Web Application Description Language)
  - ✓ An early, but largely ignored, attempt at an IDL for REST
  - ✓ Lightweight, easier to understand, and easier to write than WSDL
  - ✓ In some respects, it is not as flexible as WSDL (no binding to SMTP servers), but it is sufficient for any REST service and much less verbose.

LearnQuest

15

WADL is championed by Sun Microsystems. The WADL file can be shared and used to test the service in the REST explorer within MEB.

REST services have a WADL file: MEB professional subscribers can use our REST Web Services Explorer to test RESTful web services in MEB; all you need is the path to a WADL file describing the service.

Swagger/OpenAPI

- **OpenAPI** (originally, and often still, called Swagger).
  - Widely deployed, this is now the de facto, and arguably de jure, standard for describing REST services.
  - A JSON or YAML document describing a REST interface
  - Sample GET operation:

```
paths:
  /users:
    get:
      summary: Returns a list of users.
  responses:
      '200':    # status code
        description: A JSON array of user names
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
```

LearnQuest

16

OpenAPI Specification: https://swagger.io/specification/

OpenAPI docs: https://swagger.io/docs/specification/

Summary

- In this module, participants learned the following:
  - ✓ REST Security and Validation
  - ✓ Comparison of SOAP and REST
  - ✓ Comparison of WSDL and WADL

LearnQuest

17

Lab Exercise

- Lab: Design a REST service for something in your domain.
  - ✓ Individually or in groups, pick a bounded resource type in your business domain, and design a REST service
    - Resource hierarchy
    - Properties for each resource
    - Operations to support on each resource
    - Status code and content for each operation response
    - Semantics for what each operation means

LearnQuest

18

Questions

LearnQuest

19