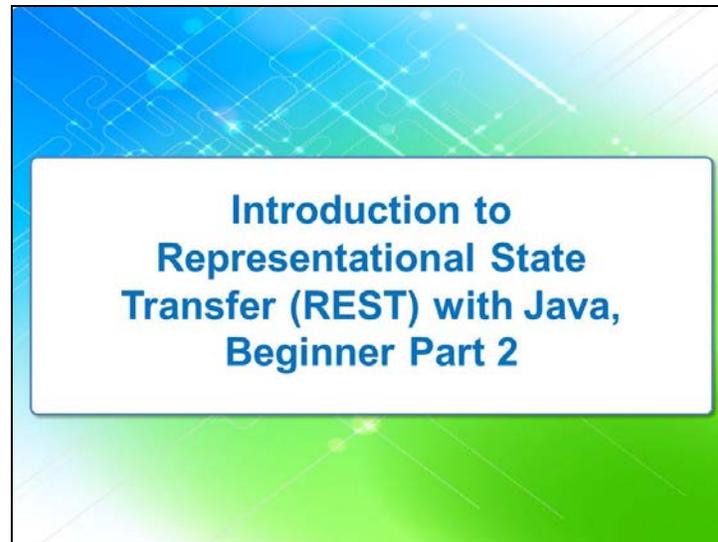


Slide 1

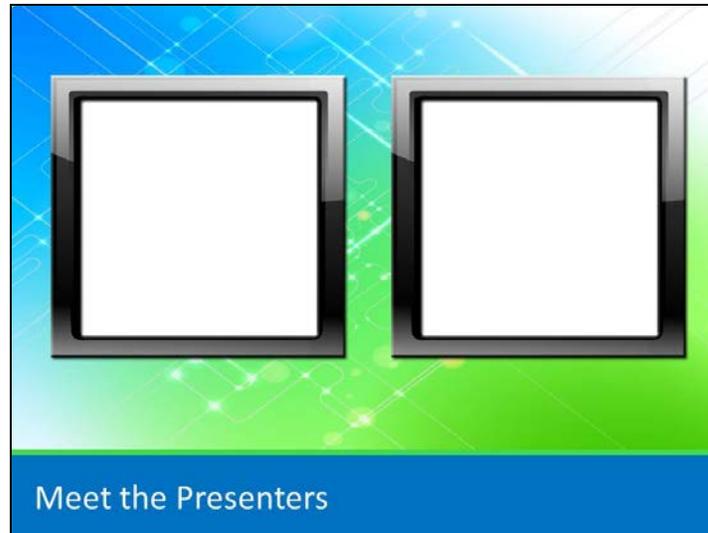


PRESENTER 1 SAY:

On behalf of IT Workforce Development, welcome to today's session of *Introduction to Representational State Transfer (REST) with Java, Beginner Part 2*. This is the second course in a three-part series on REST.

In the first course, you learned about REST and RESTful web services. In today's course, we'll talk about standardized actions, outputs, and design guidelines. We'll also show you standardized actions and outputs and assign you a take-home exercise, so you can tap into what you've learned.

Slide 2



PRESENTER 1 SAY:

We'd like to take a moment to introduce ourselves. I'm PRESENTER 1.

PRESENTER 2 SAY:

And I'm PRESENTER 2.

We're looking forward to providing you with information on today's topic. But before we get started, let's talk about a few Lync items.

Slide 3

Lync Information

Chat/Questions: Use the Chat window

Dial-in: Use the VANTS line in the invitation

Participation Credit: Submit TMS self-certification

Group Participation: Email vaitwd@va.gov

Handouts: Download via the paperclip icon



PRESENTER 1 SAY:

We want this to be an engaging session, so we'll be asking you questions throughout this session, so please use the Chat window to respond. You can also use the Chat window if you have questions for us. We'll answer your questions either during the session or during the Q&A at the end of the session.

We want to make sure you get credit for attending today's session. We'll provide you with instructions on how to complete the self-certification at the end, so be sure to stick around. If you're attending as a group, please email the VA ITWD mailbox and let us know. We'll be able to ensure you all receive completion credit.

PRESENTER 2 SAY:

To access today's handouts, select the paperclip icon in the Lync toolbar above the list of participants. Then, select the right arrow next to the file in the Attachments pop-up menu. From there you can select Open, or Save As. If you choose Save As, you can select where you want to save the download. Note that the Attachments link is not accessible via screen readers. Instead, you can use the CTRL+F keyboard command to access the downloads. Please reach out to us in the Chat or at vaitwd@va.gov for assistance with accessibility and we'll be happy to help.

Slide 4



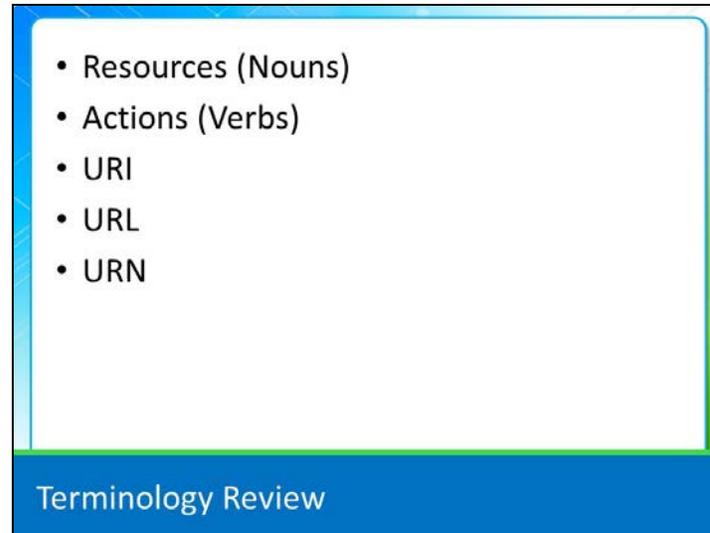
PRESENTER 2 SAY:

In today's course, we're going to talk about standardized actions in REST. Standardized actions are important to understand because they're essentially the requests, or calls, you make to retrieve the information you need. We'll show you how standardized actions work. Then we'll talk about how REST outputs data and some real-world examples of REST outputs. Finally, we'll discuss REST design considerations and answer any questions you have.

As you start using REST, it'll be crucial to use design considerations to prevent issues and avoid frustration that could arise. We'll be providing you the opportunity for further practice through homework exercises that you will complete independently in your own VMware account that we have set up for you. For your assignment, you'll be able to apply what you learned by developing a JSON application programming interface, or API using standardized actions. We'll share more information about the VMware exercise later in the course.

As we go through our topics, please reflect on how you can incorporate them into a web application you're building. We've got a lot to cover, so let's get started!

Slide 5



A slide titled "Terminology Review" with a blue header and a white body. The body contains a bulleted list of terms: Resources (Nouns), Actions (Verbs), URI, URL, and URN.

- Resources (Nouns)
- Actions (Verbs)
- URI
- URL
- URN

Terminology Review

PRESENTER 1 SAY:

Let's spend some time reviewing resources because they're crucial to understand when it comes to REST standardized actions.

We mentioned in the first course that resources are named objects, and standardized actions request instructions that are completed by the resources. Resources are information identified by a string of characters. These character strings are called uniform resource identifiers, or URIs. URIs can be classified as a universal resource locator, URL, or as a uniform resource name, URN. Often, resources are referred to as nouns because they're named objects. Standardized actions are frequently referred to as verbs because they request actions for resources.

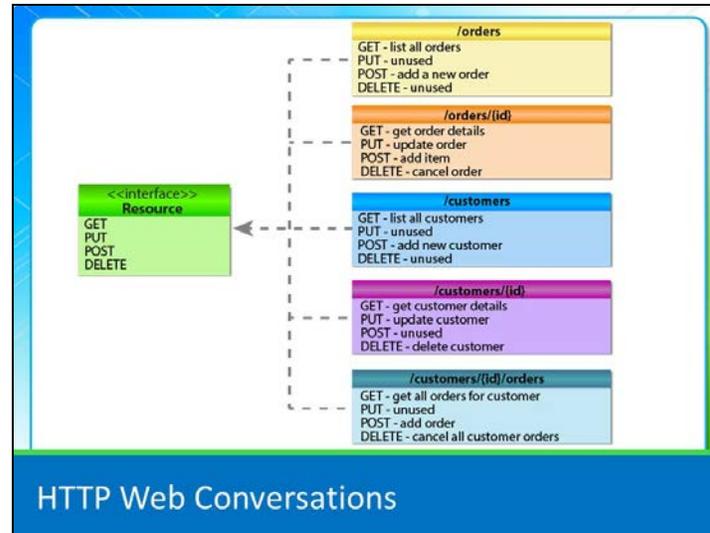
Slide 6



PRESENTER 1 SAY:

Remember, the computer sending the request has to ask for a specific action to get an exact result. Becoming familiar with how requests and standardized actions work will help you decide what type of request should be sent to the web server to get a specific result. Now we're going to talk about HTTP web conversations as they're organized around REST standardized actions and resources.

Slide 7

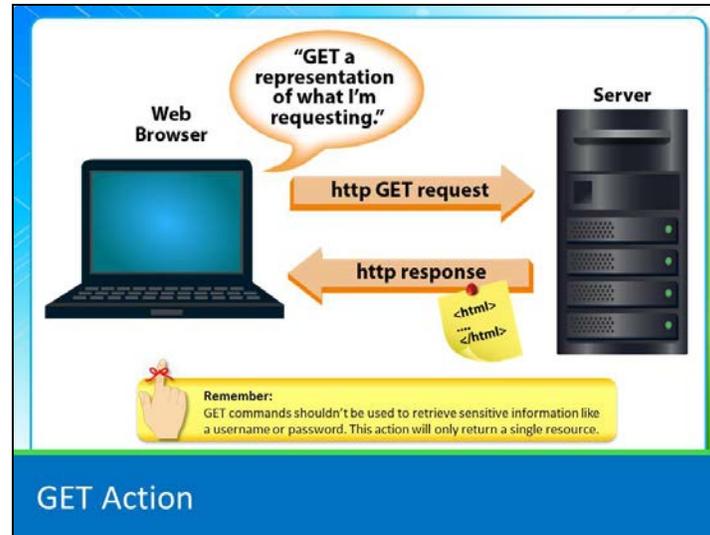


PRESENTER 2 SAY:

Hypertext Transfer Protocol, or HTTP, is an application protocol to transfer data between a client and the server. These conversations tell the web server what the requester wants to see and what action should be applied to the resources. HTTP web conversations allow web browsers to exchange inquiries and receive answers. The conversation starts with a request that is sent from a web browser to the server. Once the request is received and processed, it responds back to the web browser with an answer. The initial request to the server is what is known as a standardized action.

The most common standardized actions are GET, POST, PUT, and DELETE. The only way you can communicate in an HTTP web conversation is with a standardized action. There is no other way to make requests! Each standardized action answers a request differently. The code shown on the screen is how standardized actions would look in an HTTP web conversation when ordering a product over the internet. Let's discuss them in more detail.

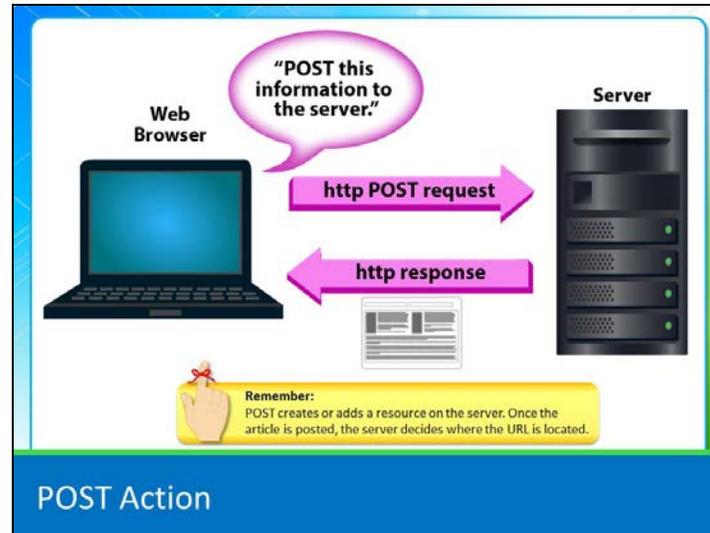
Slide 8



PRESENTER 1 SAY:

GET is the most common standardized action. This action tells the server, "GET a representation of what I'm requesting." The server will retrieve a single resource, such as an HTML document. In REST APIs, a resource won't automatically trigger other GETs, so it's up to the server to determine that. If the programmer wants to retrieve other files such as images, CSS files, or Java Script files, another GET command will need to be performed. It's important to note that these actions are passed through a URL, which isn't secure. As a result, this action shouldn't be used to retrieve sensitive information like a username and password. GET actions are a read-only operation, meaning once the information is retrieved, it can't be altered. This command is useful because it retrieves a representation of data that is being requested by the user.

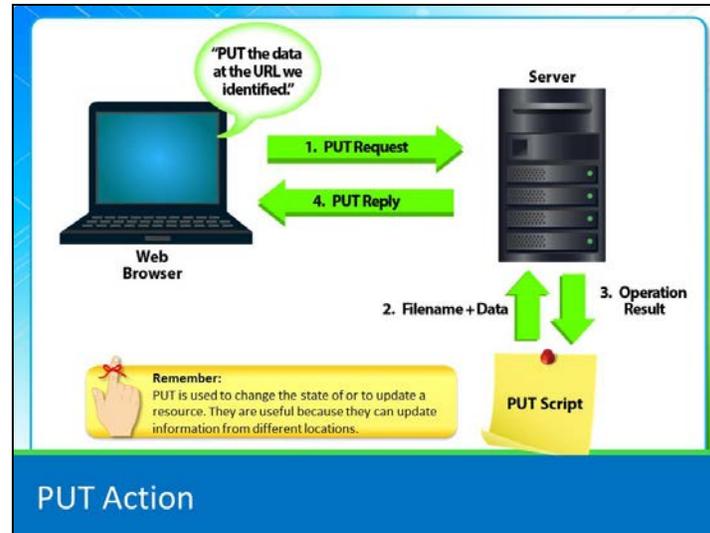
Slide 9



PRESENTER 2 SAY:

POST is used to submit information to the server. It's typically used to update existing information and can also be used to create new content. A POST action often means, "here is the information to create a new user, post it for me." For example, a POST request could be used to add a new article to a site. You would use POST to add a link of the resource to the web page. Once the article is posted, the server decides where the URL is located. The advantage of this request is that you can post information without knowing the location of the resource.

Slide 10



PRESENTER 1 SAY:

PUT is used to create or update a resource. This action tells the server, "PUT the data at the URL we identified." A PUT action tells the data at a specific URL to either overwrite or create new data. This is very similar to a POST action. Since both requests update information, you may be wondering how they differ. They differ in that a PUT action *replaces* the resource while a POST action only *updates* the resource. For instance, you would use a PUT action to replace an image on a website and a POST action to add additional information to existing content.

A PUT request could be used to update resources from different web pages. The programmer would type "HTTP PUT" to update a resource and put a new representation of the resource on a web page.

Slide 11



PRESENTER 2 SAY:

The last standardized action is DELETE. And, as you can probably guess, delete means delete! This action tells the server to "DELETE data from an existing resource." An example would be deleting an old image that isn't needed. These requests are important as they're the only way to remove unnecessary information from specific locations.

Slide 12



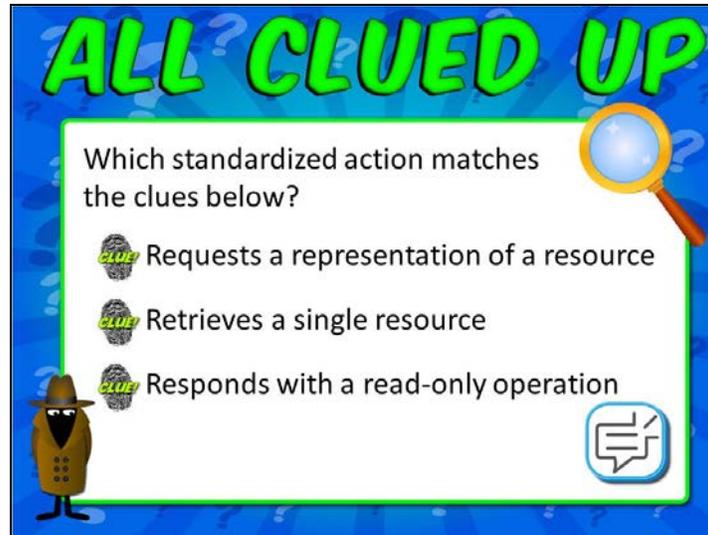
PRESENTER 1 SAY:

Now that we've discussed the four REST standardized actions, let's test your knowledge with a game of All Clued Up!

We'll need your participation, so pay close attention. Clues to a standardized action will be displayed on the screen, and you'll have the chance to name the standardized action we're describing. Once you've decided on an answer, type it in the Chat window located on the left side of your screen.

Here's the first description.

Slide 13



PRESENTER 1 SAY:

What standardized action on the screen matches the clues below? Your clues are:

- Requests a representation of a resource
- Retrieves a single resource
- Responds with a read-only operation.

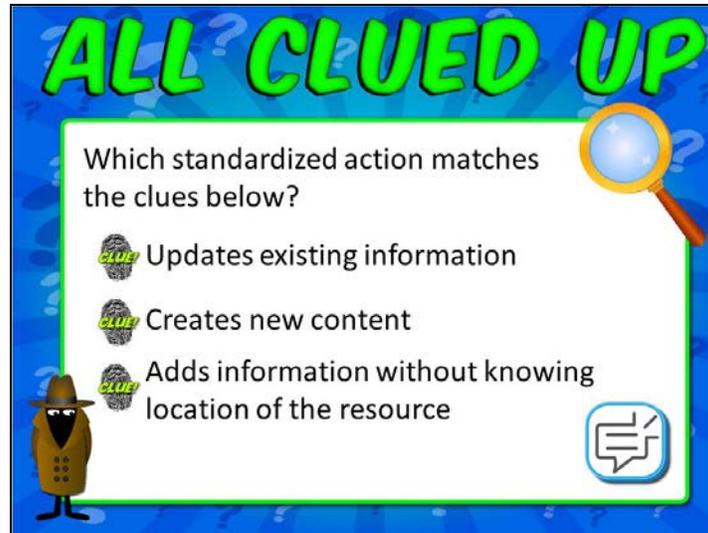
PRESENTER 2 SAY:

Take your time and think about the standardized actions we just discussed and type your answer in the Chat window. The possible choices are GET, POST, PUT, and DELETE.

If you guessed the GET action, you're correct!

Let's move on to our next set of clues.

Slide 14



PRESENTER 1 SAY:

What standardized action on the screen matches the clues below? Your clues are:

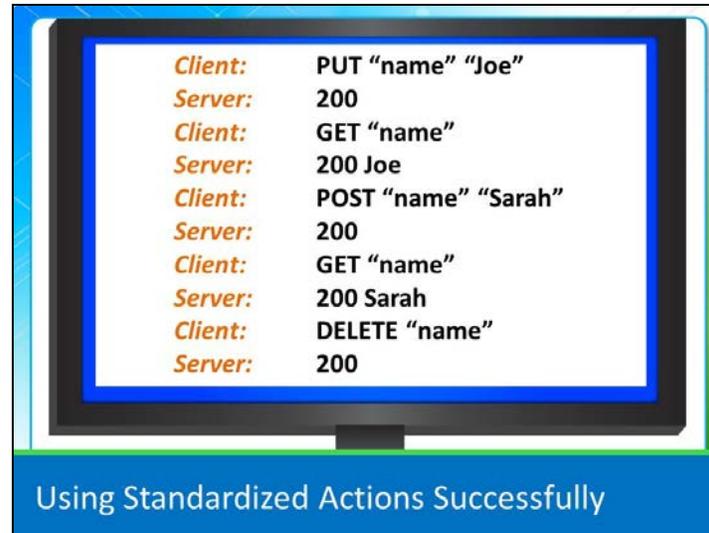
- Updates existing information
- Creates new content
- Adds information without knowing location of the resource.

PRESENTER 2 SAY:

When you're ready to respond, type your answer in the Chat window.

If you guessed the POST action, you're correct!

Slide 15



PRESENTER 1 SAY:

Now that we've talked about standardized actions, we're going to show you how they work by simulating them in a conversation. In our example, we've created an API to access a key value data structure. Our API uses the four standardized actions: GET, PUT, POST, and DELETE. Each of the server responses are a "200," which means they are successful. We'll get more into HTTP response codes in course three. This is an example of a successful HTTP Web conversation. Let's walk through the conversation line by line.

First, the client is asking the server to PUT the "name" "Joe" in its system. The server responds with a "200," which interprets as "OK," a response for successful HTTP requests.

Second, the client is asking the server to GET the "name" that was put in its system. Again, the server responds with a "200 Joe," which interprets as "Ok, I've got the name Joe and the response was successful."

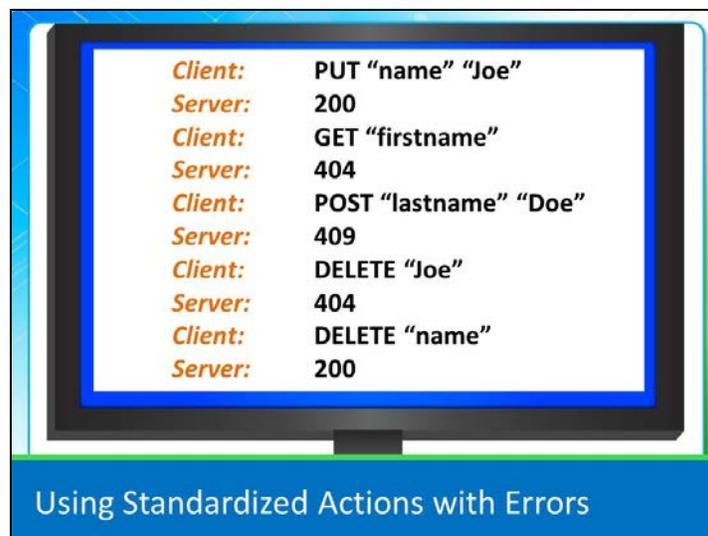
Third, the client is asking the server to POST the "name" "Sarah" in its system. The server responds with a "200," which interprets as "Ok, this has been successful." The name Joe was updated to Sarah.

Fourth, the client is asking the server to GET the "name" "Sarah" that was put in its system. The server responds with a "200 Sarah," which interprets as "Ok, I've got the name Sarah and the response was successful."

Fifth, the client is asking the server to DELETE "name" that was put in its system. The server responds with a "200," which interprets as "Ok, this has been successful."

Now that we've seen a successful HTTP conversation, let's move on to see a conversation with errors.

Slide 16



PRESENTER 2 SAY:

In our conversation, we're going to use the same standardized actions, but this time our response will return errors.

First, the client is asking the server to PUT the "name" "Joe" in its system. The server responds with a "200," which interprets as "Ok," a response for successful HTTP requests.

Second, the client is asking the server to GET the "firstname" in its system. The server responds with a "404," which interprets as "Resource not found." The server responded this way because we called our key "name" instead of "firstname."

Third, the client is asking the server to POST the "lastname" "Doe" in its system. The server responds with a "409," which interprets as a conflict, there wasn't an existing key to update. Since the server recognizes the instruction, it doesn't return a 404, resource not found.

Fourth, the client is asking the server to DELETE "Joe" in its system. The server responds with another "404," which interprets as "Resource not found." The server responded this way because we had called our key "name" instead of "Joe."

Fifth, the client is asking the server to DELETE "name" in its system. The server responds with a "200," which interprets as "Ok," a response for successful HTTP requests. This response was successful because we referred to the correct key name.

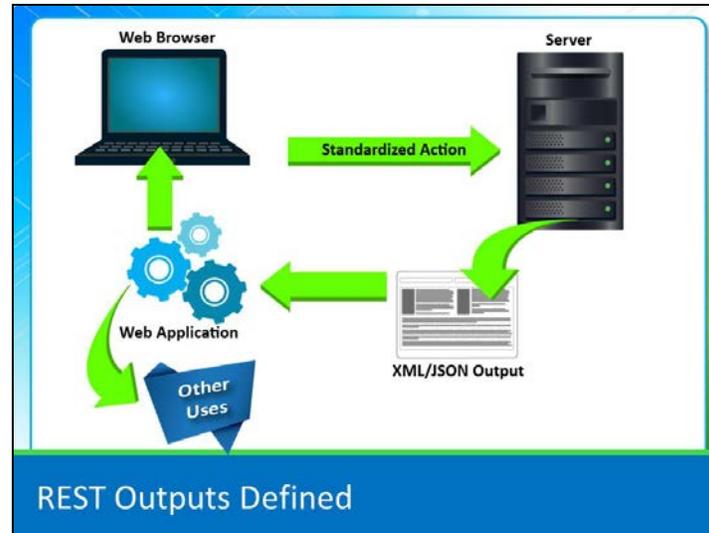
Slide 17



PRESENTER 1 SAY:

Now, we'll discuss REST outputs, including what they are and how they work.

Slide 18



PRESENTER 1 SAY:

When inputting commands to a web browser, the browser sends a standardized action to the web server. REST interprets the action and retrieves the information, and the web server takes the data and sends the output back to the web application. Two common REST outputs are extensible markup language, or XML and JavaScript Object Notation, or JSON. Outputs are essential because they become what the user will see.

The web application determines where the output should be sent. It can either be displayed back to the web browser or used elsewhere. If the output is sent back to the user, then the REST output is any data that can be displayed on the screen. Examples of this range from small bits of information like names, addresses, and phone numbers to large amounts of information, such as a user profile on Facebook or driving directions between two points on a map.

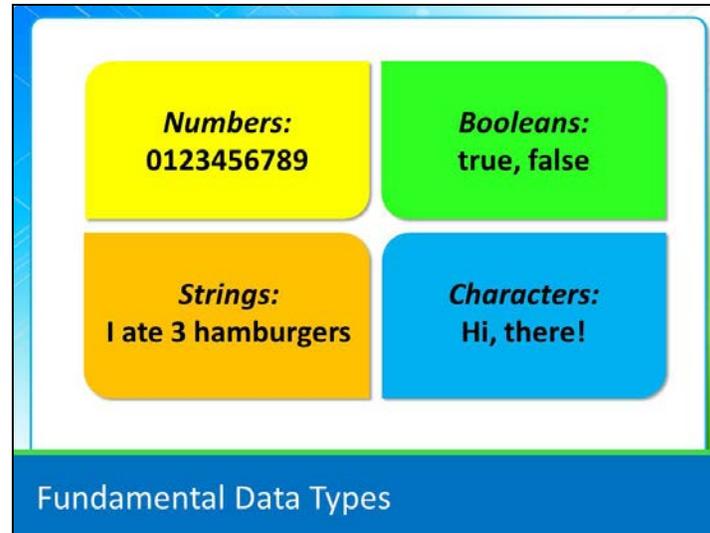
PRESENTER 2 SAY:

Other examples of REST outputs are HTTP response codes. These codes communicate the web server's result of a requested action. The most common HTTP response code is 200, which states that the request was delivered. A recognizable response code is the HTTP 404 error, which we just talked about, that is usually used when a resource is not found. A web server will post a "404 Not Found" to a web page. HTTP response codes are important because they communicate the end result of the initial request.

Additionally, Java API for RESTful Web Services, or JAX-RS, is an API that provides support in creating REST outputs. This API is used to translate Java text to XML or JSON outputs. This results in a quicker outcome for web service creation. Now that we've covered the basics of REST outputs and HTTP response codes, let's move on and talk about fundamental data

types and how they work with REST outputs.

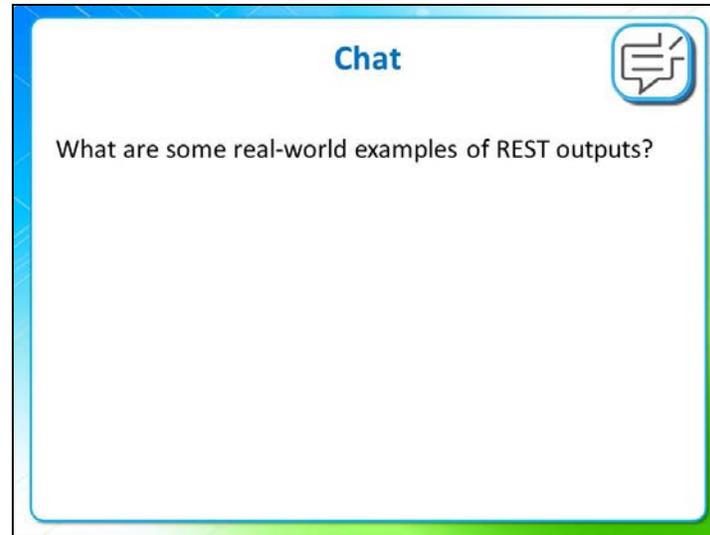
Slide 19



PRESENTER 2 SAY:

In programming languages, data types are used to assign values to information that have specific characteristics. Data types are categorized by a range of values, how they're processed, their meaning, and how they're stored. Fundamental data types include, but aren't limited to, numbers, Booleans, strings, and characters. In REST outputs, data types have an agreement with any API between the sender and the receiver that the returned data should stay consistent. For example, if a message is sent requesting the answer as a string, the receiver can't send back a Boolean. That would violate the contract.

Slide 20



PRESENTER 1 SAY:

Now that we've talked about REST outputs and data types, we'd like to hear from you. What are some real-world examples of REST outputs? Type your responses in the Lync Chat window on the left.

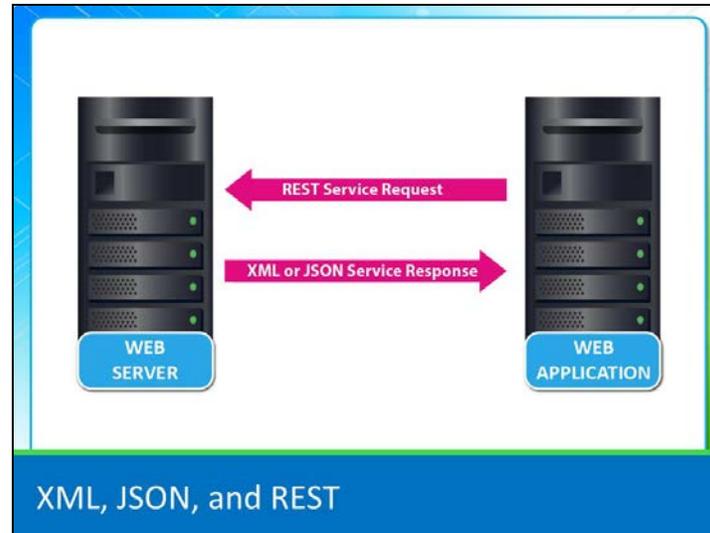
PRESENTER 2 SAY:

Remember, REST APIs allow communication between your application and another specific REST API application. Let's say you run a Twitter newsfeed. You can write software that uses Twitter's REST API to send messages to your followers. If the message is successfully sent to Twitter, you would respond with HTTP 200, or OK. If there was some issue, maybe you'd respond with a 404 or 409, depending on the situation.

PRESENTER 1 SAY:

Now, we'll move on to discuss XML and JSON, the two common formats used for REST outputs.

Slide 21



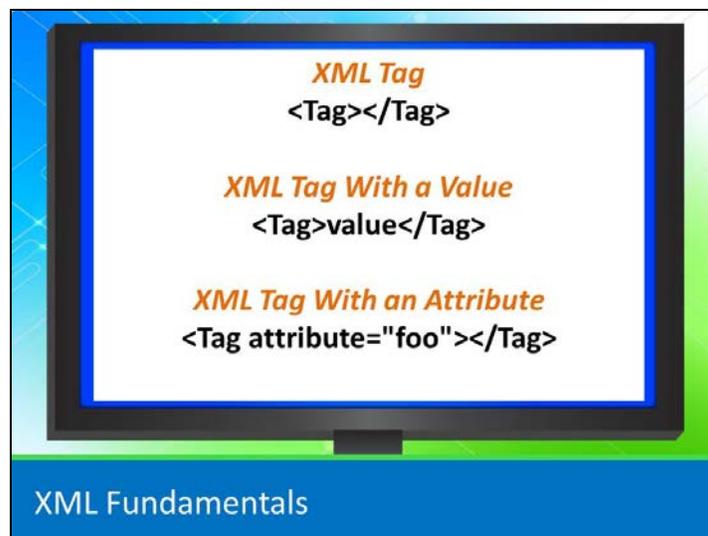
PRESENTER 2 SAY:

As we mentioned earlier, REST sends and receives HTTP messages. The information sent back to the web application is in either XML or JSON format. Which one you use will depend on many factors, including VA guidelines and other APIs you're communicating with.

Internet media types are important to XML and JSON because they identify the type of data a file on the Internet contains. For example, web browsers use them to determine how to display files that are in XML or JSON. It's important to remember that RESTful web services only support one data format. For example, if you send a media type of XML to a JSON-only web service, you're going to get JSON data back.

Let's move on and discuss the fundamentals of XML and JSON as well as some benefits of each.

Slide 22



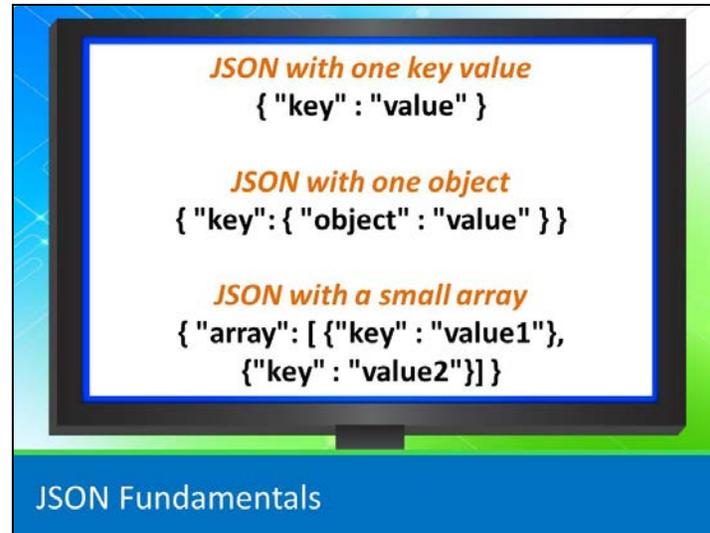
PRESENTER 2 SAY:

XML is a markup language used to make sure there's a standard when sending data. It defines a set of rules for encoding documents in both a human-readable and machine-readable format. It's a popular and widely implemented standard. You can use it to create documents and data records that are fully transferable and independent from platforms. XML differs from most markup languages because it has the capability to carry and store data, focusing on the way the data is created in the document. Other markup languages only display how data looks.

XML has a variety of features, and there are three that you should become familiar with. They are tags, values, and attributes. Tags are the names of individual elements. In XML, tags aren't predefined; the author of the document is in charge of creating them. XML tags with values are the data contained between the open and closed tags. A XML tag with an attribute is data contained within the tag itself. XML tags and their attributes are defined in document type definitions, or DTDs, also known as schemas. The schemas can be loose definitions or constrain what types of characters are allowed in values.

Now that we've discussed the basics of XML, let's move on to JSON fundamentals.

Slide 23



PRESENTER 1 SAY:

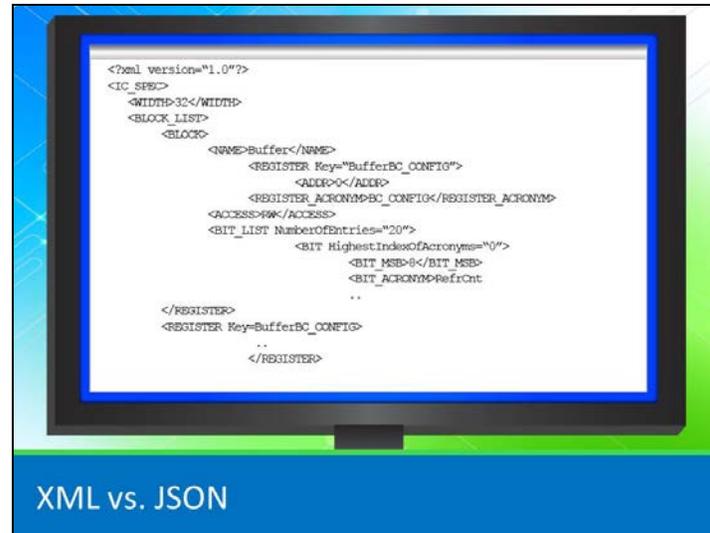
JSON is a markup language similar to XML. It stores information in an organized manner and outputs data logically so it can be interpreted for its users. Both languages use hierarchal values and are transported similarly. For example, JSON contains values within values, much like XML contains elements within elements. The JSON data is transported in the body of an HTTP message just like with XML.

JSON differs from XML because its library is smaller. However, JSON is faster and easier to dissect for programmers. It's becoming the standard for web-based applications. A study in 2012 from Engineer Magazine found that 57 percent of all web-based applications are built using JSON.

JSON has three elements that you should become familiar with. Those are key values, objects, and arrays. A JSON key value is a string in double quotes with a value, a number, true or false, and an object or array. A JSON object is an unordered set of name and value pairs and always begins and ends with a left and a right brace. Each name is followed by a colon. A JSON array is an ordered collection of values. Like a JSON object, arrays begin with a left bracket and end with a right bracket and are separated by a comma.

Now that we're familiar with JSON fundamentals, let's move on to discuss the benefits and differences between JSON and XML.

Slide 24



PRESENTER 1 SAY:

XML is a larger, more structured language and has a richer set of data-type checking tools. You can run your XML against XML schema definitions, or X-S-Ds, to make sure that data is in the format you want. XML is generic and unconstrained. You can take almost any data problem, such as configuration files or log files, and apply XML to it. JSON is the opposite of XML. It's limited and it can't do a lot of things that XML can. However, JSON's constraints produce simple and efficient results. It's smaller and concise structure makes it easier to use in a web environment and makes online performance less of an issue.

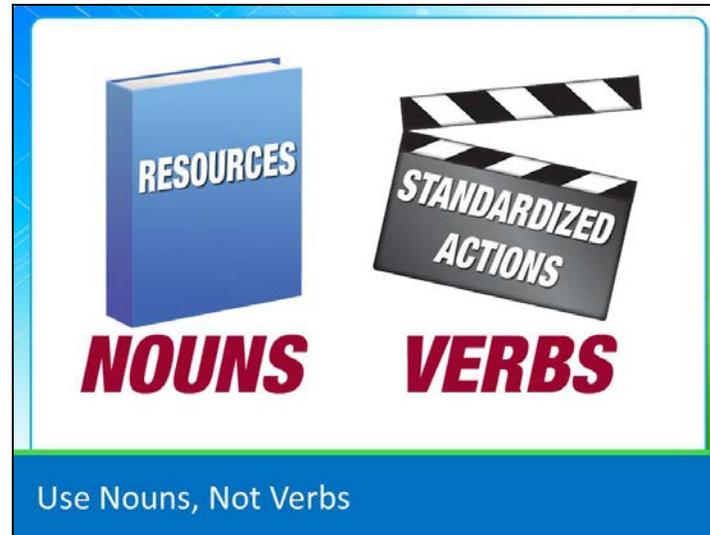
Slide 25



PRESENTER 1 SAY:

Now, let's talk about REST design guidelines that should be used to create RESTful web services. We could spend hours talking about design guidelines for RESTful web services, but we have narrowed it down to two basic guidelines. The guidelines are to use nouns, not verbs, and to create an effective API so that other programmers will be able to follow your logic. Let's talk in more depth about these.

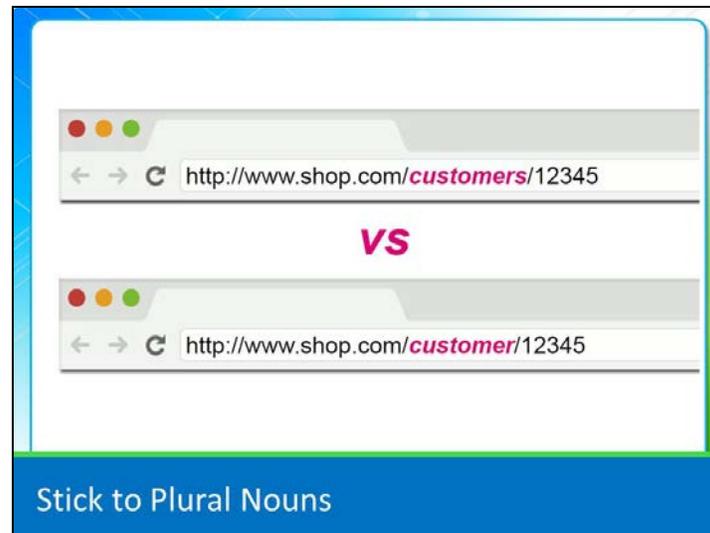
Slide 26



PRESENTER 1 SAY:

First, use nouns, not verbs. It's important to separate your API into logical resources. Like we discussed earlier, resources are nouns not verbs. Resources are identified by a URI. A good RESTful URI is a noun such as a "user" or book." It could also be a collection of resources, which would be specified with a plural noun, such as "books." RESTful URIs also identify resources or nouns. They tell you what they are. RESTful URI's shouldn't tell you what they do. Like we discussed earlier, the "do" comes when you apply a verb or a standardized action to the URL.

Slide 27



PRESENTER 1 SAY:

Another consideration is to stick to plural nouns in your URI node names. It's an accepted practice to always use plural node names to keep your API URIs consistent across all HTTP methods. This is based on the concept that "customers" are a collection within the service suite, and the ID or "12345" refers specifically to an individual customer. Whatever convention you use, make sure it is acceptable at VA and is recognized across your development team to ensure consistent interfaces.

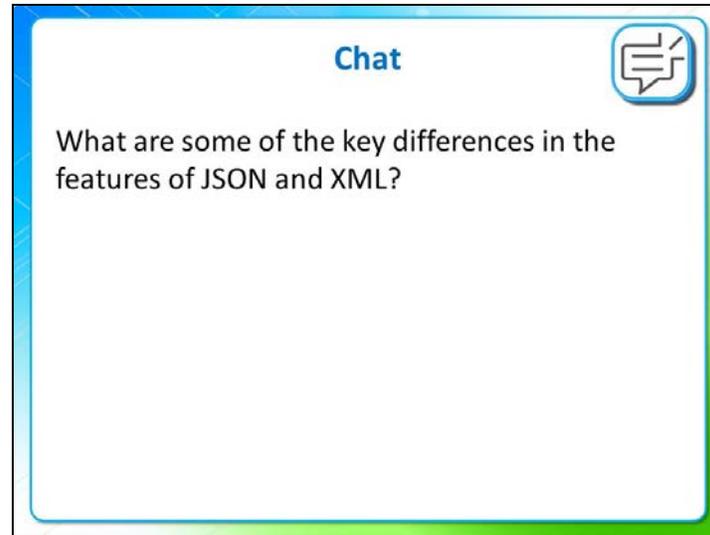
Slide 28



PRESENTER 1 SAY:

You should create an effective API so that other programmers will be able to follow your logic. Remember, an API is an interface plus the documentation. So when creating APIs, it's important to remember that they're only as good as your documentation. Remember, APIs are also user interfaces for other developers to use in the future. When creating APIs, you need to make them easy to understand. It's a good practice to create APIs that have version numbers. Doing this is a great way for other users to locate and choose which version to use.

Slide 29



PRESENTER 2 SAY:

Now that we've talked about REST design guidelines, we'd like to see what you've learned about JSON and XML. So based on our discussion, what are some key differences in the features of JSON and XML? Type your responses in the Lync Chat window on the left.

PRESENTER 1 SAY:

Something that I think is interesting about JSON and XML is that they have different structures. Requests and returned data will always stay in a consistent format. If you send an XML request, you can't expect to receive your response in JSON.

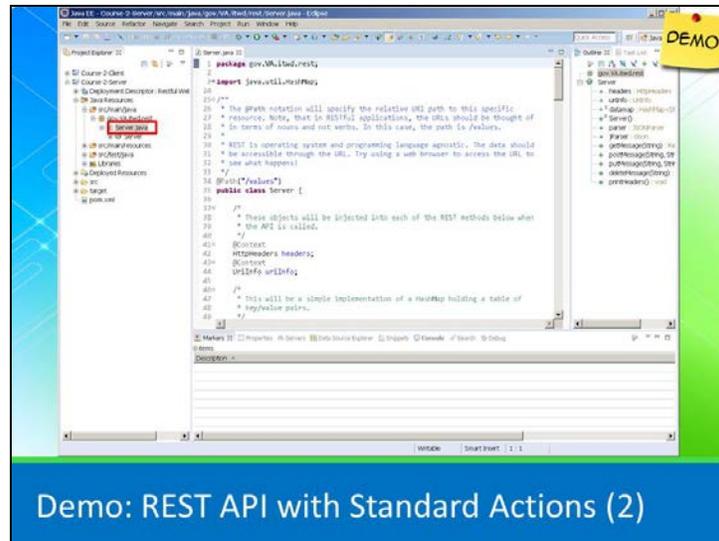
Slide 30



PRESENTER 2 SAY:

Now we're going to show you how to create a more complex API with the standard actions we discussed today, which is also your homework exercise. In your exercise, you're going to use the four standard actions: GET, PUT, POST, and DELETE. We're going to show you what you'll do for this exercise.

Slide 31

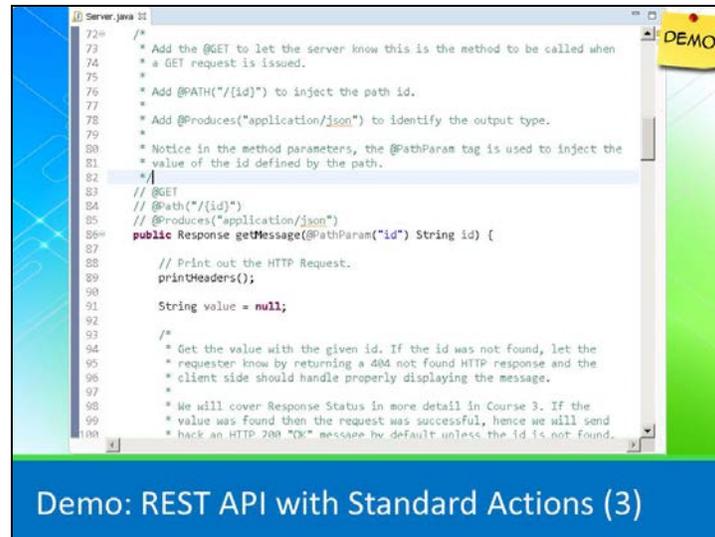


Demo: REST API with Standard Actions (2)

PRESENTER 2 SAY:

To begin this demo, we're going to use your exercise files to import Course-2.zip into Eclipse, and then expand the folders to find the Server.java file. Next, you'll need to expand the Java Resources folder. Once you've expanded this folder, locate the src/main/java folder and expand it to open up the gov.va.rest file. After expanding the gov.VA.itwd.rest file, locate the Server.java file. Double-click the Server.java file to open the exercise on your screen. Your screen will open and code will be displayed.

Slide 32



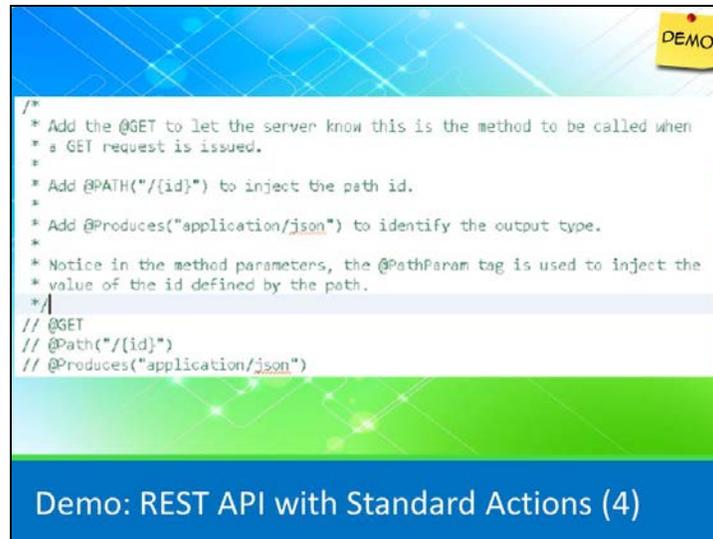
```
72  /*
73  * Add the @GET to let the server know this is the method to be called when
74  * a GET request is issued.
75  *
76  * Add @PATH("/{id}") to inject the path id.
77  *
78  * Add @Produces("application/json") to identify the output type.
79  *
80  * Notice in the method parameters, the @PathParam tag is used to inject the
81  * value of the id defined by the path.
82  */
83  // @GET
84  // @Path("/{id}")
85  // @Produces("application/json")
86  public Response getMessage(@PathParam("id") String id) {
87
88      // Print out the HTTP Request.
89      printHeaders();
90
91      String value = null;
92
93      /*
94      * Get the value with the given id. If the id was not found, let the
95      * requester know by returning a 404 not found HTTP response and the
96      * client side should handle properly displaying the message.
97      *
98      * We will cover Response Status in more detail in Course 3. If the
99      * value was found then the request was successful, hence we will send
100     * back an HTTP 200 "OK" message by default unless the id is not found.
```

Demo: REST API with Standard Actions (3)

PRESENTER 2 SAY:

Scroll down until you see the block notes about @GET. Locate the line comment about adding an @Path and @Produces. Your exercise for the GET standardized action starts here. Notice it says in green in the code, “Add the @GET to let the server know this is the method to be called when a GET request is issued.”

Slide 33



```
/*
 * Add the @GET to let the server know this is the method to be called when
 * a GET request is issued.
 *
 * Add @PATH("/{id}") to inject the path id.
 *
 * Add @Produces("application/json") to identify the output type.
 *
 * Notice in the method parameters, the @PathParam tag is used to inject the
 * value of the id defined by the path.
 */
// @GET
// @Path("/{id}")
// @Produces("application/json")
```

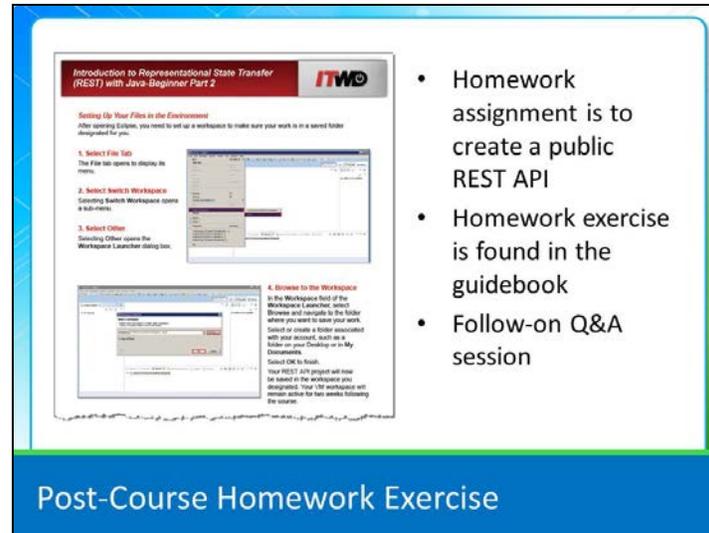
Demo: REST API with Standard Actions (4)

PRESENTER 2 SAY:

Specifically, it says to add @GET, the @PATH("/{id}") to inject the path ID, and @Produces("application/json") to identify the output type. The green notes tell you exactly what to put in the code and why.

Just follow the line comments and add the necessary Java when it asks you for it. It's very easy to follow the directions in the code. The Guidebook walks you through the whole activity. When you scroll down through the code, you'll see additional block notes about @POST, @PUT, and @DELETE. Just follow the directions for each one. That's it!

Slide 34



The slide contains a screenshot of a presentation slide titled "Introduction to Representational State Transfer (REST) with Java-Beginner Part 2". The slide lists four steps for setting up the environment:

- 1. Select File Tab:** The File tab opens to display its menu.
- 2. Select Switch Workspace:** Clicking Switch Workspace opens a sub-menu.
- 3. Select Other:** Clicking Other opens the Workspace Launcher dialog box.
- 4. Return to the Workspace:** In the Workspace field of the Workspace Launcher, click Browse and navigate to the folder where you want to store your work. Select or create a folder associated with this account, such as a folder on your Desktop or in My Documents. Select OK to finish. Your REST API project will now be listed in the workspace area designated "Your IDE workspace will remain active for the entire training" in the sidebar.

Below the screenshot is a blue box with the text "Post-Course Homework Exercise".

- Homework assignment is to create a public REST API
- Homework exercise is found in the guidebook
- Follow-on Q&A session

PRESENTER 1 SAY:

For the homework assignment, we're going to ask you to create a public REST API using the four main REST standardized actions GET, POST, PUT, and DELETE. You're going to go into the same file that you did for course one and complete the code. You'll also be able to review the standardized functions and become more familiar with how these work.

For those of you who haven't done this before, it may sound a little scary, but the guidebook will walk you through it. We would like for you to complete this assignment before the *REST Beginner Part 3* session because we will build upon what you have learned in this course.

If you have any issues or questions about the assignment, we'll have a separate Q&A session that you can dial into and ask questions.

Slide 35



PRESENTER 1 SAY:

We've covered a lot today. We first talked about REST standardized actions and HTTP web conversations. We explained common standardized actions and how they worked. We also described how to use standardized actions using the Jersey library. Next, we talked about REST outputs and fundamental data types, and then we discussed XML and JSON and talked about their unique characteristics. Along the way we showed you each of these languages and talked about guidelines for how to design RESTful web services.

PRESENTER 2 SAY:

Now that we've given a summary, let's open up the conversation to you and answer any questions you might have.

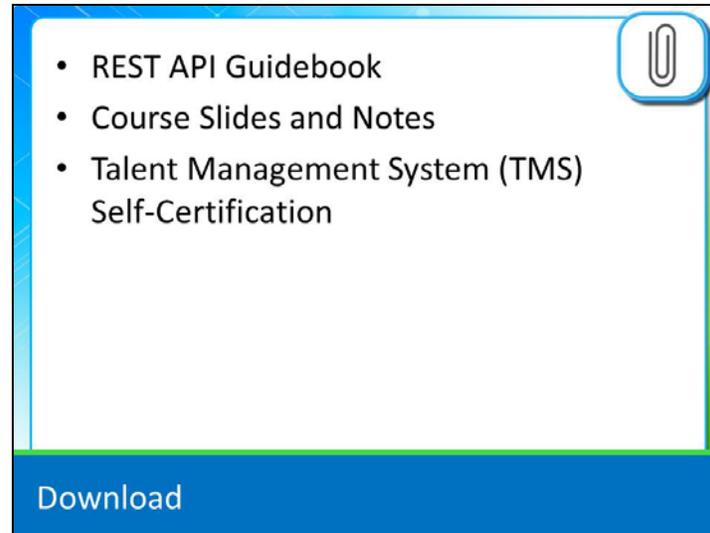
Slide 36



PRESENTER 1 SAY:

Please type in your questions in the Chat window on the left-hand side of your screen.

Slide 37



PRESENTER 1 SAY:

We've got three handouts for you to download today. We've got a guidebook for your self-paced homework assignment, the full course transcript, and the directions for self-certifying in the TMS that you took this course.

To access the downloads, select the paperclip icon in the Lync toolbar above the list of participants. Next, select the right arrow next to the file in the Attachments pop-up menu. From there you can select Open, or select Save As to navigate to a location where you can save the download.

Slide 38



PRESENTER 1 SAY:

As we mentioned earlier, this is the second course in a three-course series talking about the basics of REST. Be sure to watch for ITWD training announcements or look at the training calendar in the ITWD portal for additional REST training. Thank you for joining us in today's session.

Slide 39

TMS Self-Certification

- Log in to the TMS <https://www.tms.va.gov>
- Enter **3878053** in the Search Catalog field on your TMS home page and select the **Go** button
- Select the **Start Course** button
- Select the **Yes** button
- Select the **OK** button
- Select the **Close Window** button
- **NEW!** Complete the course evaluation survey that is on your TMS To-Do List

For assistance, contact the **TMS Help Desk:**
vatmshelp@va.gov or **1-866-496-0463**

PRESENTER 1 SAY:

Take a minute to self-certify to get credit for your participation in this training. TMS self-certification instructions are provided on the slide.

Once you're self-certified, you will receive a link to complete an evaluation for this training.

We will leave the instructions up for a couple minutes to allow you time to self-certify.

This concludes *Introduction to Representational State Transfer (REST) with JAVA, Beginner Part 2*.

Thank you for your participation.