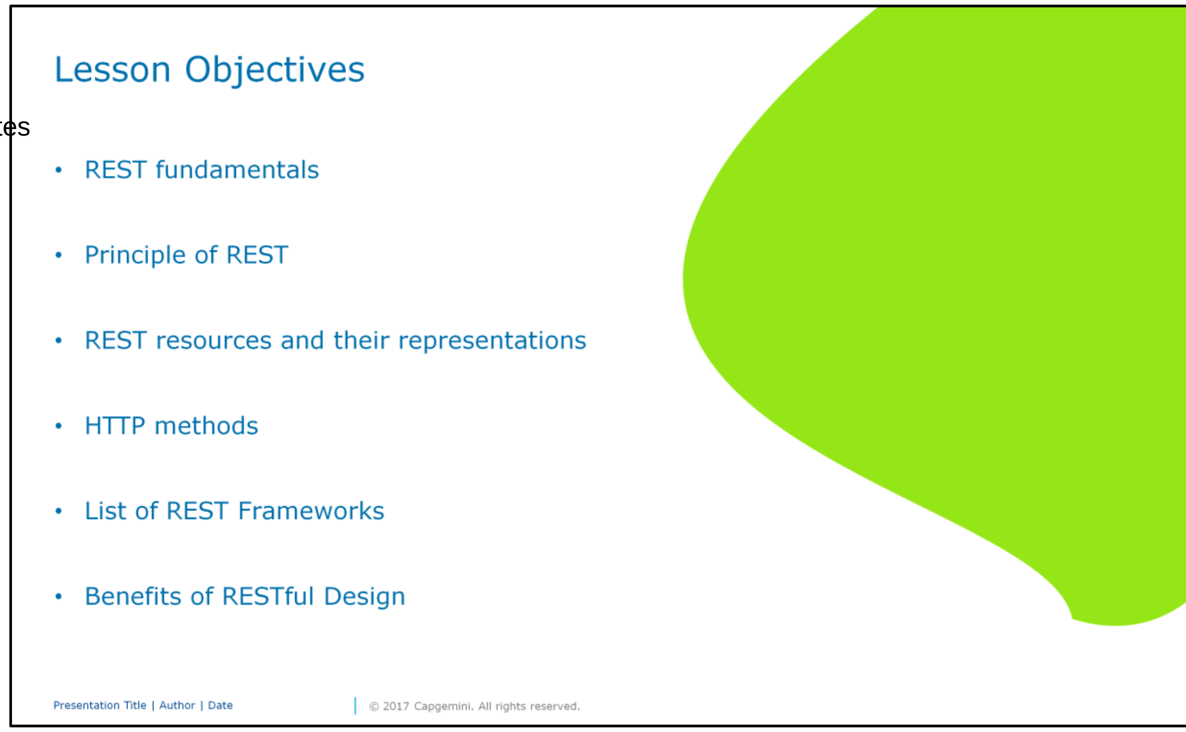
RESTFul Webservices with Spring

Introduction to REST

Capgemini

Add instructor notes here.



Lesson Objectives

- REST fundamentals
- Principle of REST
- REST resources and their representations
- HTTP methods
- List of REST Frameworks
- Benefits of RESTful Design

Following contents would be covered:

1.1 : What are Web services

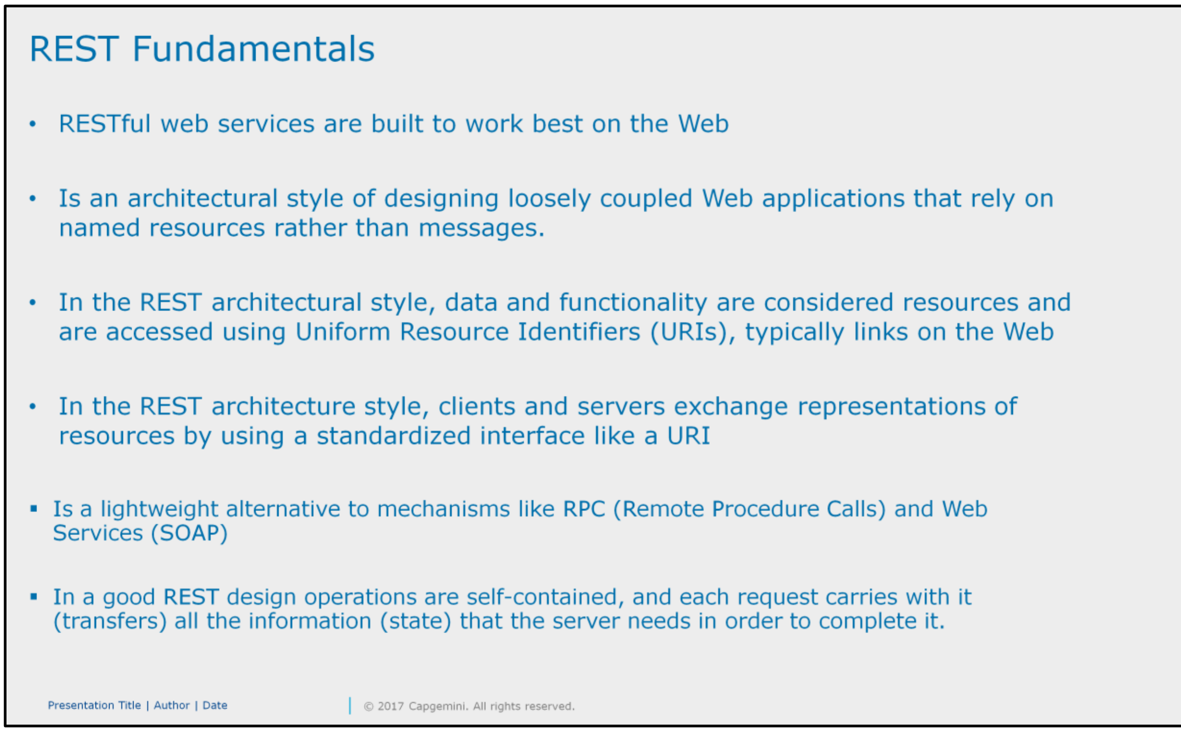
  1.1.1 Web service components and architecture

  1.1.2 How do Web services work

1.2: HTTP and SOAP messages

1.3: Overview of JAX – WS and JAX – RS

Slide illustrates regarding REST architecture.



REST Fundamentals

- RESTful web services are built to work best on the Web
- Is an architectural style of designing loosely coupled Web applications that rely on named resources rather than messages.
- In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web
- In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface like a URI
- Is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP)
- In a good REST design operations are self-contained, and each request carries with it (transfers) all the information (state) that the server needs in order to complete it.

Presentation Title | Author | Date | © 2017 Capgemini. All rights reserved.

REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP)

Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web

The following principles encourage RESTful applications to be simple, lightweight, and fast:

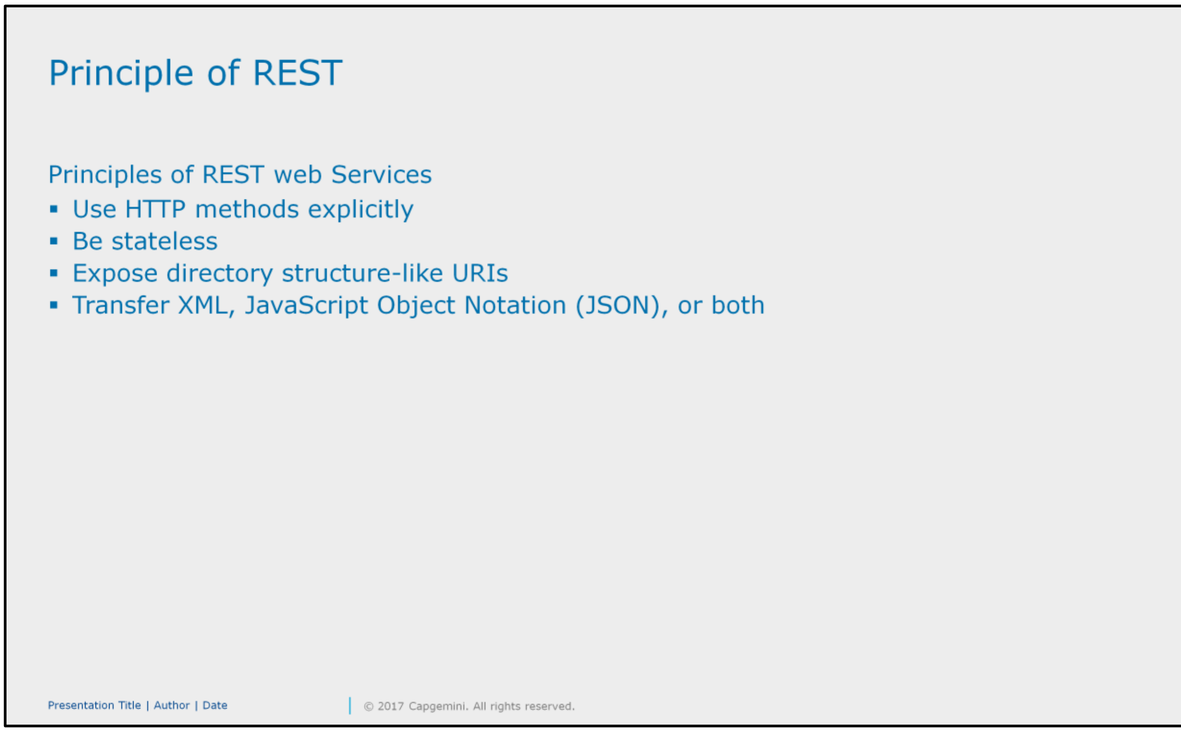
Resource identification through URI: A RESTful web service exposes a set of resources. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT updates an existing resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.

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

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

Slide illustrates regarding REST architecture.



Principle of REST

Principles of REST web Services

- Use HTTP methods explicitly
- Be stateless
- Expose directory structure-like URIs
- Transfer XML, JavaScript Object Notation (JSON), or both

REST is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP)

Representational State Transfer (REST) is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web

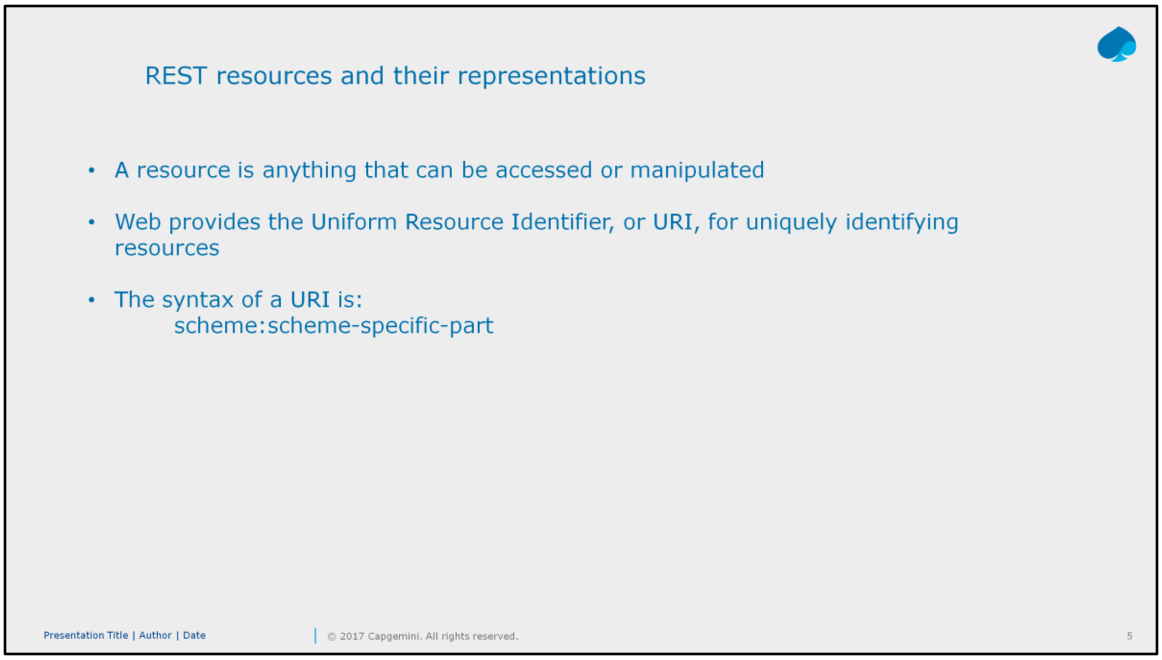
The following principles encourage RESTful applications to be simple, lightweight, and fast:

Resource identification through URI: A RESTful web service exposes a set of resources. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT updates an existing resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.

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

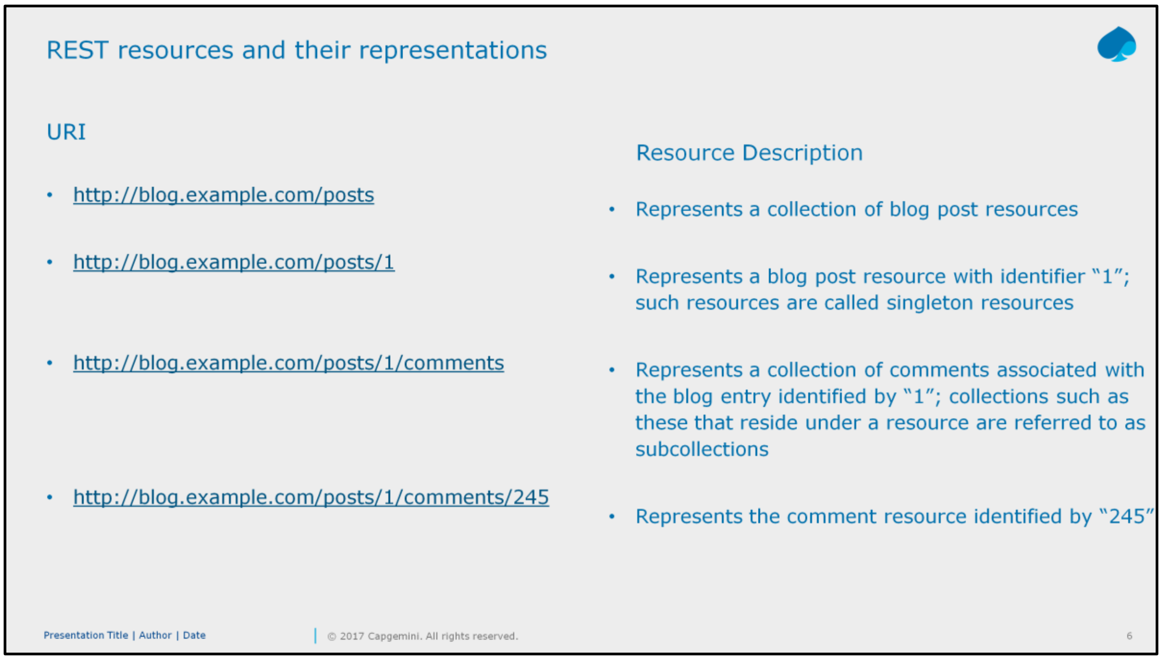
Stateful interactions through hyperlinks: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form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


REST resources and their representations

- A resource is anything that can be accessed or manipulated
- Web provides the Uniform Resource Identifier, or URI, for uniquely identifying resources
- The syntax of a URI is:
  scheme:scheme-specific-part

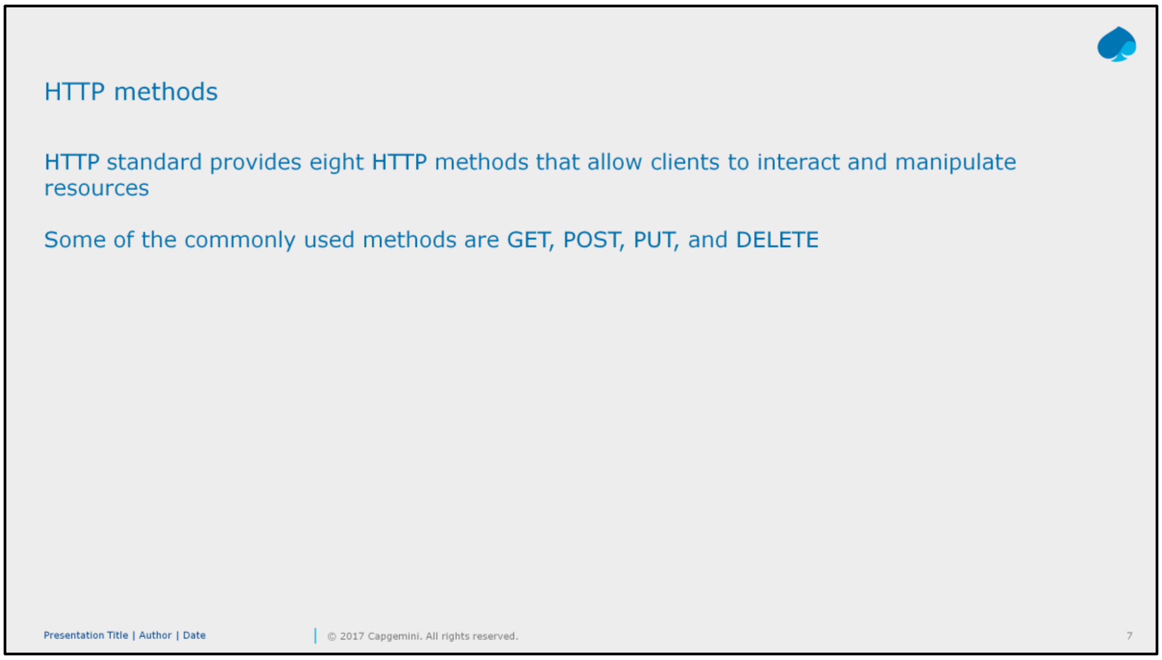
Presentation Title | Author | Date | © 2017 Capgemini. All rights reserved. 5

Fundamental to REST is the concept of resource. A resource is anything that can be accessed or manipulated. Examples of resources include “videos,” “blog entries,” “user profiles,” “images,” and even tangible things such as persons or devices. Resources are typically related to other resources. For example, in an ecommerce application, a customer can place an order for any number of products. In this scenario, the product resources are related to the corresponding order resource. It is also possible for a resource to be grouped into collections. Using the same ecommerce example, “orders” is a collection of individual “order” resources.
Identifying Resources
Before we can interact and use a resource, we must be able to identify it. The Web provides the Uniform Resource Identifier, or URI, for uniquely identifying resources. The syntax of a URI is:
scheme:scheme-specific-partThe scheme and the scheme-specific-part are separated using a semicolon. Examples of a scheme include http or ftp or mailto and are used to define the semantics and interpretation of the rest of the URI. Take the example of the URI—http://www.apress.com/9781484208427. The http portion of the example is the scheme; it indicates that a HTTP scheme should be used for interpreting the rest of the URI. The HTTP scheme, defined as part of RFC 7230,[2] indicates that the resource identified by our example URI is located on a machine with host name apress.com.



REST resources and their representations

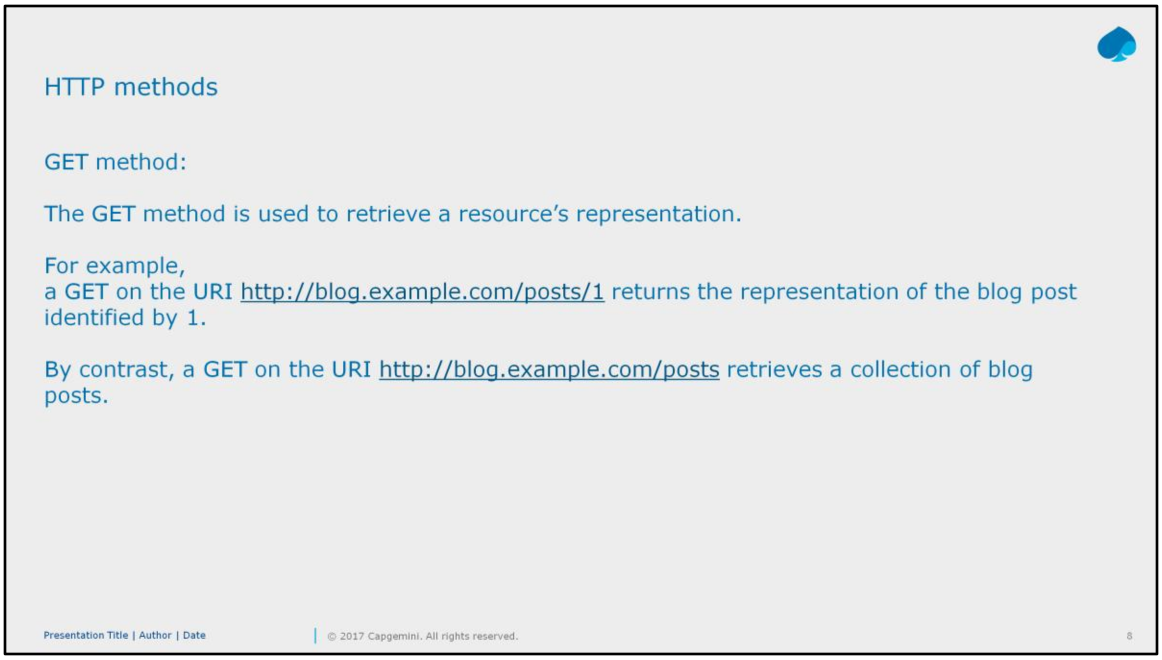
URI	Resource Description
http://blog.example.com/posts	Represents a collection of blog post resources
http://blog.example.com/posts/1	Represents a blog post resource with identifier "1"; such resources are called singleton resources
http://blog.example.com/posts/1/comments	Represents a collection of comments associated with the blog entry identified by "1"; collections such as these that reside under a resource are referred to as subcollections
http://blog.example.com/posts/1/comments/245	Represents the comment resource identified by "245"



HTTP methods

HTTP standard provides eight HTTP methods that allow clients to interact and manipulate resources

Some of the commonly used methods are GET, POST, PUT, and DELETE



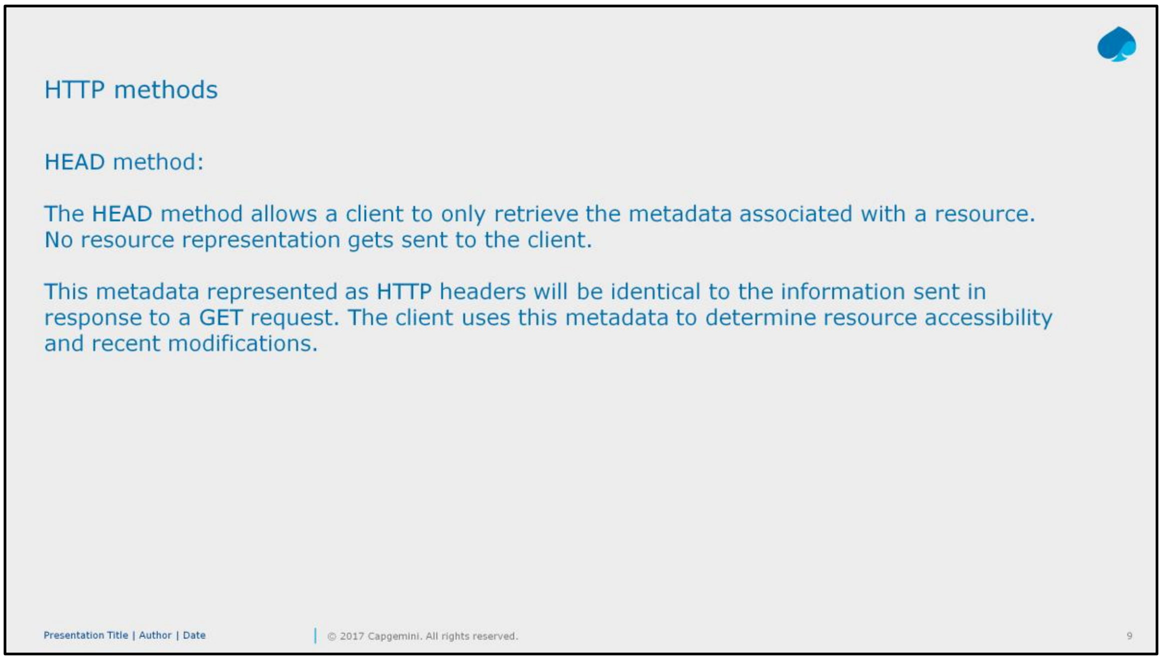
HTTP methods

GET method:

The GET method is used to retrieve a resource’s representation.

For example,
a GET on the URI http://blog.example.com/posts/1 returns the representation of the blog post identified by 1.

By contrast, a GET on the URI http://blog.example.com/posts retrieves a collection of blog posts.

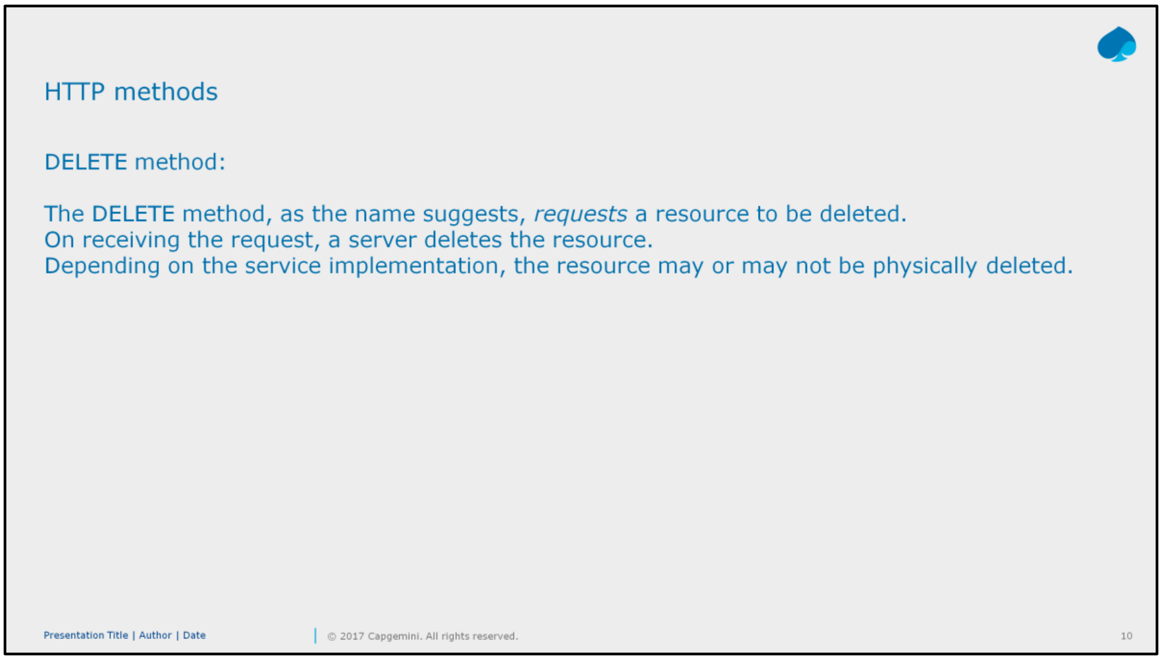


HTTP methods

HEAD method:

The HEAD method allows a client to only retrieve the metadata associated with a resource. No resource representation gets sent to the client.

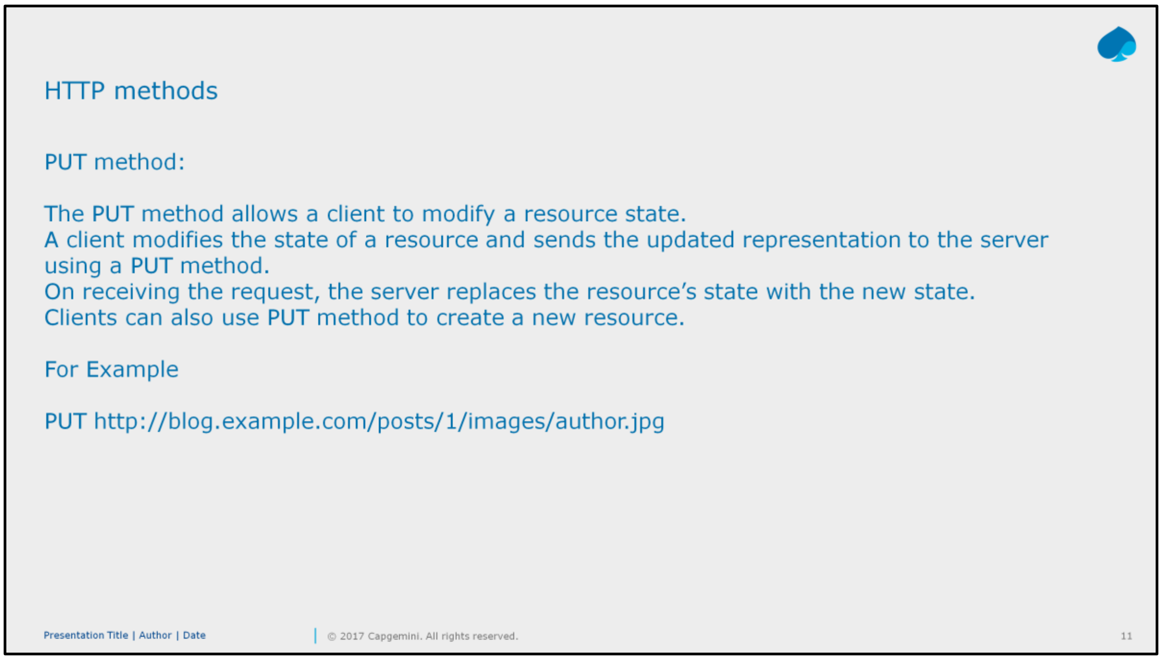
This metadata represented as HTTP headers will be identical to the information sent in response to a GET request. The client uses this metadata to determine resource accessibility and recent modifications.



HTTP methods

DELETE method:

The DELETE method, as the name suggests, *requests* a resource to be deleted.
On receiving the request, a server deletes the resource.
Depending on the service implementation, the resource may or may not be physically deleted.



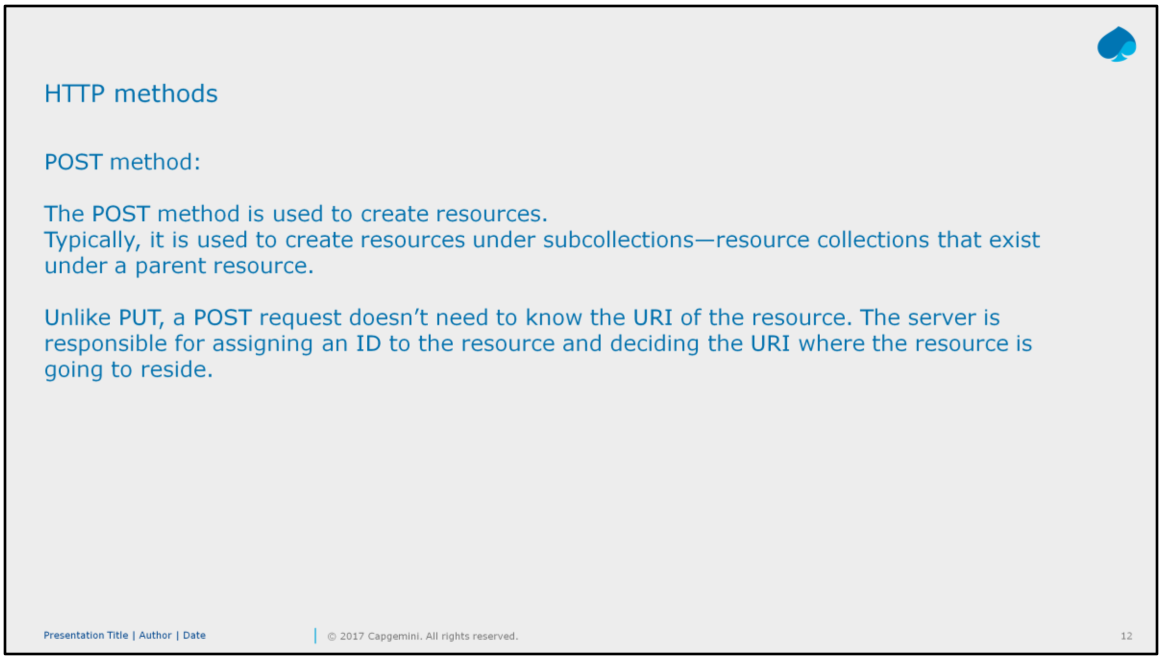
HTTP methods

PUT method:

The PUT method allows a client to modify a resource state.
A client modifies the state of a resource and sends the updated representation to the server using a PUT method.
On receiving the request, the server replaces the resource's state with the new state.
Clients can also use PUT method to create a new resource.

For Example

PUT http://blog.example.com/posts/1/images/author.jpg

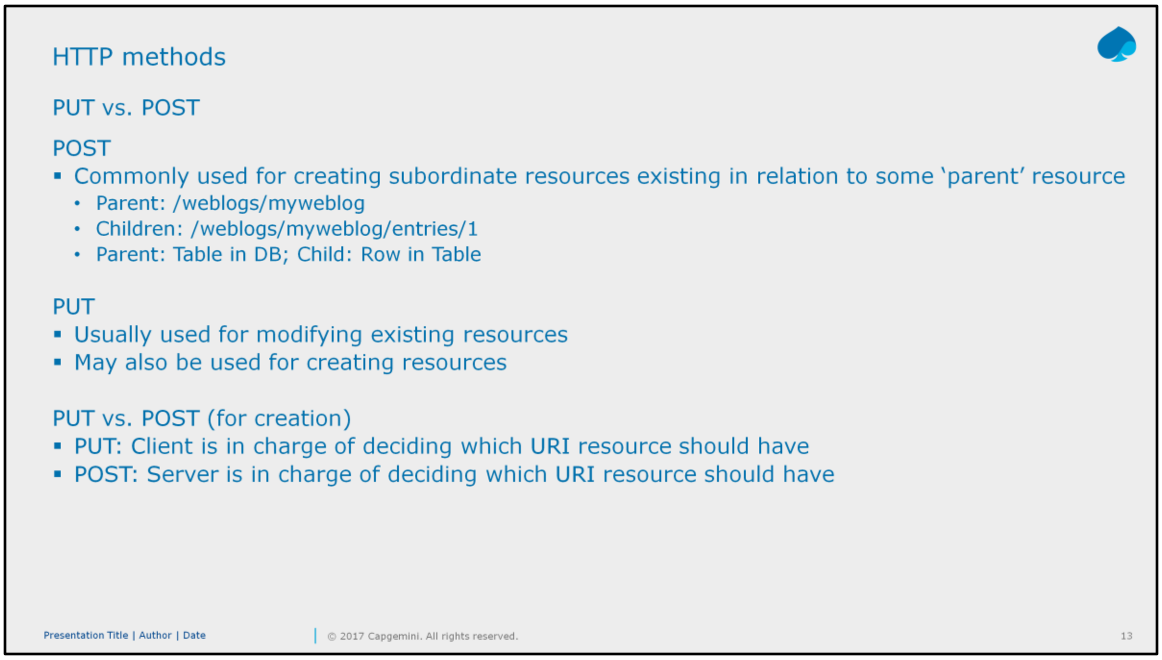


HTTP methods

POST method:

The POST method is used to create resources.
Typically, it is used to create resources under subcollections—resource collections that exist under a parent resource.

Unlike PUT, a POST request doesn’t need to know the URI of the resource. The server is responsible for assigning an ID to the resource and deciding the URI where the resource is going to reside.



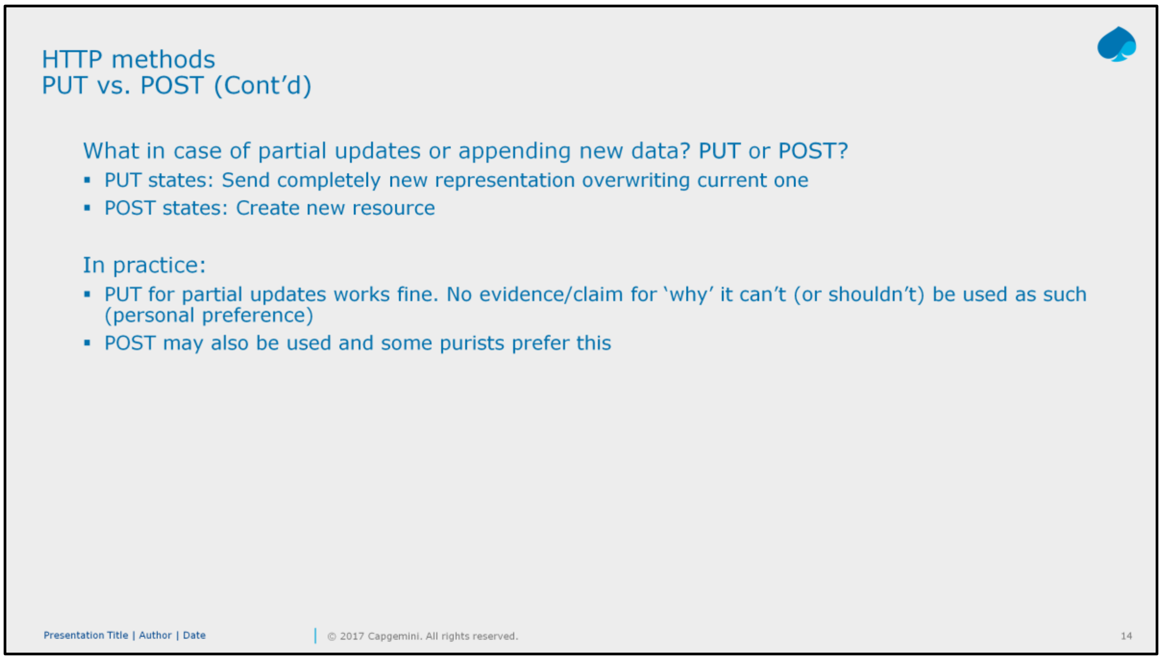
HTTP methods

PUT vs. POST

POST
- Commonly used for creating subordinate resources existing in relation to some 'parent' resource
  - Parent: /weblogs/myweblog
  - Children: /weblogs/myweblog/entries/1
  - Parent: Table in DB; Child: Row in Table

PUT
- Usually used for modifying existing resources
- May also be used for creating resources

PUT vs. POST (for creation)
- PUT: Client is in charge of deciding which URI resource should have
- POST: Server is in charge of deciding which URI resource should have



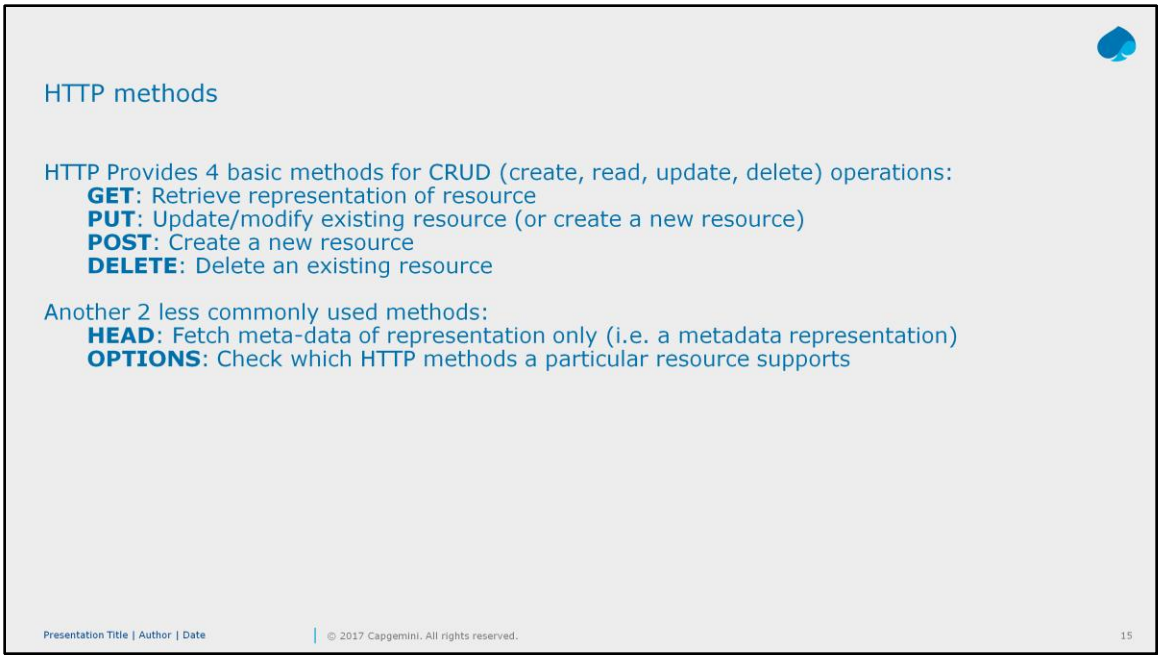
HTTP methods
PUT vs. POST (Cont’d)

What in case of partial updates or appending new data? PUT or POST?

- PUT states: Send completely new representation overwriting current one
- POST states: Create new resource

In practice:

- PUT for partial updates works fine. No evidence/claim for ‘why’ it can’t (or shouldn’t) be used as such (personal preference)
- POST may also be used and some purists prefer this



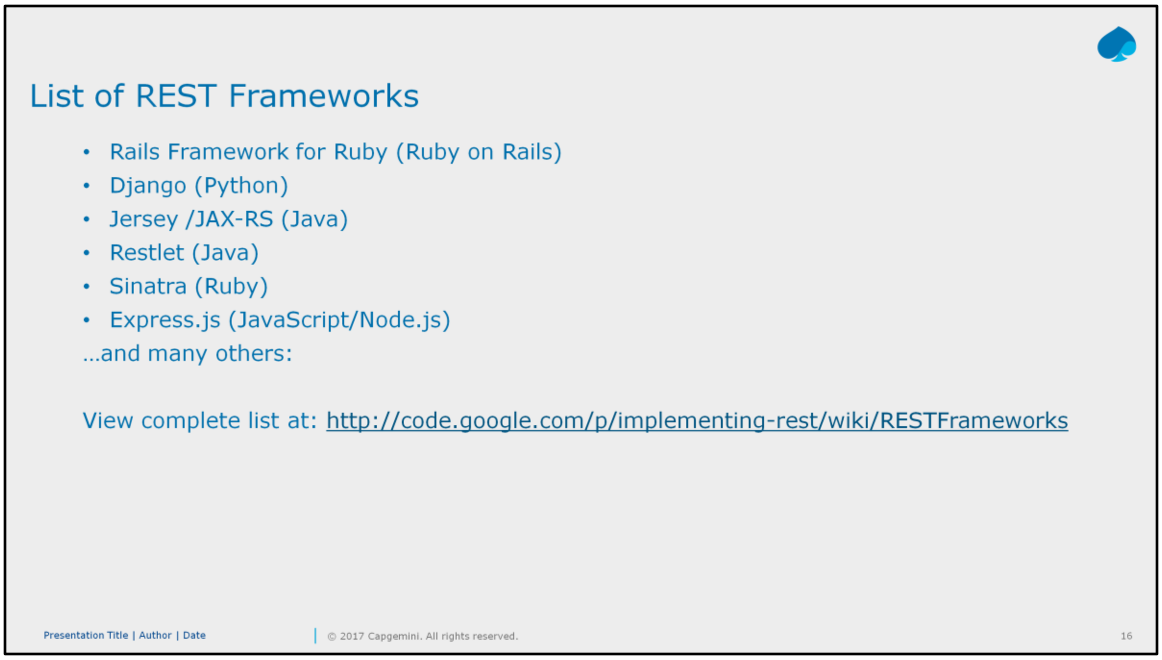
HTTP methods

HTTP Provides 4 basic methods for CRUD (create, read, update, delete) operations:

- **GET**: Retrieve representation of resource
- **PUT**: Update/modify existing resource (or create a new resource)
- **POST**: Create a new resource
- **DELETE**: Delete an existing resource

Another 2 less commonly used methods:

- **HEAD**: Fetch meta-data of representation only (i.e. a metadata representation)
- **OPTIONS**: Check which HTTP methods a particular resource supports

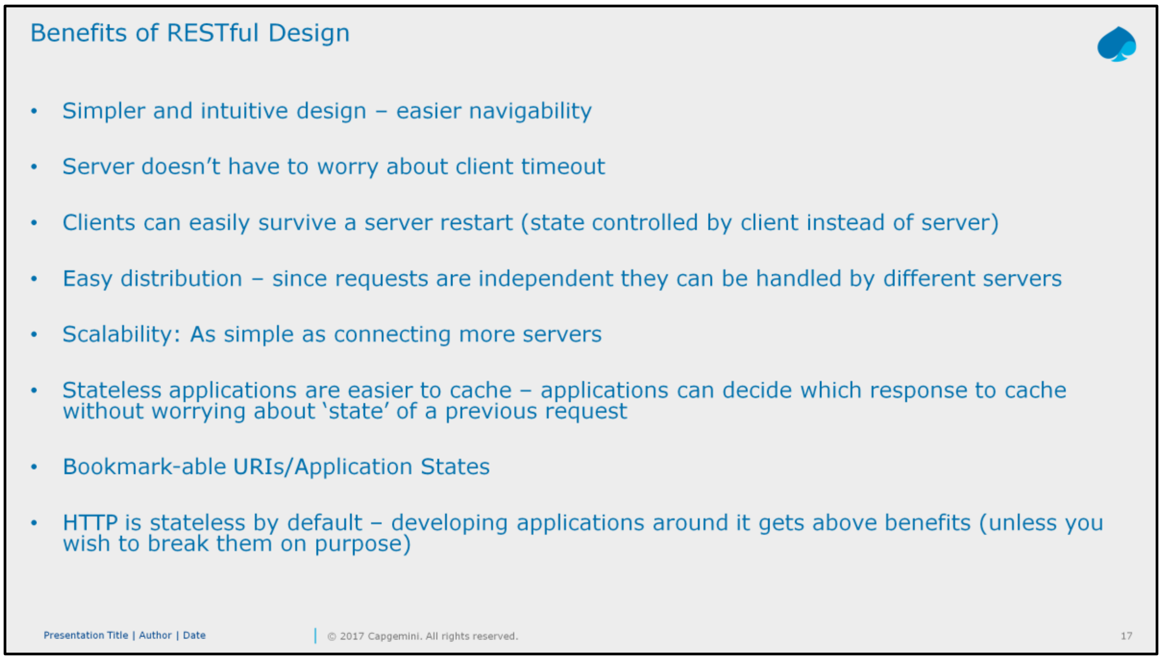


List of REST Frameworks

- Rails Framework for Ruby (Ruby on Rails)
- Django (Python)
- Jersey /JAX-RS (Java)
- Restlet (Java)
- Sinatra (Ruby)
- Express.js (JavaScript/Node.js)

…and many others:

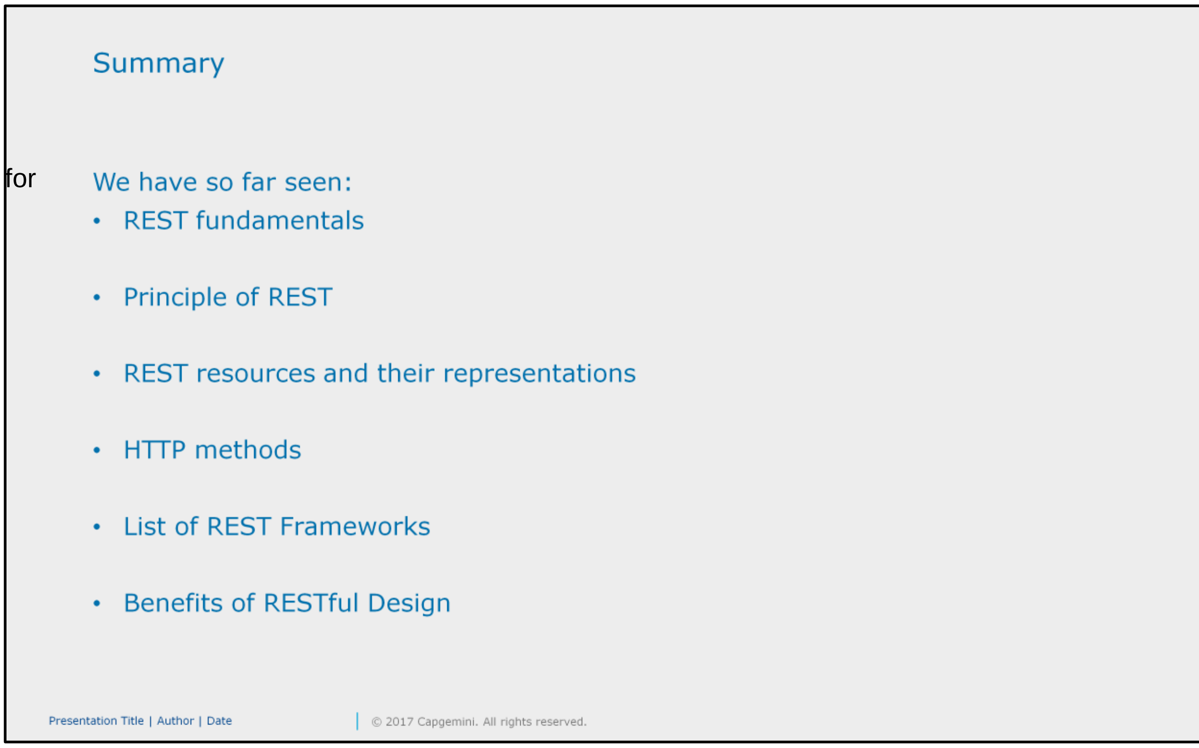
View complete list at: http://code.google.com/p/implementing-rest/wiki/RESTFrameworks



Benefits of RESTful Design

- Simpler and intuitive design – easier navigability
- Server doesn't have to worry about client timeout
- Clients can easily survive a server restart (state controlled by client instead of server)
- Easy distribution – since requests are independent they can be handled by different servers
- Scalability: As simple as connecting more servers
- Stateless applications are easier to cache – applications can decide which response to cache without worrying about 'state' of a previous request
- Bookmark-able URIs/Application States
- HTTP is stateless by default – developing applications around it gets above benefits (unless you wish to break them on purpose)

Additional notes for instructor



Summary

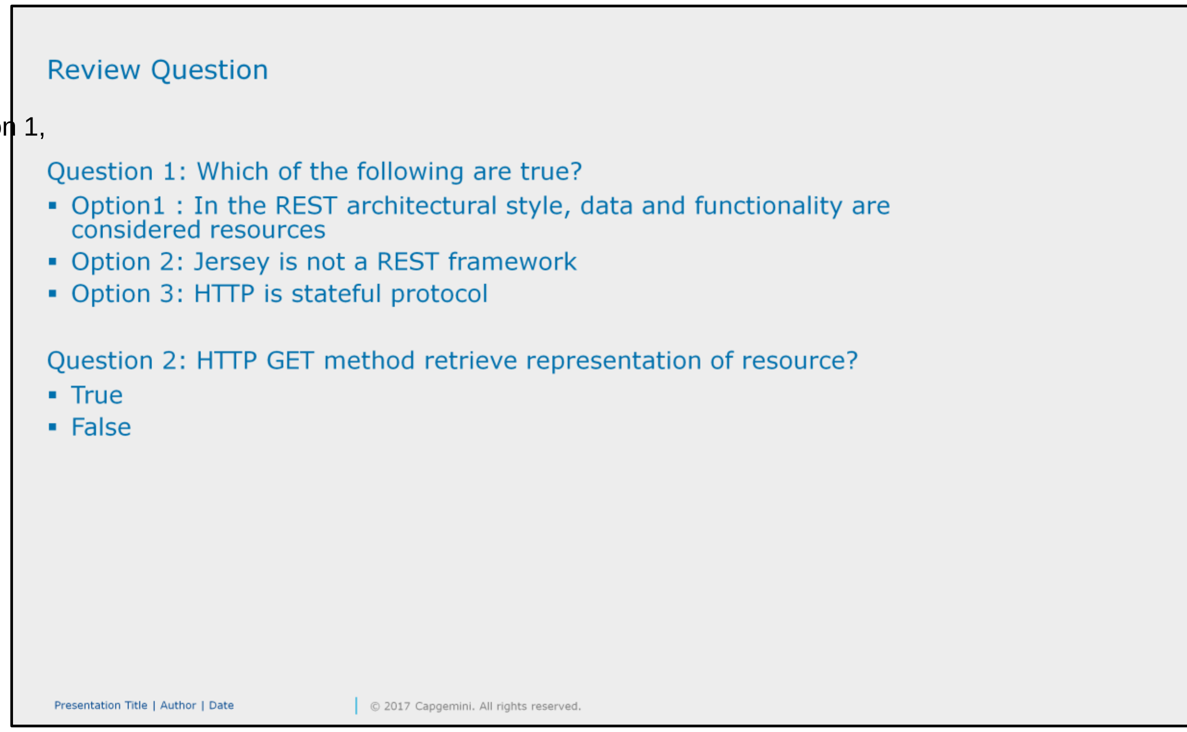
We have so far seen:

- REST fundamentals
- Principle of REST
- REST resources and their representations
- HTTP methods
- List of REST Frameworks
- Benefits of RESTful Design

AOP is a powerful complement to object-oriented programming. With aspects, you can now group application behavior that was once spread throughout your applications into reusable modules. You can then declaratively or programmatically define exactly where and how this behavior is applied. This reduces code duplication and lets your classes focus on their main functionality. Thus, the AOP technique helps you add design and run-time behavior to an object model in a non-obtrusive manner by using static and dynamic crosscutting.
The first thing that comes to mind when someone mentions AOP is logging, followed by transaction management. However, these are just special applications of AOP. AOP can be used for performance monitoring, call auditing, caching and error recovery too. More advanced uses of AOP include compile-time checks of architecture standards. For example, you can write an aspect that will enforce you to call only certain methods from certain classes.
Today, Spring AOP offers fine grained object advising capabilities using AspectJ support.

Question 1: Option 1,

Question 2: True



Review Question

Question 1: Which of the following are true?

- Option1 : In the REST architectural style, data and functionality are considered resources
- Option 2: Jersey is not a REST framework
- Option 3: HTTP is stateful protocol

Question 2: HTTP GET method retrieve representation of resource?

- True
- False

Add the notes here.