Introduction to REST & Spring

CST 365 – Web Applications
Michael Ruth, Ph.D.
Associate Professor
Computer Science & I.T.
mruth@roosevelt.edu

Objectives

- Explain the fundamentals of the Spring framework for developing Web applications (inversion of control)
- Discuss REST and REST Services
- Explain the architectural style of applications built using REST Services
- Discuss REST and HTTP Methods
- Explain JSON and its use as a medium

Full Stack Web Development

- One of the most important elements of modern full-stack development is Inversion of Control (IoC)
  - In traditional programming world, we develop objects that carry logic and data and the objects interact with each other to do the work
  - However, with IoC, we separate the logic from the data to loosen the coupling between the data and the logic involved
    - Basically, we'll define some objects and then define the program's logic separately…

IoC Framework

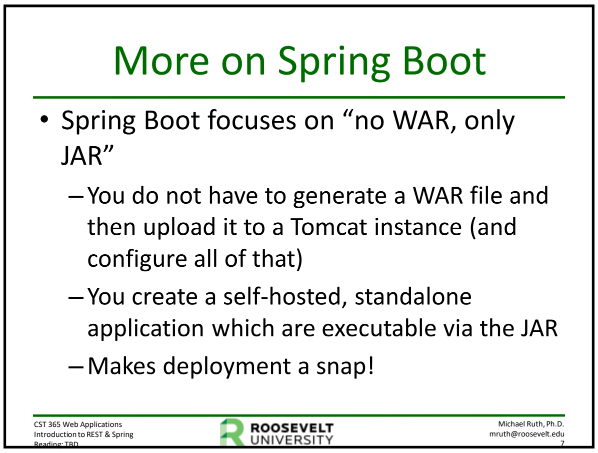
- Frameworks provide, among many other things, a IoC container which provides a consistent means of configuring and managing Java objects
  - We define the objects (but only the data)
    - The container is then responsible for managing object lifecycles of specific objects
    - Objects created/managed by the container are called managed objects (or managed beans)
    - We configure the container objects by either writing them in XML or annotating POJOs
    - We obtain them through dependency injection
      - Dependency injection is a pattern where the container passes objects by name to other objects, via either constructors, properties, or factory methods

Spring Framework

- Spring initially was just an IoC container, but now it's a application framework that allows you to build EE Java applications
- Spring also has a configuration module where Spring handles many common concerns such as handling HTTP requests, connecting to DBs, etc.
  - Allows the developer to focus on business services
    - We develop the business classes and annotate our classes with Spring annotations and Spring takes care of the details for us

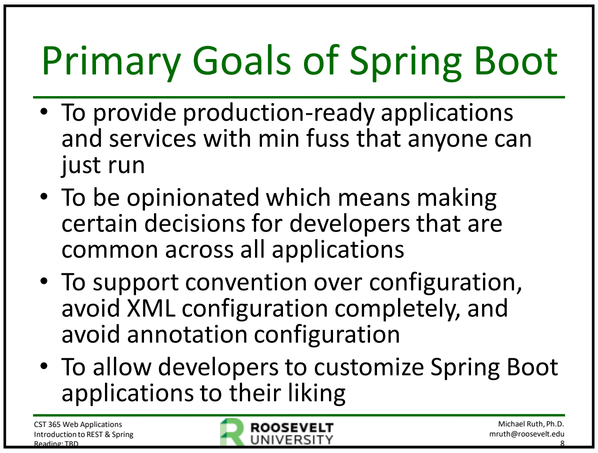
Spring Boot

- Spring is a huge framework that has several setup and configuration steps and several build and deploy steps
- Spring boot addresses these concerns and abstracts these steps and allows the developers that use it to focus on the business logic
  - Main aim is to address the complexity of configuration in the Spring framework by taking MOST of the work away



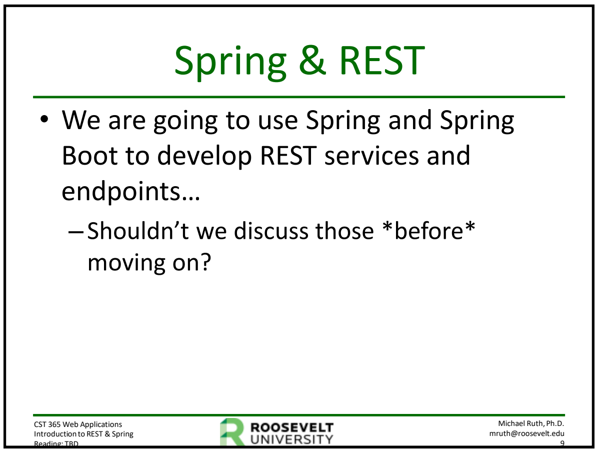
More on Spring Boot

- Spring Boot focuses on “no WAR, only JAR”
  - You do not have to generate a WAR file and then upload it to a Tomcat instance (and configure all of that)
  - You create a self-hosted, standalone application which are executable via the JAR
  - Makes deployment a snap!



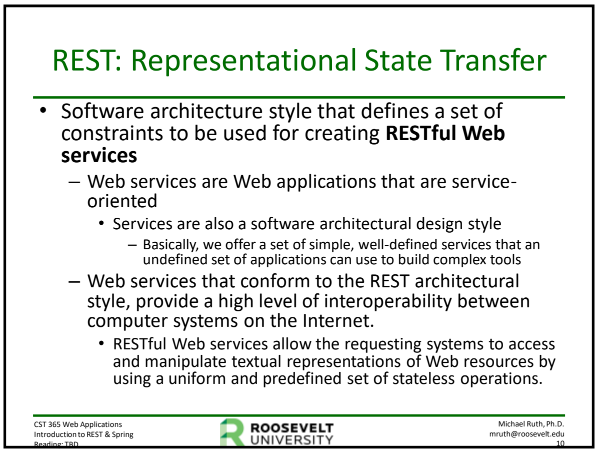
Primary Goals of Spring Boot

- To provide production-ready applications and services with min fuss that anyone can just run
- To be opinionated which means making certain decisions for developers that are common across all applications
- To support convention over configuration, avoid XML configuration completely, and avoid annotation configuration
- To allow developers to customize Spring Boot applications to their liking



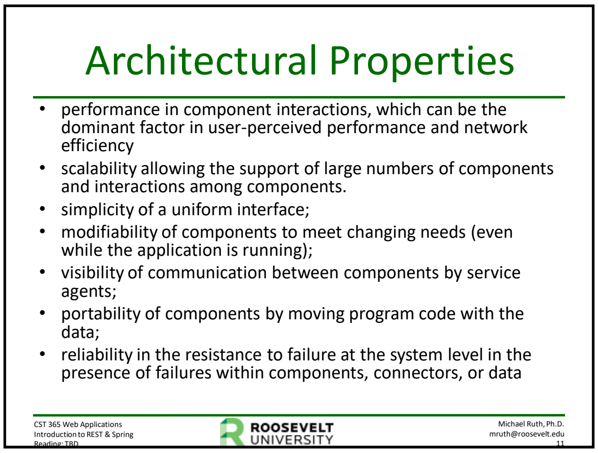
Spring & REST

- We are going to use Spring and Spring Boot to develop REST services and endpoints…
  - Shouldn’t we discuss those *before* moving on?



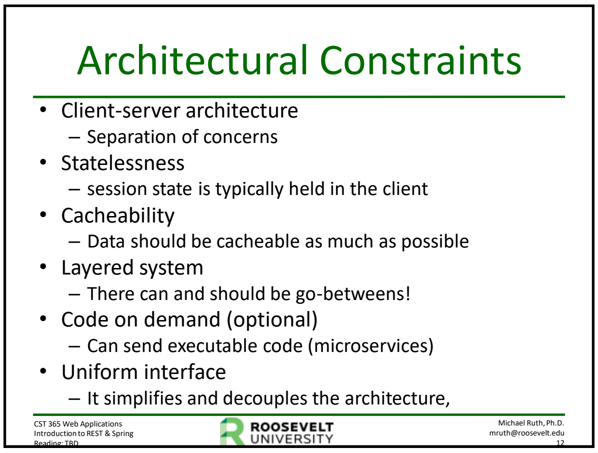
REST: Representational State Transfer

- Software architecture style that defines a set of constraints to be used for creating **RESTful Web services**
  - Web services are Web applications that are service-oriented
    - Services are also a software architectural design style
      - Basically, we offer a set of simple, well-defined services that an undefined set of applications can use to build complex tools
  - Web services that conform to the REST architectural style, provide a high level of interoperability between computer systems on the Internet.
    - RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.



Architectural Properties

- performance in component interactions, which can be the dominant factor in user-perceived performance and network efficiency
- scalability allowing the support of large numbers of components and interactions among components.
- simplicity of a uniform interface;
- modifiability of components to meet changing needs (even while the application is running);
- visibility of communication between components by service agents;
- portability of components by moving program code with the data;
- reliability in the resistance to failure at the system level in the presence of failures within components, connectors, or data



Architectural Constraints

- Client-server architecture
  - Separation of concerns
- Statelessness
  - session state is typically held in the client
- Cacheability
  - Data should be cacheable as much as possible
- Layered system
  - There can and should be go-betweens!
- Code on demand (optional)
  - Can send executable code (microservices)
- Uniform interface
  - It simplifies and decouples the architecture,

Uniform Interface

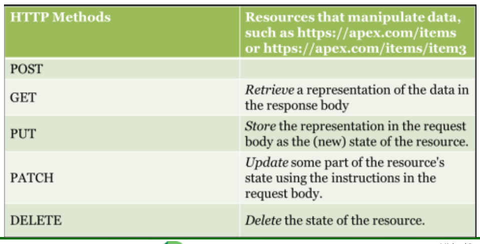
- Resource identification in requests
  - Individual resources are identified in requests
  - The resources themselves are conceptually separate from the representations that are returned to the client
- Resource manipulation through representations
  - When a client holds a representation of a resource, including any metadata attached, it has enough information to modify or delete the resource's state.
- Self-descriptive messages
  - Each message includes enough information to describe how to process the message
- Hypermedia as the engine of application state (HATEOAS)
  - Having accessed an initial URI for the REST application—analogous to a human Web user accessing the home page of a website—a REST client should then be able to use server-provided links dynamically to discover all the available resources it needs.

Web Service APIs

- Web service APIs that adhere to the REST architectural constraints are called RESTful APIs
- HTTP-based RESTful APIs are defined with the following aspects:
  - a base URI, such as http://api.example.com/collection/;
  - standard HTTP methods (e.g., GET, POST, PUT, PATCH and DELETE);
  - a media type that defines state transition data elements
    - The current representation tells the client how to compose requests for transitions to all the next available application states.
    - This could be as simple as a URI or as complex as a Java applet

URI & HTTP Methods (1)

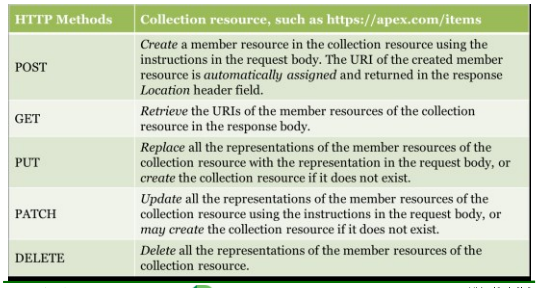
- The following table shows how HTTP methods are intended to be used in HTTP APIs, including RESTful ones.



HTTP Methods	Resources that manipulate data, such as https://apex.com/items or https://apex.com/items/item3
POST	
GET	*Retrieve* a representation of the data in the response body
PUT	*Store* the representation in the request body as the (new) state of the resource.
PATCH	*Update* some part of the resource's state using the instructions in the request body.
DELETE	*Delete* the state of the resource.

URI & HTTP Methods (2)

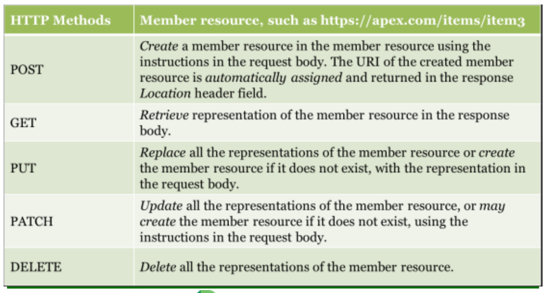
- How HTTP methods are typically used in REST APIs



HTTP Methods	Collection resource, such as https://apex.com/items
POST	*Create* a member resource in the collection resource using the instructions in the request body. The URI of the created member resource is *automatically assigned* and returned in the response *Location* header field.
GET	*Retrieve* the URIs of the member resources of the collection resource in the response body.
PUT	*Replace* all the representations of the member resources of the collection resource with the representation in the request body, or *create* the collection resource if it does not exist.
PATCH	*Update* all the representations of the member resources of the collection resource using the instructions in the request body, or *may create* the collection resource if it does not exist.
DELETE	*Delete* all the representations of the member resources of the collection resource.

URI & HTTP Methods (3)

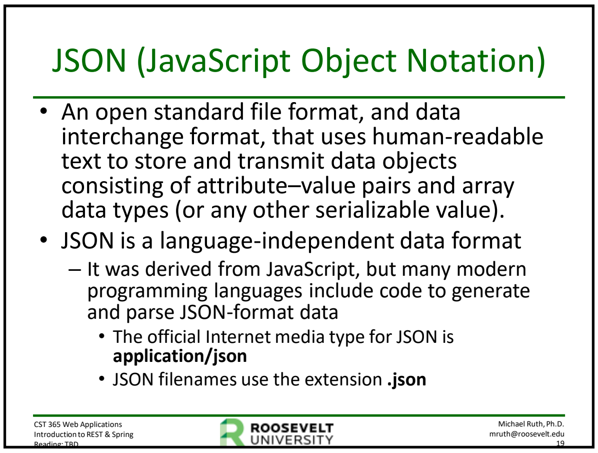
- how HTTP methods are typically used in REST APIs



HTTP Methods	Member resource, such as https://apex.com/items/item3
POST	*Create* a member resource in the member resource using the instructions in the request body. The URI of the created member resource is *automatically assigned* and returned in the response *Location* header field.
GET	*Retrieve* representation of the member resource in the response body.
PUT	*Replace* all the representations of the member resource or *create* the member resource if it does not exist, with the representation in the request body.
PATCH	*Update* all the representations of the member resource, or *may create* the member resource if it does not exist, using the instructions in the request body.
DELETE	*Delete* all the representations of the member resource.

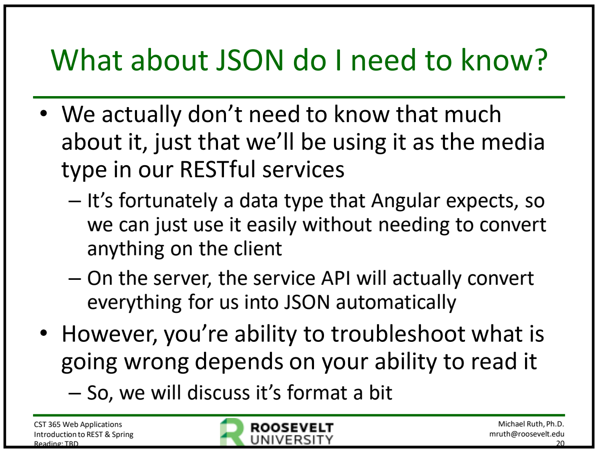
Media Type?

- So, the media type can be anything as long as it's textual
  - However, let's think realistically!
  - Can be code: Java, C, etc.
  - Can be XML
  - Can be JSON
    - most commonly used for Web applications



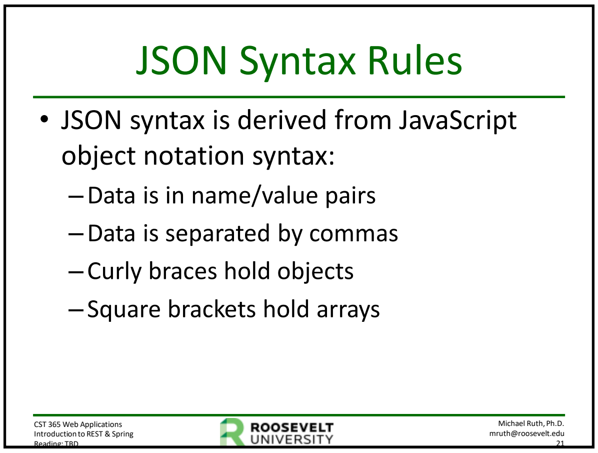
JSON (JavaScript Object Notation)

- An open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).
- JSON is a language-independent data format
  - It was derived from JavaScript, but many modern programming languages include code to generate and parse JSON-format data
    - The official Internet media type for JSON is **application/json**
    - JSON filenames use the extension **.json**



What about JSON do I need to know?

- We actually don't need to know that much about it, just that we'll be using it as the media type in our RESTful services
  - It's fortunately a data type that Angular expects, so we can just use it easily without needing to convert anything on the client
  - On the server, the service API will actually convert everything for us into JSON automatically
- However, you're ability to troubleshoot what is going wrong depends on your ability to read it
  - So, we will discuss it's format a bit



JSON Syntax Rules

- JSON syntax is derived from JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

name/value pairs

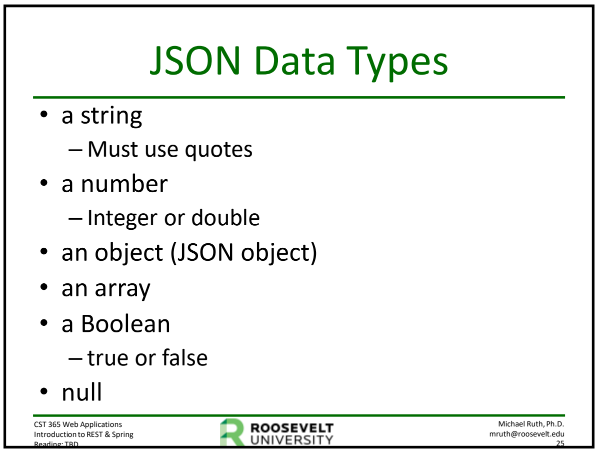
- JSON data is written as name/value pairs.
  - A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:
    - “name”:””Michael”
    - You can typically get away with things like:
      - name:”Michael”
      - name:’Michael’

Object (& commas)

- We can define an object with curly braces:
  - { name: “Mike", age: 45, city: “Chicago" };
    - The commas separate the fields that are being provided
    - The 3 fields together make up an object using Javascript notation
    - Objects can be given a name:
      - {
        “student":{ "name":“Mike", "age":45, "city":“Chicago" }
        }

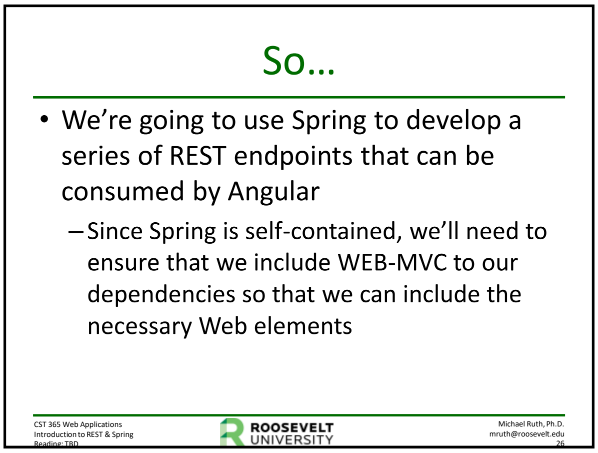
JSON Arrays

- We typically use arrays just like we might a scalar (single variable), but using square braces:
  - {
    “students":[“Anna", “Bob", “Carol"]
    }



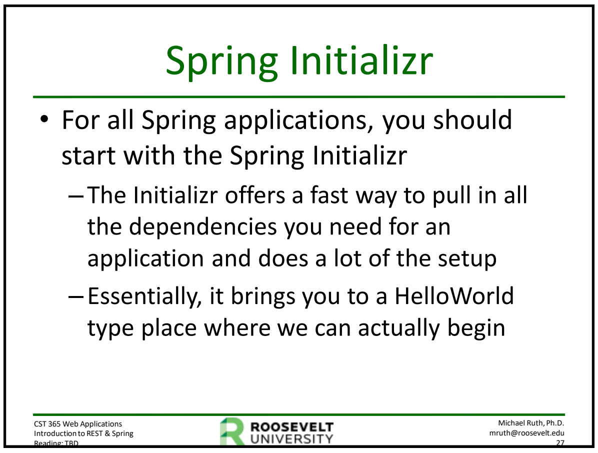
JSON Data Types

- a string
  - Must use quotes
- a number
  - Integer or double
- an object (JSON object)
- an array
- a Boolean
  - true or false
- null



So…

- We're going to use Spring to develop a series of REST endpoints that can be consumed by Angular
  - Since Spring is self-contained, we'll need to ensure that we include WEB-MVC to our dependencies so that we can include the necessary Web elements



Spring Initializr

- For all Spring applications, you should start with the Spring Initializr
  - The Initializr offers a fast way to pull in all the dependencies you need for an application and does a lot of the setup
  - Essentially, it brings you to a HelloWorld type place where we can actually begin

Summary

- Explained the fundamentals of the Spring framework for developing Web applications (inversion of control)
- Discussed REST and REST Services
- Explained the architectural style of applications built using REST Services
- Discussed REST and HTTP Methods
- Explained JSON and its use as a medium

For More Information

- For this presentation, I primarily used Wikipedia and Spring's main site for most of the details:
  - Inversion of Control:
    - https://en.wikipedia.org/wiki/Inversion_of_control
  - REST
    - https://en.wikipedia.org/wiki/Representational_state_transfer
  - JSON:
    - https://en.wikipedia.org/wiki/JSON
  - Spring:
    - https://spring.io/
  - Spring Guides:
    - https://spring.io/guides

Questions?