REST – Representation State Transfer

an architectural style consisting of a coordinated state of constraints applied to components, connectors, and data elements within a distributed hypermedia system

architectural properties-

* performance
* scalability of component interactions
* modifiability of components
* visibility of communication between components
* portability of components
* reliability

constraints

* client/server
* stateless
* cacheable
* layered systems
* code on demand (optional)

Into to MS and REST

* RESTful services isolates the application from direct manipulation
* limited operations (HTTP verbs)
* easier to code than SOAP
* stateless interaction
* HTTP requests
* Request headers –message to server as to what type of response is required
* request methods- GET POST PUT DELETE
* HTTP message
  + request method
  + URI – location of resource
  + header fields – optional
* API entry point
* Status codes
  + 200 – OKAY
  + 201 – CReated
  + 204 – No Content
  + 400 – Invalid Request
  + 404 – not found

OAuth – Open Authorization

* authorization protocol
* way for users to grant websites or applications access to their information onother websites without giving them their password
* aka Google will authorize you to use another website by logging into Google account
* Passport
  + middleware service for Node.js for implementing OAuth
  + uses strategies
    - strategies are a way to authenticate requests
    - each strategy has own npm package
    - must be configured before usage
* npm install cookie-session ejs express nodemon passport passport-google-oauth20
* npm I request //for API requests
* console.developers.google.com
  + API & Services- create new project
  + within project- Credentials-
  + create new credential - OAuth client ID
    - choose Web application
    - choose origin – URL of our site aka http://localhost:8000
    - redirect URI – what set in our ap aka <http://localhost:8000/auth/google/callback>

Module- in Node.js a module is a reusable piece of simple or complex functionality that you can organize into a single or multiple JS files. each module has own context (so it wont’ interfere with other modules or the global scope)

* core modules
* local modules
* third party modules
* var module = require(‘hapi’) (after installing hapi)

Returning Data with JSON- res.json({ ‘name’ : ‘Johnny’})

nodemon is a tool that helps Node.js applications by automatically restarting the app after changes (a wrapper for node)

Postman Test

pm.response.object ->

* pm.test(“Name is correct”, function() {
  + pm.expect(pm.response.json()[0].name).to.include(“first task”);
* });

Deplying APIs

npm i –g serverless

sls config credentials –provider aws –key asdflkdsafj –secret qwerasdf

* file folder

sls create –t aws-nodejs –n NAME

npm i serverless-http

* name.yml file

sls deploy

What makes architectural interface uniform in REST?

* a client with permission can modify any resource on the server
* each message contains enough info to describe how it should be processed
* clients an make state transitions only through hyperlinks within hypertext
* resources are identified as requests and data is sent using web-based representations

Resource – example Web API

* Base URI – <http://example.com> – indicates where the services can be found (API)
* web-supported Internet media type- usually JSON
* set of operations that use HTTP methods – GET PUT POST DELETE
* hypertext driven

Two types of resource URIs

* collection URI – <http://example.com>/resources
* element URI - <http://example.com/resources/item2>

Uniform Interface

* identification of resources – resources are identified as requests and data
* manipulation of resources through representations –a client can modify any resource on the server and should haave the permission to do so
* self-descriptive messages
* hypermedia as the engine of application state

Resource Representations

* plain text – easiest to display, hardest to consume
* form encoding – non-web service type
* HTML –difficult with web-service, good for others
* XML – easy to consume, best for programming languages
* JSON – popular for browser based and for JS

High-Level Overview of RESTful service

* client sends request
* server receives request and sends response
* response and request sent via HTTP method like GET
* response is sent using xml representation

Mapping URLs

REST

* promotes scalability
* supports general interface
* deploy independently
* enforce security
* reduce latency
* support legacy systems
* elastic
* resilient
* composable
* cohesive
* complete

Representational State Transfer

* proposed and originated by World Wide Web
* provides a standard for doing CRUD operations
* move away from dense XML requests into URLs
* relies on stateless client-server protocol
* applied to connectors and data elements
* enhances induced performance, scalability, simplicity, modifiability, visibility, portability, and reliability
* developed to identify existing problems, to compare alternative solutions, and to ensure protocol extensions
* built with a set of core restraints
* simpler clients
* Client-Server
  + uniform interface separating clients from servers
  + improves the portability of the client code
  + servers are simpler and scalable
* Stateless
  + client context is not stored on the server while processing requests
  + information required to service the request is contained in the request
  + session state is held in the client
* Cacheable
  + not client-server interactions are needed when the page is well-managed and cached
  + responses are cached by clients and intermediaries
  + only implicit or explicit definition of responses should exist
* Layered System
  + layered system was designed as the client was not able to figure out if it is connected to the end server
  + system scalability is improved by enabling load balancing and by providing shared caches
  + security policies are enforced
* Code-on-Demand
  + servers can extend the functionality of a client by transferring the executable code
  + compiled components (Java applets) and client-side scripts (JS) are examples of code-on-demand
  + optional constraint
* HTTP-based RESTful APIs are categorized with the following-
  + base URI
  + internet media type for the data
  + standard HTTP methods
  + hypertext links to reference state
  + hypertext links to reference-related resources

Architectural Properties of REST

* Performance – component interactions to be dominant for user-perceived performance and network efficiency
* Scalability – for supporting many components and interactions among components
* Simplicity of interfaces
* Modifiability of components to meet changing needs
* Visibility of communication to process the components by service agents
* Portability of components to move the program code with the data

|  |  |
| --- | --- |
| REST | SOAP |
| public APIs are exposed over the internet to handle CRUD operations on data | pieces of application logic are exposed as services with their own protocol |
| focuses on accessing named resources through a single consistent interface | focuses on accessing named operations |
| provides more web-based services because of the HTTP client-based behavior | wrongly referred to as “web services” |

REST is simpler, easier to understand, permits many different data formats, can be cached, and has better performance and scalability

SOAP has better security through supporting SSL and WS-Security, provides a standard implementation of data privacy and data integrity, and supports ACID transactions support with multi-phase commit across different distributed transactional resources

Apache CXF Framework

* open source framework for Services
* provides implementation for Java Service standards
* architecture-
  + front ends – for writing services
  + interceptors – custom behaviors on messaging
  + service model – defining a service hosted in CFX
  + bus – central, backbone of CFX, driving and giving access to resources
  + data bindings- converting between XML and Java
  + protocol bindings – message transport (SOAP, REST, XML, etc)
  + transport = how messages are sent/received

Maven

* software development automation
* it orders what happens as part of development
* builds software into deployable code
* runs tests on code
* generates reports and information
* realizes continuous integration
* Building
  + manages software dependencies
  + executes the compilation and bundling of software
  + uses conventions to simplify the work in getting a project set up
  + allows for full customization
* Process
  + ensures code is compiled and configured to the latest
  + makes it easier to deploy and execute complex software
  + runs unit tests as software is built
  + whatever you tell it to
* Modes
  + command line – code where you wish, maven does the rest, including builds
  + integrated development environment IDE – works within the IDE tools to make common builds on servers and tools
* Project Object Model – POM file
  + defines the work Maven does
  + what dependencies it needs
  + what ‘tasks’ you can ask it to do
  + how it does those tasks
  + Standards simplify
    - many projects can use a common POM with a few tweaks
    - or use an archetype to setup a POM with your needed elements
    - fully customizable
* Quick Start up
* Rigorous Standards (consistent)

CXF Bus

* backbone of Apache CXF
* point of access for all services
* provides a place to customize- WSDL Managers, Factory Classes, HTTP Destination Details
  + these areas are tweaked using ocnfig files eg META-INF/cxf/cfx.xml
* can configure the bus to customize all features- logging, failover, load distribution, etc

CXF Data Bindings

* the key to trading between our incoming and outgoing data
* data is binary on the wire, text in XML format, want to convert to object for Java
* takes in XML as input and transforms it to an object
  + as a Stream for SAX processing or Node for DOM processing
  + must supply mapping and direct it to a class
* the configure binding realizes the reader and writer and moves between XML and Objects
* configuration swaps all this out for us
* Data Binding Configuration is given its own tag
  + databinding tag points to the bean that realizes the binding
  + can be configured on the client or server
* Basic Data Binding Options
  + JAXB – Java standardized XML mapping
  + Aegis = included as part of CXF as an alternative
    - includes own Annotations
    - allows for custom mapping
  + MTOM – Message Transmission Optimization Mechanism
    - allows binary transmission of data
    - makes SOAP more efficient on the wire
    - easier attachments/files
  + JSON – allows data to be sent as JS objects
    - often used in REST services
    - makes web clients easier to access infomation

CXF Transports

* traditional web options
  + HTTP/HTTPS
    - client
    - server
  + Asynchronous HTTP
  + Web Server Specific
* alternative options
  + JMS Java Messaging Service
  + UPD – multi-broadcast
  + local – in same JVM
  + socket – direct connection
* the transporter is plugged in as an interceptor
* will set up all sorts of different things like listening to the messages, routing data, etc
* part of a lot of features
* default on HTTP

Apache Tomcat

* popular web server

Client Creation

* create bean with an ID
* specific the interface URL
* point to the interface class

WSDL Web Service Definition Language

* defines how 2 programs will communicate
  + what is the structure of data being shared
  + what functionality is being request
  + what are the inputs/outputs
* WSDL is written in XML defining how XML messages are constructed
  + shows the list of all things that can be request from that service
  + defines or refers to all data types
  + defines a specific hierarchy to define features for reuse
* WSDL parts
  + type
  + message
  + operation
  + port type
  + binding
  + port
  + service

WSDL parts

* Type
  + defines the data times that can be parts of a message
    - objects in your model
    - ideally common across many services
  + defined in the WSLD or in separate XSD
* Data Message
  + an organization of the Types into a specific request/response
    - what gets sent to the service as the inputs and request data for the message
    - what is returned from the message
    - largely using the types already defined but customized
* Port Type
  + defines the operation
    - as a paired set of inputs and outputs given a name
    - not an actual service that can be called
    - like Java Interface, where you know how you interact, but need a specific object to interact with
* Binding
  + defines the implementers
    - takes the port type and realizes it into a service
    - defines the SOAP needs
    - places the service at a URL
    - allows for a service to be called/used in multiple different deployable configurations
* Port
  + places the binding at a specific network address
    - could be multiple versions or even parallel versions for different purposes
    - we finally define a specific instance
* Service
  + a collection of ports
    - we don’t want to deal with calls individually
    - we release a portfolio of specific services
    - the client binds to this and calls all it wants as needed
* You can define messages
  + one way (outputs)
  + synch or async
  + define faults – possible errors
  + with specific parameter order
* advanced features
  + extensions for SOAP bindings
  + specific HTTP elements and data
  + use multiple MIME types
* built to be very flexible
* usually only build at start of project, might tweak later

XML Schemas

* defining data is the core part of modeling systems
  + need to know what information is incoming and available
  + functionality revolves around manipulating data
  + need to define it with as much precision as possible
* XML gives robust modeling
* highly descriptive and readable/writable by people
* XML Schema Definition - XSD
* XSD provides the ‘mold’ for XML data
  + defines a type(criteria) for XML structures to meet
  + gives rules to define if all the data is present and valid
* XSD elements
  + defines what data items make up the document
  + structure of elements
    - nested items
    - order of elements
    - optional items, required items, or count limits
  + attributes of elements for further data
* XSD Values
  + defines types of data for individual elements (text, number, date, Boolean)
  + specific values- default and fixed

REST

* utilizes a web sites’ URI
  + the path points to rest and then the type of data within
  + we can navigate to data without complex and expensive XML
* use HTTP methods
* JS can easily map to REST
* dynamic content can focus on URLs

JAX-RS Jakarta RESTful Web Services

* provides support in creating web services according to the REST architectural pattern
* a REST data type maps to a class that provides that URL
* each operation then maps to a method
* the linking of two technologies
* to bind the REST elements with Java we use annotations
  + they are metadata on top of the code
  + the Annotations tell the implementing framework where to plug in REST needs
  + @Path – identifies a REST Enabled Bean
  + @GET @POST @PUT @DELETE
  + @Consumes – defines the incoming data type to the service
  + @Produces- defines the outgoing data type from the service
* HTTP Elements can be accessed to ruther customize our services
  + use an annotation to identify the data we want
  + we bind to a parameter into our method
  + includes- form data, cookies, matrix data
* Custom Handlers
  + handling the HTML HEAD
  + @HEAD annotation binds your action if a HEAD request is sent
  + @Provider changes how JAX-RS handle that MIME type when received in the implementation
* JAX-RS is not an implementation, it defines annotations and binds REST to business logic
* we need a provider (Apache CXF or Spring)
* @XmlRootElement(name = “asdf”)
* Invocation.Builder
* JAXRSServerFactoryBean sf = new JAXRSServerFactoryBean(); // creates server
* sf.setResourceClasses(AutoService.class, BikeService.class);
* sf.sefAddress(“http://localhost:9000/”);
* sf.create()

JSON Message Support

* $.getJSON(allAutosRest, {}) // jquery
* @Produces({“application/json”}) // annotation
* @Produces({“mutlipart/mixed”});

Multipurpose Internet Mail Extensions (MIME) – standard that indicates the nature and format of a document or file. structure is type/subtype aka “multipart/mixed”

HTTP-based services

* services designed to transport data over the Web using native protocols such as
  + TCP/IP, HTTP, XML, SOAP, JSON
* browser issues GET request, retrieving HTML
* browser looks at HTML and determines if there any other files that need to be requested
* browser retrieves any files referenced in HTML
* browser renders everything

ASP.NET WEB API

* all-in-one solution for developing, hosting, consuming, and testing HTTP-based services using ASP.NET
* Richardson Maturity Model
* HTTP services, different URLs, HTTP verbs, hypermedia

HTTP Status Codes

* 100 – Informational: used to provide informational responses (not common)
* 200 – Success: provides information for successful, completed, or accepted actions
  + 200 OK
  + 201 Created
  + 202 Accepted
* 300 – Redirection: used when the client has to do something else, usually a redirect
* 400 – Client Error: used when you or the client did something wrong
  + 401 – unauthorized
  + 404 – not found
* 500 – Server Error: used when the server did something wrong

OData

* odata.org
* for the standards of web requests
* has filter expressions and other info

JAXP artifact is XML to object mapping

* @Bean
* public XsdSchema countriesSchema(){
  + return new SimpleXsdSchema(new ClassPathResource(“countries.xsd”)); }

Richardson Maturity Model

* level 3: hypermedia communication (uses hypermedia to make service granular and portable)
* level 2: HTTP verbs-based communication
* level 1: resource based communication
* level 0: service end point communication

Active MQ Messaging

* dependency in spring boot
* application.properties or application.yml

Minimum Start for MS

* Eureka Discovery, Eureka Server, and Web dependencies from Spring
* configure application.properties file for 1st MS
  + eureka.instance.hostname=localhost
  + eureka.client.register-with-eureka=false (automatically true, but this is the discoverer)
  + eureka.client.fetch-registry=false (same as above)
  + server.port=6767
* @EnableEurekaServer to main method, above @Spring annotation
* use Spring again, with E Discovery, Server, and Web
* modify application.properties file TO application.yml
* application.yml file
  + server:
    - port: