Java Academy - 2022

Topic 8: REST Application Design with Spring Boot

REST Basics

Concepts

- What is REST?
  - Stands for REpresentational State Transfer.
  - Architectural style for loosely coupled applications over HTTP.
  - Presented first time in 2000 by Roy Fielding in his PhD dissertation.
  - Has guiding principles & constraints
    - Must be satisfied to call a REST API a RESTFul.
  - Based on ***HTTP*** methods to manage ***resources***
- Benefits
  - Simple & Scalable, easy to understand and implement
  - Faster data interchange (Uses JSON which is more compact and smaller than XML). Is able to respond with different formats
  - Easy to modify

Reading / Viewing Material

- Why REST What is an API
- REST Architectural Constraints
- Representational State Transfer

REST Basics (Cont.)

Concepts

- What is Resource?
  - The key abstraction of information.
  - Any information that we can name could be a rest resource.
  - The state of a resource is known as resource representation consisting of
    - Data
    - Metadata
    - Hypermedia links
- Why REST?
  - Easy to understand and implement
  - Scalability & Flexibility

Reading / Viewing Material

- Why REST
- Why REST API is important
- Resources
- SOAP vs REST

HTTP Basics

Concepts

- What is HTTP?
  - Stands for Hypertext Transfer Protocol
  - Application-layer protocol for communicating between distributed systems, and is the foundation of the modern web
- URLs
  - Uniform Resource Locators (URLs)
  - Contains protocol (http / https), domain/host, port, resource path, query
    - http://www.domain.com:8010/path/to/resource?attribute=value&attribute2=value
- HTTP Verbs
  - Fetch Resource(GET), Create Resource(POST),Full Update Resource(PUT),Partial Update Resource(PATCH), Delete Resource(DELETE),
- HTTP Status codes
  - 2xx - Successful
  - 4xx - Client Error
  - 5xx - Server Error

Reading / Viewing Material

- HTTP, protocol, verbs and codes

REST Design Example

Context

- You have been tasked with the design of an application and we need manage all the items we sell in physical stores and customization possibilities we offer as suppliers: Users, Accounts, Network devices, Professional Services, etc.

Resources

- To keep it simple we are going to think about ***devices*** and possibles ***configurations*** for them.
    - Use nouns to represent resources instead of actions.
    - Some resources can be sub-resources for others. A device can have multiple configuration options.
    - Identify an attribute that uniquely identifies the resource, this case could be an ID

URIs

- With resources defined, we need to define URIs. Focus on relationships and sub-resources. Should be nouns-only
    - A configuration is specific to a resource, so the URIs can be:
        - /devices ⟶ Represents all devices
        - /devices/{id} ⟶ Represents an specific device
        - /configurations ⟶ Represents all configurations
        - /configurations/{id} ⟶ Represents an specific configuration
        - /devices/{id}/configurations ⟶ Represents all possible configurations for a specific device
        - /devices/{id}/configurations/{id} ⟶ Represents a specific configuration for a specific device

REST Design Example (Cont.)

Resource Representations

- Once we have the URIs, let's define how they will be represented. We can use XML, JSON, or YAML. Can be represented as single resources or a collection of them.

```
<devices size="2">

    <link rel="self" href="/devices"/>

    <device id="12345">
        <link rel="self" href="/devices/12345"/>
        <deviceFamily>apple-es</deviceFamily>
        <OSVersion>10.3R2.11</OSVersion>
        <platform>SRX100B</platform>
        <serialNumber>32423457</serialNumber>
        <connectionStatus>up</connectionStatus>
        <ipAddr>192.168.21.9</ipAddr>
        <name>apple-srx_200</name>
        <status>active</status>
    </device>

    <device id="556677">
        <link rel="self" href="/devices/556677"/>
        <deviceFamily>apple-es</deviceFamily>
        <OSVersion>10.3R2.11</OSVersion>
        <platform>SRX100B</platform>
        <serialNumber>6453534</serialNumber>
        <connectionStatus>up</connectionStatus>
        <ipAddr>192.168.20.23</ipAddr>
        <name>apple-srx_200</name>
        <status>active</status>
    </device>

</devices>
```

```
<device id="12345">
    <link rel="self" href="/devices/12345"/>

    <id>12345</id>
    <deviceFamily>apple-es</deviceFamily>
    <OSVersion>10.0R2.10</OSVersion>
    <platform>SRX100-LM</platform>
    <serialNumber>32423457</serialNumber>
    <name>apple-srx_100_lehar</name>
    <hostName>apple-srx_100_lehar</hostName>
    <ipAddr>192.168.21.9</ipAddr>
    <status>active</status>

    <configurations size="2">
        <link rel="self" href="/configurations" />

        <configuration id="42342">
            <link rel="self" href="/configurations/42342" />
        </configuration>

        <configuration id="675675">
            <link rel="self" href="/configurations/675675" />
        </configuration>
    </configurations>

    <method href="/devices/12345/exec-rpc" rel="rpc"/>
    <method href="/devices/12345/synch-config"rel="synch device configuration"/>
</device>
```

REST Design Example (Cont.)

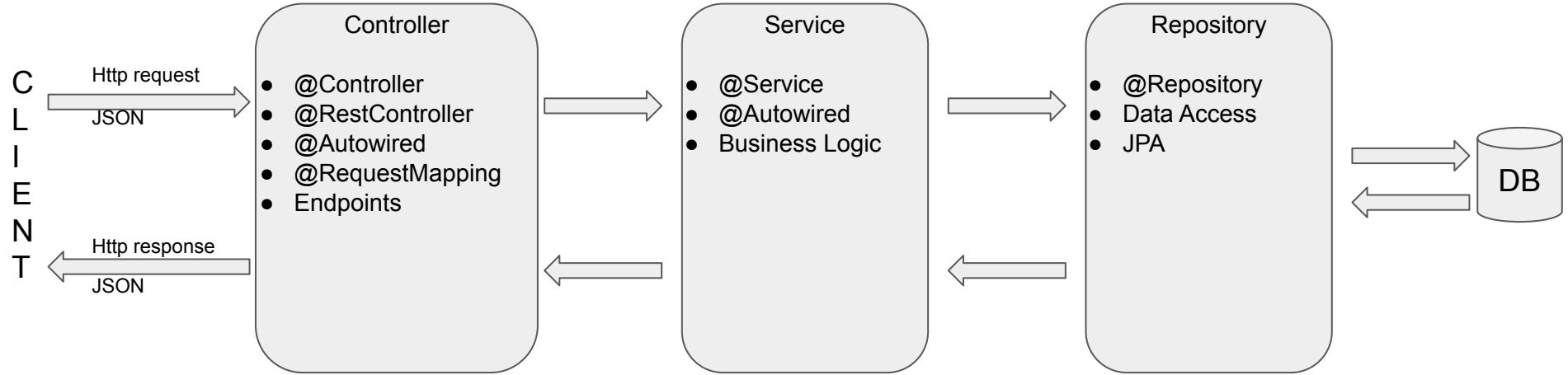
Assign HTTP Methods

- Now let's decide all possible operations for our API and map operations to resources
  - Retrieval of resources
    - HTTP GET /devices — Retrieves all devices information
    - HTTP GET /configurations — Retrieves all configurations information
    - HTTP GET /devices/{id} — Retrieves all information for the specific device id
    - HTTP GET /devices/{id}/configurations — Retrieves all configurations for the specific device id
    - HTTP GET /devices/{id}/configurations/{id} — Retrieves all information of specific configuration for a specific device id
    - HTTP GET /configurations/{id} — Retrieves all information for the specific configuration id
  - Create new resource
    - HTTP POST /devices — Creates new device. Information is sent as payload.
    - HTTP POST /configurations — Creates new configuration. Information is sent as payload.
  - Update a resource
    - HTTP PUT /device/{id} — Updates an specific device id
  - Delete a resource
    - HTTP DELETE /device/{id} — Deletes an specific device id

Spring Boot

Concepts

- Spring Boot
- Implementing REST with Spring Boot

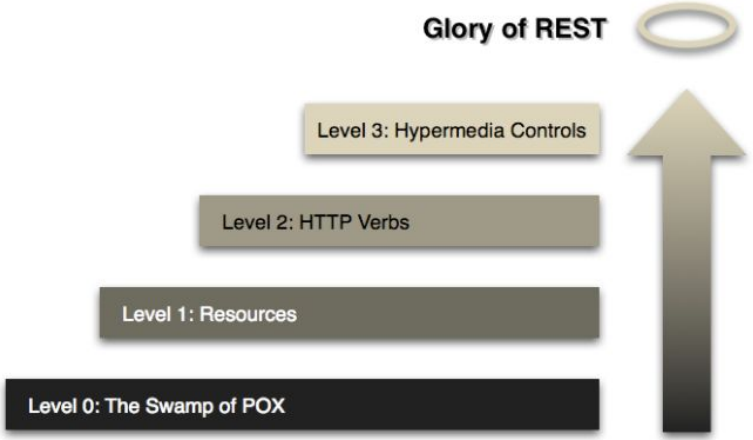


Reading / Viewing Material

- Simple Spring Boot Example
- Difference between @Controller, @Service, @Repository
- @RestController vs @Controller

Richardson Maturity Model (RMM)

Concepts



Taken from: https://martinfowler.com/articles/richardsonMaturityModel.html

- Richardson Maturity Model (RMM)
  - Model for restful maturity. Can be a way to understand concepts
  - REST Approach broken in 3 steps:
    - Resources
    - Verbs
    - Hypermedia controls
  - Level 0: It's like RPC Call. Talking to global endpoint
  - Level 1: Resource Introduction. Start talking to individual resources
  - Level 2: Use HTTP Verbs depending on the interaction with the service
  - Level 3: HATEOAS. Telling what can do next with the resource. Hypermedia controls tell how to do next interaction.

- See an example in the next slide.

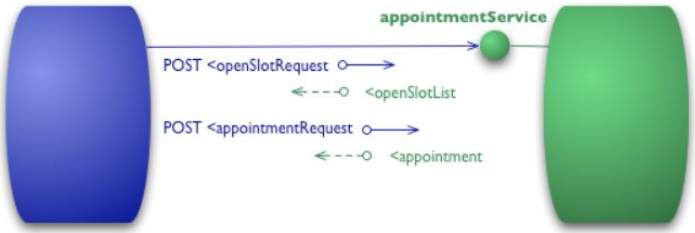
Reading / Viewing Material

- RMM

RMM - Example

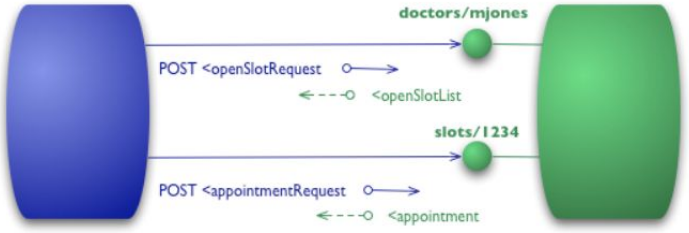
Context: Application to book appointments with Doctor.

Level 0



- Client talk to a single exposed service: appointmentService
- Retrieval of appointment information & availability using same method

Level 1



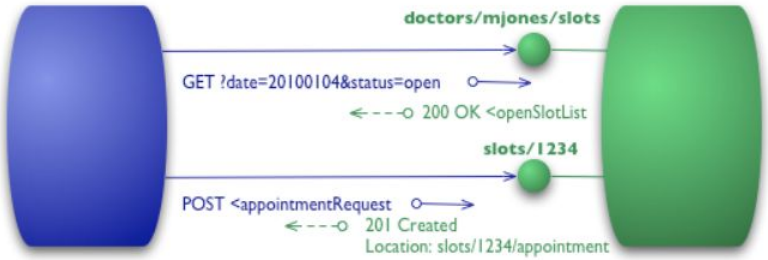
- Resources identified: doctor, slots
- Request sent to specific resources to retrieve information or book appointments.

Images taken from: https://martinfowler.com/articles/richardsonMaturityModel.html

RMM - Example (Cont.)

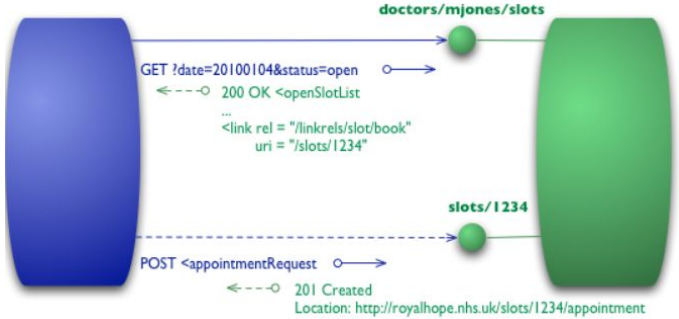
Context: Application to book appointments with Doctor.

Level 2



- Client talk to specific resource with specific verb/method to interact with resource
- GET will retrieve resource information (doctor's availability - slots-)
- POST will create new resource (new appointment)

Level 3



- Client talk to specific resource with specific verb/method to interact with resource
- GET will retrieve resource information (doctor's availability - slots-) and in addition how to book an appointment (hyperlink)
- POST will create new resource (new appointment) using the link sent in the response of the GET request.

Images taken from: https://martinfowler.com/articles/richardsonMaturityModel.html

REST Architectural Style

Concepts

- REST Architectural constraints
  - Uniform Interface: uniform interface includes using standard HTTP verbs to perform operations on resources.
    - Resource-Based: Individual resource identified in request
    - Resource manipulation through representation. Resource contains enough information to modify or delete it.
    - Self-descriptive messages: Each message include enough information for the server to process it.
    - HATEOAS: Hypermedia as the engine of application state. Response include links so client can discover other resources and interactions.
  - Client-Server: Client request resources without concerns with data storage. Server holds resources. Both evolve independently without knowledge of each other.
  - Stateless: Server will not store resource information in the session. Client must include all information required to fulfil request.
  - Cacheable: Responses should include whether it is cacheable or not and how long can be cached.
  - Layered System: Multiple layers can be used.
  - Code on Demand: Optional feature. Response can include executable code such as client-side scripts as JavaScript

Reading / Viewing Material

- REST Architectural Constraints

REST Architectural Style

Concepts

- REST Rules
  - Based on resource. Example: /api/users
  - HTTP Verbs used to identify the action
    - GET, POST, PUT, DELETE, PATCH
    - Used to modify resources
  - Send proper HTTP code to indicate success or failure.
  - Use Plural for resource naming

Reading / Viewing Material

- REST Architectural Constraints
- How to design REST API

URI	HTTP verb	Description
api/users	GET	Get all users
api/users/new	GET	Show form for adding new user
api/users	POST	Add a user
api/users/1	PUT	Update a user with id = 1
api/users/1/edit	GET	Show edit form for user with id = 1
api/users/1	DELETE	Delete a user with id = 1
api/users/1	GET	Get a user with id = 1

Taken from: https://www.geeksforgeeks.org/rest-api-architectural-constraints/

REST API Design Approach

Basic Steps

- Identify Resources (Object Modeling). REST services are based on resources which are any kind of data that can be accessed by client
- Create Model URIs. Each resource should be identified by a unique URI.
- Determine Resource Representations
  - JSON / YAML
  - Hypermedia links required? (HATEOAS Approach)
- Assigning HTTP Methods
- Version the API

Reading / Viewing Material

- REST Best Practices
- Best Practices #2

Other actions

- Depending on the overall system architecture, some additional topics should be considered for each service:
  - Logging?
  - Security?
  - Service Discovery? Re-Try?
  - Service Dockerization?
  - Service Documentation? (Swagger / OpenAPI Spec)

REST API Versioning

Concepts

- APIs Evolve and change in time. So versioning the API is important.
- Versioning is a way to manage changes to the API
- Required when we have breaking changes like:
  - Change in response format
  - Chante in request or response type
  - Removing parts of the API
- Strategies
  - URI versioning → http://api.example.com/v1
  - Versioning using custom request header → Accept-version: v1
  - Versioning using request header → Accept: application/vnd.example.v1
  - Versioning using query parameters → http://api.example.com/devices?version=1

Reading / Viewing Material

- Versioning Versioning 2 Versioning 3

Based or taken from: https://swagger.io/docs/specification/about/

REST API Security

Concepts

- API Keys
- OAuth
- JWT

Reading / Viewing Material

- Security Essentials
- Securing REST APIs
- Spring Boot OAuth
- Spring Boot OAuth 2

Based or taken from: https://swagger.io/docs/specification/about/

Documentation with Swagger

Concepts

- OpenAPI Specification (formerly Swagger Specification) is an API description format for REST APIs. An OpenAPI file allows you to describe your entire API, including:
    - Available endpoints (/users) and operations on each endpoint (GET /users, POST /users)
    - Operation parameters Input and output for each operation
    - Authentication methods. Contact information, license, terms of use and other information.
- API specifications can be written in YAML or JSON. The format is easy to learn and readable to both humans and machines
- Swagger is a set of open-source tools built around the OpenAPI Specification that can help you design, build, document and consume REST APIs.

Reading / Viewing Material

- About Swagger
- Adding Swagger Documentation
- Swagger / Open API Spec

Based or taken from: https://swagger.io/docs/specification/about/

Docker

Concepts

- Docker is an open platform for developing, shipping, and running applications
  - Enables you to separate your applications from your infrastructure so you can deliver software quickly
  - An image is a read-only template with instructions for creating a Docker container
  - A container is a runnable instance of an image
  - Docker Compose is a tool that was developed to help define and share multi-container applications. With Compose, we can create a YAML file to define the services and with a single command, can spin everything up or tear it all down.

Reading / Viewing Material

- What's docker?
- Spring Boot with docker
- REST API with docker compose

Based or taken from: https://docs.docker.com/get-started/

Testing Service with Postman

Concepts

- You can test your API using this tool Postman. Easily create different request to your APIs.

Reading / Viewing Material

- Postman Site

Exercise

Description

Practice REST with Spring Boot by doing!

Requirements.

- Attached to the learning module you will find an API document. Please create a REST API that implements the spec..
- Deadline for exercise: March 25th, 2022.