Internet Applications Design and Implementation

(Lecture 3 - Server side programming, RESTful APIs)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Outline

- The architectural style REST to instantiate webservices
- Specifying webservices with OpenAPI and Spring
- Richardson Maturity Model
- Server Side Patterns
  - Model View Controller
  - Dependency Injection
  - Builder

Internet Applications Design and Implementation

(Lecture 3, Part 1 - Software Architecture - RESTful applications)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Restful interface design (Recap)

- Follows an architectural style (convention)
  - Architectural style that promotes a simpler and more efficient way of providing and connecting web services. Built on top of basic HTTP
- Promotes the decoupling from Data-centric server side applications and client user-centric applications
- Implementations provide (convenient) flavours
  - Web-service style pure JSON/XML Data
  - Complete/partial HTML view responses
  - Javascript code responses (e.g. Rails AJAX responses)
- Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". Architectural Styles and the Design of Network-based Software Architectures (Ph.D.). University of California, Irvine

REST - Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (optional)

Representational State Transfer

- Resource Based
  - vs Action Based
  - Nouns and not verbs to identify data in the system
  - Identified (represented) by URI
  - Aliasing is admissible
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
  - JSON or XML representation of the state of a given resource transferred between client and server at a given verb in a given URL.
  - Well identified interface (the information retrieved at an URL — the type)
- Uniform Interface
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
  - standard HTTP verbs (GET, PUT, POST, DELETE)
  - standard HTTP response (status code, info in the response body)
  - Uniform structure of URIs with a name, identifying the resource
  - References inside responses must be complete.
- Stateless
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
  - Server does not hold session state
  - Messages are self contained
- Cacheable
- Client-Server
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
  - Responses can be tagged as cacheable (in the server)
  - (also) Bookmarkable
- Layered System
- Code on Demand (not talking about it)

Representational State Transfer

- Resource Based
- Representation
- Uniform Interface
- Stateless
- Cacheable
- Layered System
  - Establishes an API between a client and a “database”
- Code on Demand (not talking about it)

EXAMPLES

6. Real REST Examples

Here's a very partial list of service providers that use a REST API. Note that some of them also support a WSDL (Web Services) API, in addition, so you can pick which to use; but in most cases, when both alternatives are available, REST calls are easier to create, the results are easier to parse and use, and it's also less resource-heavy on your system.

So without further ado, some REST services:

- The Google Glass API, known as "Mirror API", is a pure REST API. Here is an excellent video talk about this API. (The actual API discussion starts after 16 minutes or so.)
- Twitter has a **REST API** (in fact, this was their original API and, so far as I can tell, it's still the main API used by Twitter application developers),
- **Flickr**,
- **Amazon.com** offer several REST services, e.g., for their S3 storage solution,
- **Atom** is a RESTful alternative to RSS,
- **Tesla Model S** uses an (undocumented) REST API between the car systems and its Android/iOS apps.

in … http://rest.elkstein.org/2008/02/real-rest-examples.html

Mirror API - Google Glasses

Contacts

For Contacts Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/mirror/v1, unless otherwise noted		
delete	DELETE /contacts/*id*	Deletes a contact.
get	GET /contacts/*id*	Gets a single contact by ID.
insert	POST /contacts	Inserts a new contact.
list	GET /contacts	Retrieves a list of contacts for the authenticated user.
patch	PATCH /contacts/*id*	Updates a contact in place. This method supports patch semantics.
update	PUT /contacts/*id*	Updates a contact in place.

in … https://developers.google.com/glass/v1/reference/

Mirror API - Google Glasses

Timeline

For Timeline Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/mirror/v1, unless otherwise noted		
delete	`DELETE /timeline/id`	Deletes a timeline item.
get	`GET /timeline/id`	Gets a single timeline item by ID.
insert	`POST https://www.googleapis.com/upload/mirror/v1/timeline` and `POST /timeline`	Inserts a new item into the timeline.
list	`GET /timeline`	Retrieves a list of timeline items for the authenticated user.
patch	`PATCH /timeline/id`	Updates a timeline item in place. This method supports patch semantics.
update	`PUT https://www.googleapis.com/upload/mirror/v1/timeline/id` and `PUT /timeline/id`	Updates a timeline item in place.

Mirror API - Google Glasses

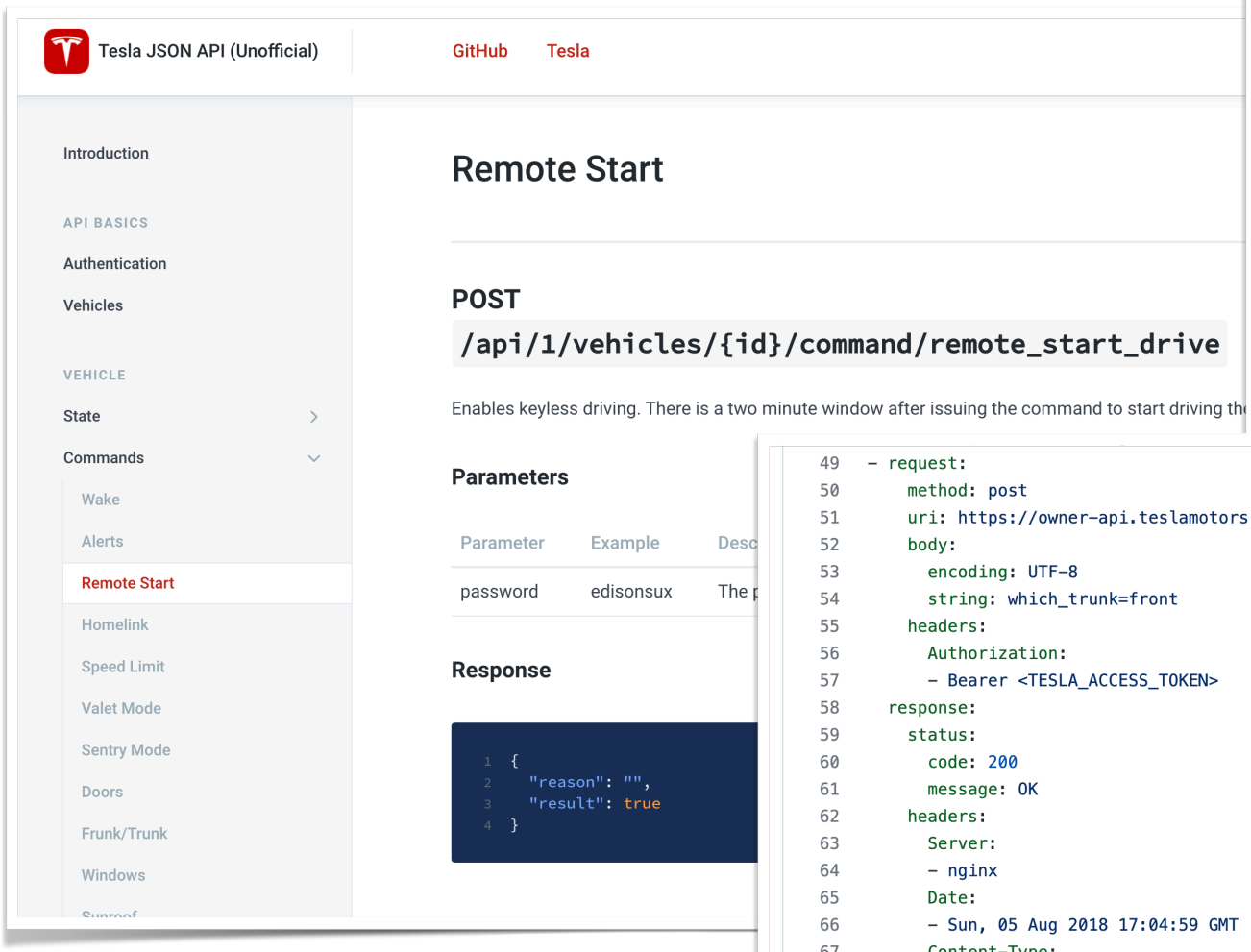
Timeline.attachments

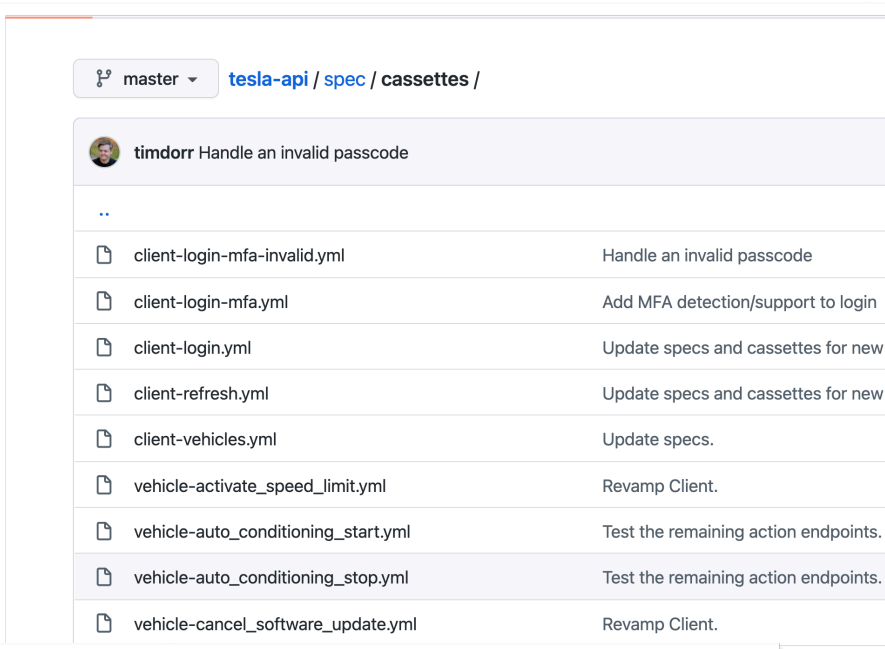
For Timeline.attachments Resource details, see the resource representation page.

Method	HTTP request	Description
URIs relative to https://www.googleapis.com/mirror/v1, unless otherwise noted		
delete	`DELETE /timeline/itemId/attachments/attachmentId`	Deletes an attachment from a timeline item.
get	`GET /timeline/itemId/attachments/attachmentId`	Retrieves an attachment on a timeline item by item ID and attachment ID.
insert	`POST https://www.googleapis.com/upload/mirror/v1/timeline/itemId/attachments`	Adds a new attachment to a timeline item.
list	`GET /timeline/itemId/attachments`	Returns a list of attachments for a timeline item.

in … https://developers.google.com/glass/v1/reference/

Tesla API






```
49  - request:
50      method: post
51      uri: https://owner-api.teslamotors.com/api/1/vehicles/1514029006966957156/command/actuate_trunk
52      body:
53        encoding: UTF-8
54        string: which_trunk=front
55      headers:
56        Authorization:
57        - Bearer <TESLA_ACCESS_TOKEN>
58    response:
59      status:
60        code: 200
61        message: OK
62      headers:
63        Server:
64        - nginx
65        Date:
66        - Sun, 05 Aug 2018 17:04:59 GMT
67        Content-Type:
68        - application/json; charset=utf-8
```

RESTful design

- Resource = object or representation of something
- Collection = a set of resources
- URI = a path identifying **resources** and allowing actions on them
- URL methods represents standardised actions
  - GET = request resources
  - POST = create resources
  - PUT = update or create resources
  - DELETE = deletes resources
- HTTP Response codes = operation results
  - 20x Ok
  - 3xx Redirection (not modified)
  - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
  - 5xx Server Error
- Searching, sorting, filtering and pagination obtained by query string parameters
- Text Based Data format (JSON, or XML)

https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9

Example

- Application to manage contacts of partner companies (e.g. for security clearance in events)
- Resources
  - Companies (name, address, email, list of contacts (employees))
  - Contact/Employee (name, email, job, company)
- Operations (CRUD)
  - List, add, update, and delete resources

Partner companies

- `GET /companies` - List all the companies
- `GET /companies?search=<criteria>` - List all the companies that contain the substring `<criteria>`
- `POST /companies` - Create a company described in the payload. The request body must include all the necessary attributes.
- `GET /companies/{id}` - Shows the company with identifier {id}
- `PUT /companies/{id}` - Updates the company with {id} having values in the payload. The updatable items may vary (name, email, etc.)
- `DELETE /companies/{id}` - Removes the company with {id}

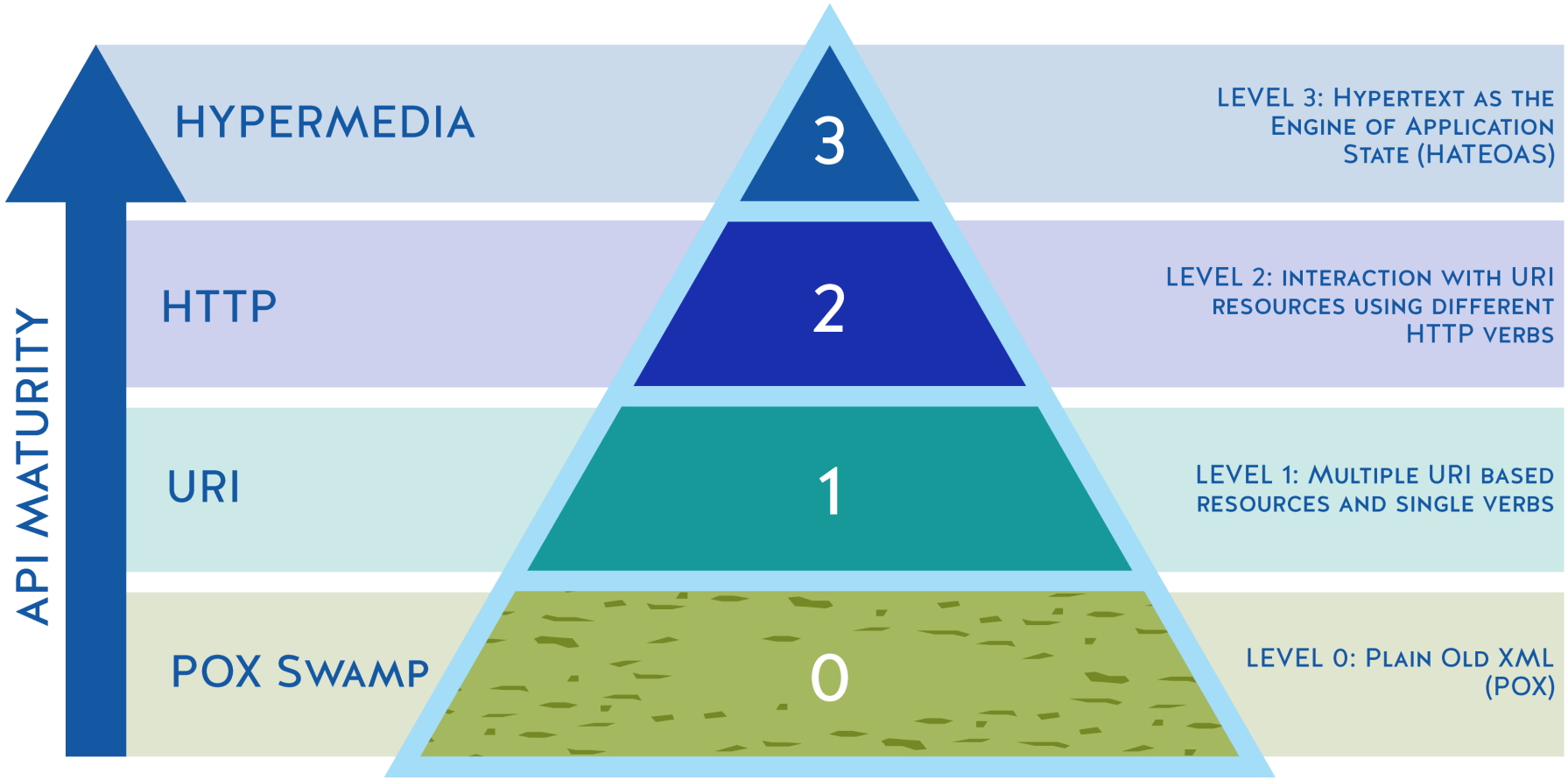
Partner contacts

- `GET /contacts` - List all the contacts
- `GET /contacts?search=<criteria>` - List all the contacts that contain the substring `<criteria>`
- `POST /contacts` - Create a contact described in the payload. The request body must include all the necessary attributes.
- `GET /contacts/{id}` - Shows the contact with identifier {id}
- `PUT /contacts/{id}` - Updates the contact with {id} having values in the payload. The updatable items may vary (name, email, etc.)
- `DELETE /contacts/{id}` - Removes the contact with {id}

Partner contacts of companies

- `GET /companies/{id}/contacts` - List all the contacts of a company
- `GET /companies/{id}/contacts?search=<criteria>` - List all the contacts of a company that contain the substring `<criteria>`
- `POST /companies/{id}/contacts` - Create a contact of company {id} described in the payload. The request body must include all the necessary attributes.
- `GET /companies/{id}/contacts/{cid}` - Shows the contact of company {id} with identifier {cid}
- `PUT /companies/{id}/contacts/{cid}` - Updates the contact with {cid} of company {id} having values in the payload. The updatable items may vary (name, email, etc.)
- `DELETE /companies/{id}/contacts/{cid}` - Removes the contact with {id}

THE RICHARDSON MATURITY MODEL



Example: Contacts in a Spring Controller

```
@RestController
@RequestMapping("/people")
public class PeopleController {

    @Autowired
    PeopleRepository people;

    @Autowired
    PetRepository pets;

    @GetMapping("")
    Iterable<Person> getAllPersons(@RequestParam(required = false) String search)
{
        if( search == null )
            return people.findAll();
        else
            return people.searchByName(search);
    }

    @PostMapping("")
    void addNewPerson(@RequestBody Person p) {
        p.setId(0);
        people.save(p);
    }

    @GetMapping("{id}")
    Optional<Person> getOne(@PathVariable long id) {
        return people.findById(id);
    }
```

JAX-RS: A standard for API declaration

- A lightweight specification method with (Java) annotations
- Implemented by RESTEasy and Jersey
- Similar to Spring annotations
- Official Java Specification
- //jcp.org/en/jsr/detail?id=339

```
@Path("/notifications")
public class NotificationsResource {
    @GET
    @Path("/ping")
    public Response ping() {
        return Response.ok().entity("Service online").build();
    }

    @GET
    @Path("/get/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getNotification(@PathParam("id") int id) {
        return Response.ok()
          .entity(new Notification(id, "john", "test notification"))
          .build();
    }

    @POST
    @Path("/post/")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response postNotification(Notification notification) {
        return Response.status(201).entity(notification).build();
    }
}
```

Spring Example

Kotlin to the rescue

Kotlin

- Language and idioms

Internet Applications Design and Implementation 2020 - 2021

(Lecture 3, Part 2 - Software Architecture - OpenAPI)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Swagger/OpenAPI

- Specification language for REST APIs (Yaml or JSON)
- Provides online (reflective) information on service(s)
  - Paths and operations (GET /companies, POST /employees)
  - Input and output parameters for each operation (samples)
  - Authentication methods
  - Contact information, license, terms of use and other information.
- Design, implementation and validation tools
- Editor, UI, Codegen, Spring Annotations
- Extensions to include more information about contracts

Swagger/OpenAPI - Yaml

- General information about the API

```
swagger: "2.0"
info:
  description: "This is a sample directory of partner companies."
  version: "1.0.0"
  title: "Partner Companies"
host: "partners.swagger.io"
basePath: "/"
tags:
- name: "companies"
  description: "Everything about your partner companies"
  externalDocs:
    description: "Find out more"
    url: "http://swagger.io"
- name: "contacts"
  description: "Know all about your partners employees"
schemes:
- "https"
- “http"
paths:
  ...
definitions:
  ...
externalDocs:
  description: "Find out more about Swagger"
  url: "http://swagger.io"
```

Swagger/OpenAPI - Yaml

- Specific information about each path/operation available

```
paths:
  /companies:
    get:
      tags:
      - "companies"
      summary: "Get the list of all companies"
      description: ""
      operationId: "getCompanies"
      produces:
      - "application/json"
      parameters:
      - in: "query"
        name: "search"
        description: "Filter companies by name, description, or address"
        type: "string"
        required: false
      responses:
        200:
          description: "successful operation"
          schema:
            type: "array"
            items:
```

Swagger/OpenAPI - Yaml

- Specific information about each path/operation available

```
post:
  tags:
  - "companies"
  summary: "Add a new partner company to the collection"
  description: ""
  operationId: "addCompany"
  consumes:
  - "application/json"
  parameters:
  - in: "body"
    name: "company"
    description: "Company object that needs to be added to the collection"
    required: true
    schema:
      $ref: "#/definitions/Company"
  responses:
    200:
      description: "Company added"
    405:
      description: "Invalid input"
```

Swagger/OpenAPI - Yaml

- Specific information about each path/operation available

```
/companies/{id}:
  get:
    tags:
    - "companies"
    summary: "Gets an existing company with {id} as identifier"
    description: "Gets an existing company with {id} as identifier"
    operationId: "getCompany"
    parameters:
    - in: "path"
      name: "id"
      description: "The identifier of the company to be updated"
      required: true
      type: "integer"
      format: "int64"
    responses:
      200:
        description: "The company data"
        schema:
          $ref: "#/definitions/Company"
```

Swagger/OpenAPI - Yaml

- Specific information about each path/operation available

```
put:
  tags:
  - "companies"
  summary: "Update an existing company with {id} as identifier"
  description: "Update an existing company with {id} as identifier"
  operationId: "updateCompany"
  consumes:
  - "application/json"
  parameters:
  - in: "path"
    name: "id"
    description: "The identifier of the company to be updated"
    required: true
    type: "integer"
    format: "int64"
  - in: "body"
    name: "company"
    description: "Company object that needs to be updated in the collection"
    required: true
    schema:
      $ref: "#/definitions/Company"
  responses:
    200:
      description: "Updated company"
    400:
      description: "Invalid ID supplied"
    404:
      description: "Company not found"
    405:
      description: "Validation exception"
```

Swagger/OpenAPI - Yaml

- Specific information about datatypes

```
definitions:

  Company:
    type: "object"
    required:
    - "name"
    - "address"
    - "email"
    properties:
      id:
        type: "integer"
        format: "int64"
      name:
        type: "string"
        example: "ecma"
      address:
        type: "string"
        example: "Long Street"
      email:
        type: "string"
        example: "info@acme.com"
      employees:
        type: "array"
        items:
          $ref: "#/definitions/Employee"
```

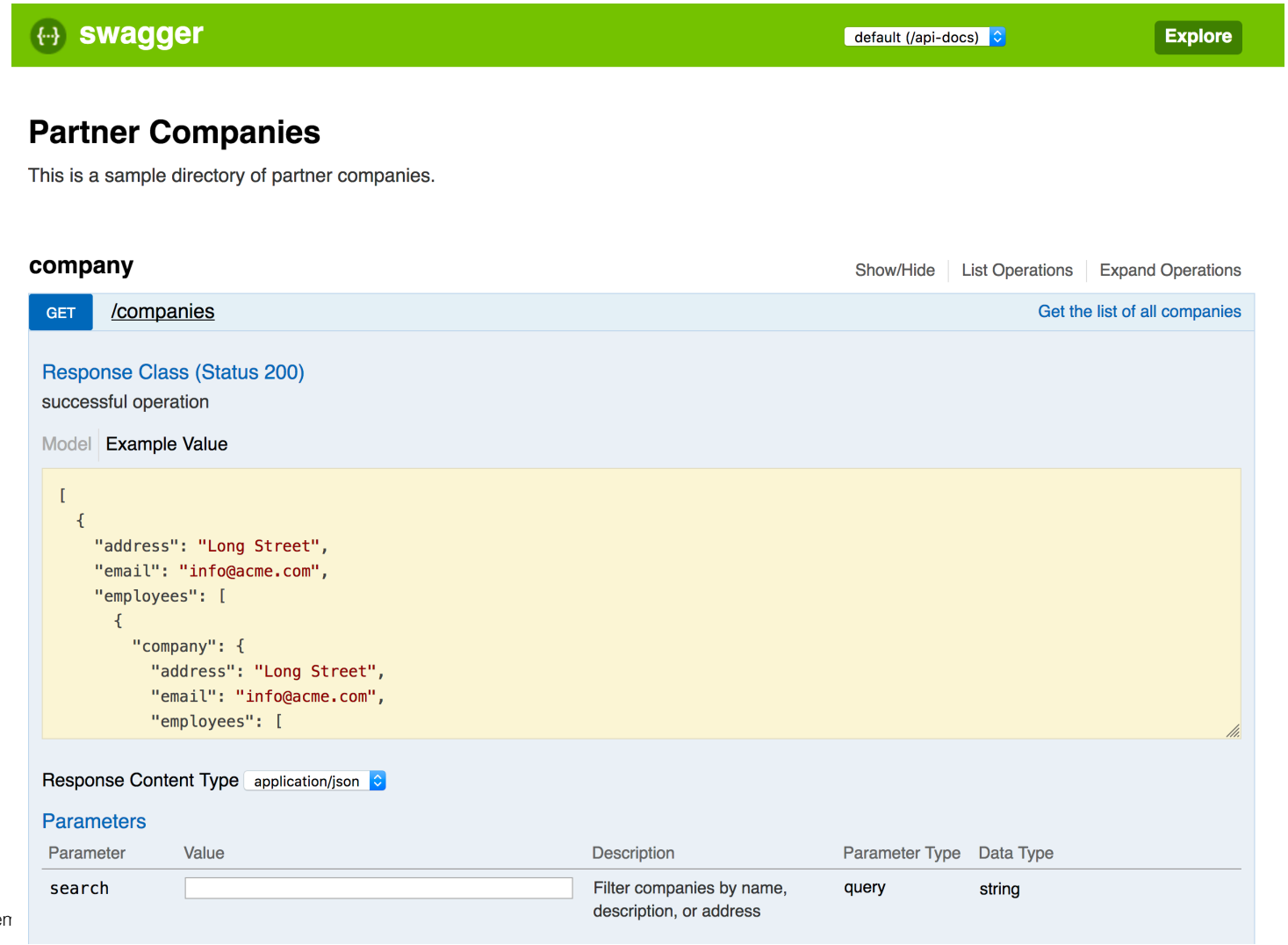
Generated API code (in Java)

```
@Api(value = "companies", description = "the companies API")
public interface CompaniesApi {

    @ApiOperation(value = "Add a new partner company to the collection", nickname = "addCompany", notes = "", tags={ "company", })
    @ApiResponses(value = { @ApiResponse(code = 405, message = "Invalid input") })
    @RequestMapping(value = "/companies",
        produces = { "application/json" },
        consumes = { "application/json" },
        method = RequestMethod.POST)
    ResponseEntity<Void> addCompany(@ApiParam(value = "Company object that needs to be added to the collection" ,required=true )  @Valid @RequestBody Company company);

    @ApiOperation(value = "Get the list of all companies", nickname = "getCompanies", notes = "", response = Company.class, responseContainer = "List", tags={ "company", })
    @ApiResponses(value = { @ApiResponse(code = 200, message = "successful operation", response = Company.class, responseContainer = "List") })
    @RequestMapping(value = "/companies",
        produces = { "application/json" },
        consumes = { "application/json" },
        method = RequestMethod.GET)
    ResponseEntity<List<Company>> getCompanies(@ApiParam(value = "Filter companies by name, description, or address") @Valid @RequestParam(value = "search", required = false) String
search);

    @ApiOperation(value = "Update an existing company", nickname = "updateCompany", notes = "", tags={ "company", })
    @ApiResponses(value = { @ApiResponse(code = 400, message = "Invalid ID supplied"),
                            @ApiResponse(code = 404, message = "Company not found"),
                            @ApiResponse(code = 405, message = "Validation exception") })
    @RequestMapping(value = "/companies",
        produces = { "application/json" },
        consumes = { "application/json" },
        method = RequestMethod.PUT)
    ResponseEntity<Void> updateCompany(@ApiParam(value = "Company object that needs to be updated in the collection" ,required=true )  @Valid @RequestBody Company company);
}
```

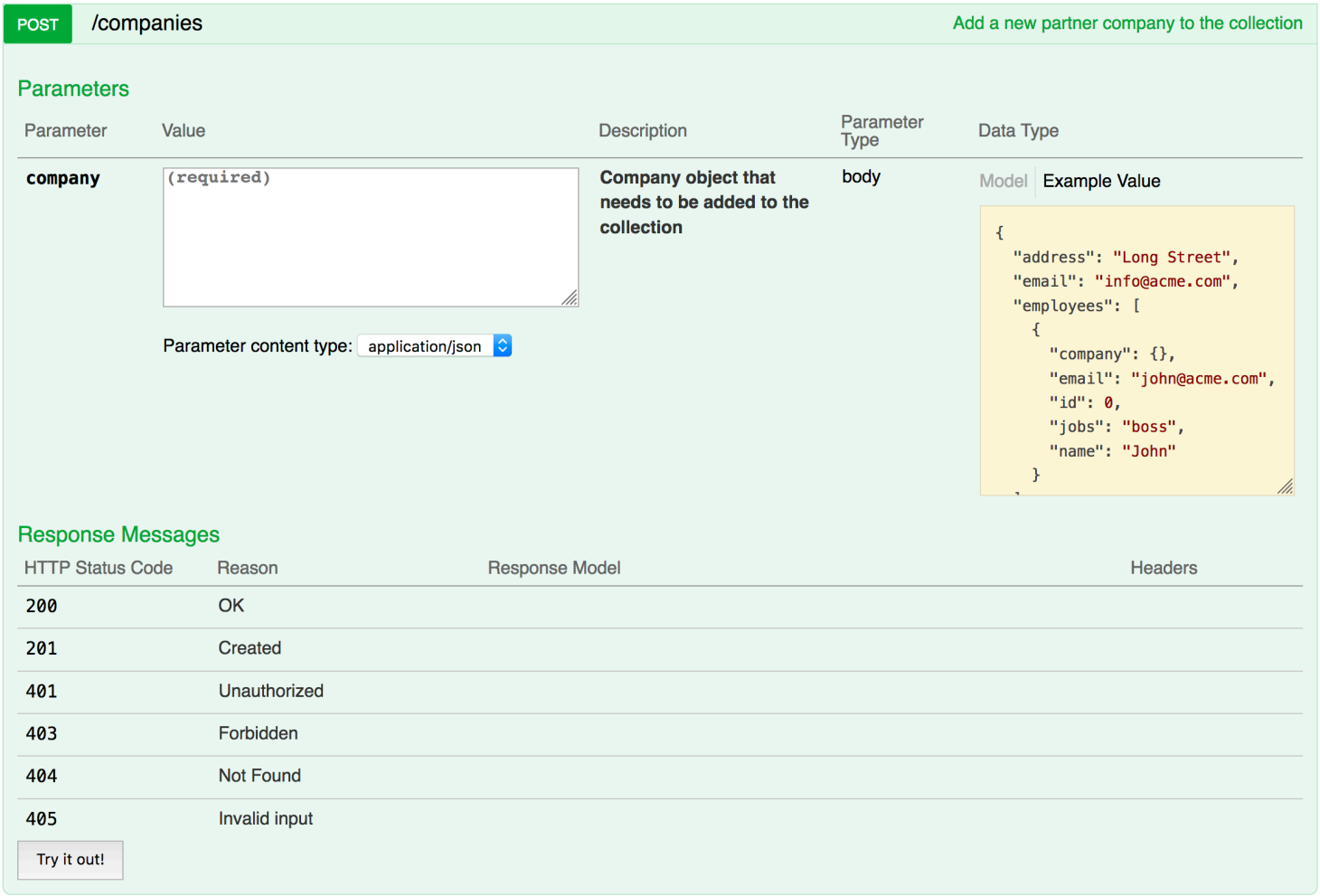
Generated Model Code

```
public class Company   {
  @JsonProperty("id")
  private Long id = null;

  @JsonProperty("name")
  private String name = null;

  @JsonProperty("address")
  private String address = null;

  @JsonProperty("email")
  private String email = null;

  @JsonProperty("employees")
  @Valid
  private List<Employee> employees = null;
…
```

Online information about API



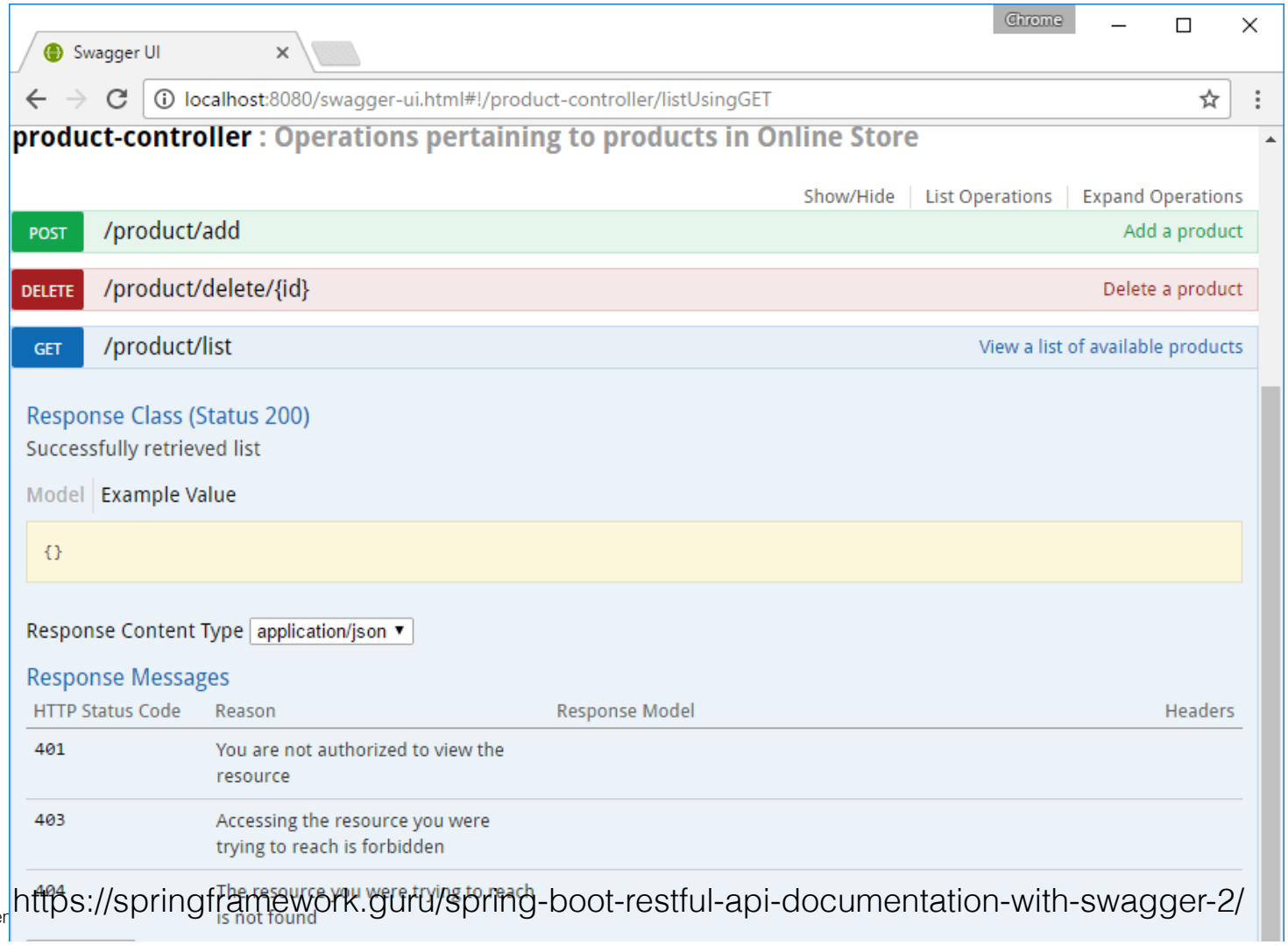
Online information about API



Machine readable specification

```
@RestController
@RequestMapping("/product")
@Api(value="onlinestore", description="Operations pertaining to products in Online Store")
public class ProductController {

    private ProductService productService;

    @Autowired
    public void setProductService(ProductService productService) {
        this.productService = productService;
    }

    @ApiOperation(value = "View a list of available products",response = Iterable.class)
    @ApiResponses(value = {
            @ApiResponse(code = 200, message = "Successfully retrieved list"),
            @ApiResponse(code = 401, message = "You are not authorized to view the resource"),
            @ApiResponse(code = 403, message = "Accessing the resource you were trying to reach is forbidden"),
            @ApiResponse(code = 404, message = "The resource you were trying to reach is not found")
    }
    )
    @RequestMapping(value = "/list", method= RequestMethod.GET, produces = "application/json")
    public Iterable<Product> list(Model model){
        Iterable<Product> productList = productService.listAllProducts();
        return productList;
    }
    @ApiOperation(value = "Search a product with an ID",response = Product.class)
    @RequestMapping(value = "/show/{id}", method= RequestMethod.GET, produces = "application/json")
    public Product showProduct(@PathVariable Integer id, Model model){
       Product product = productService.getProductById(id);
        return product;
    }
```

Machine readable specification



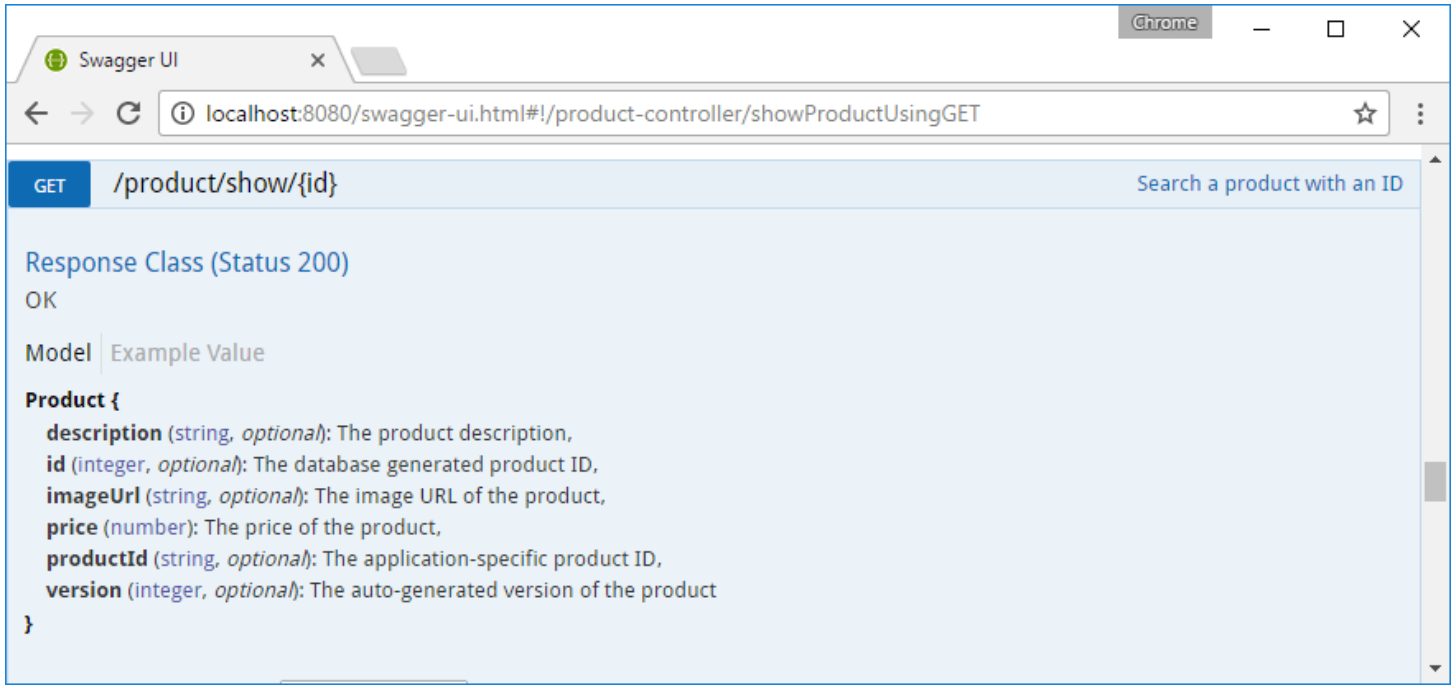
https://springframework.guru/spring-boot-restful-api-documentation-with-swagger-2/

Machine readable specification

```
@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @ApiModelProperty(notes = "The database generated product ID")
    private Integer id;
    @Version
    @ApiModelProperty(notes = "The auto-generated version of the product")
    private Integer version;
    @ApiModelProperty(notes = "The application-specific product ID")
    private String productId;
    @ApiModelProperty(notes = "The product description")
    private String description;
    @ApiModelProperty(notes = "The image URL of the product")
    private String imageUrl;
    @ApiModelProperty(notes = "The price of the product", required = true)
    private BigDecimal price;
…
```

https://springframework.guru/spring-boot-restful-api-documentation-with-swagger-2/

Machine readable specification



https://springframework.guru/spring-boot-restful-api-documentation-with-swagger-2/

Internet Applications Design and Implementation

2020 - 2021

(Lab class 2)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Eduardo Geraldo (e.geraldo@campus.fct.unl.pt)

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Lab Class 2&3

Swagger/Rest in Spring and Kotlin

Internet Applications Design and Implementation

(Lecture 3 - Server side programming, Data Sources)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Internet Applications Design and Implementation 2020 - 2021

(Lecture 3 - Part 3 - RESTful interfaces in practice)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

RESTful design

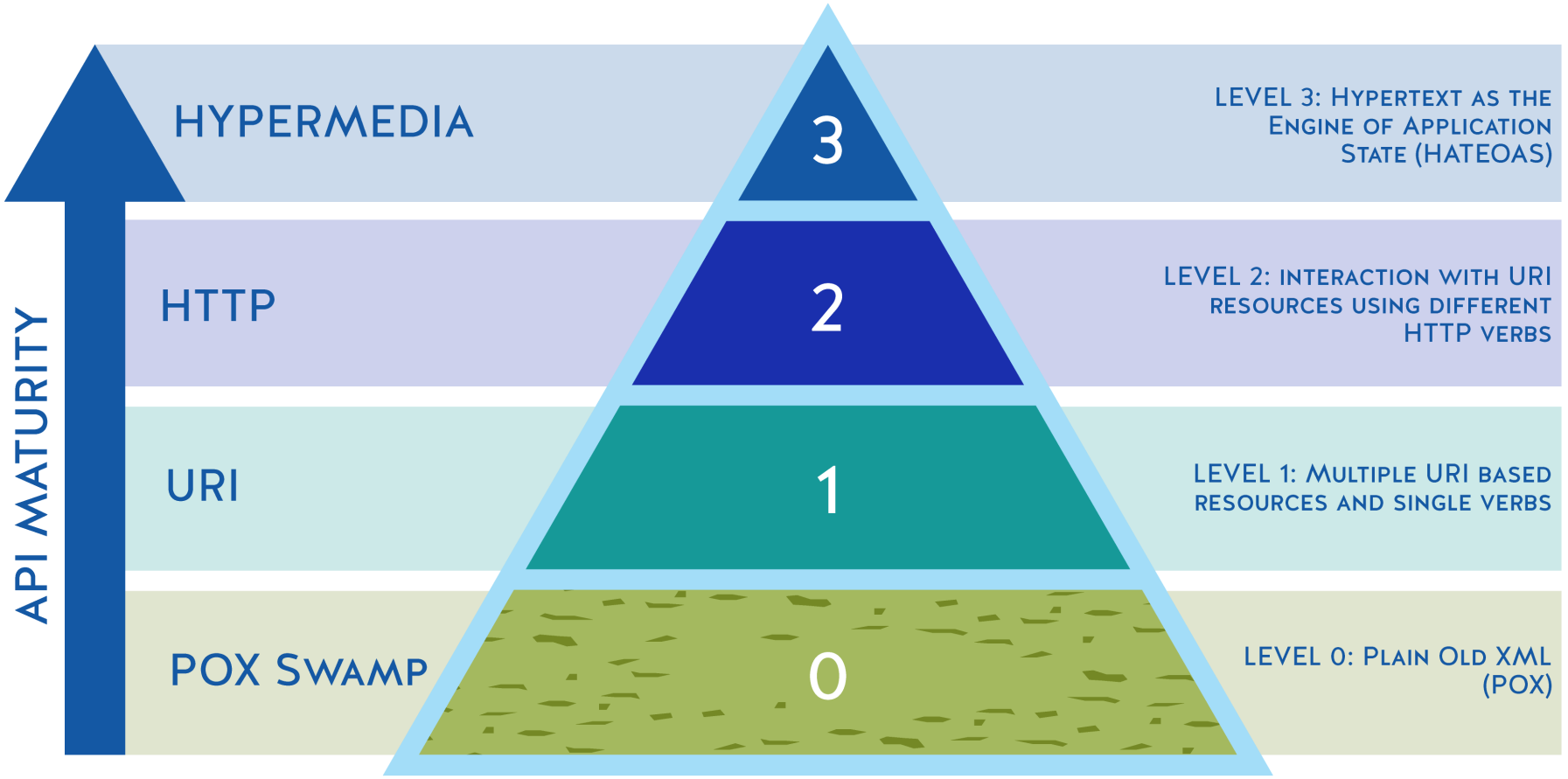
RECAP

- Resource = object or representation of something
- Collection = a set of resources
- URI = a path identifying **resources** and allowing actions on them
- URL methods represents standardised actions
  - GET = request resources
  - POST = create resources
  - PUT = update or create resources
  - DELETE = deletes resources
- HTTP Response codes = operation results
  - 20x Ok
  - 3xx Redirection (not modified)
  - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found
  - 5xx Server Error
- Searching, sorting, filtering and pagination obtained by query string parameters
- Text Based Data format (JSON, or XML)

https://hackernoon.com/restful-api-designing-guidelines-the-best-practices-60e1d954e7c9

RECAP

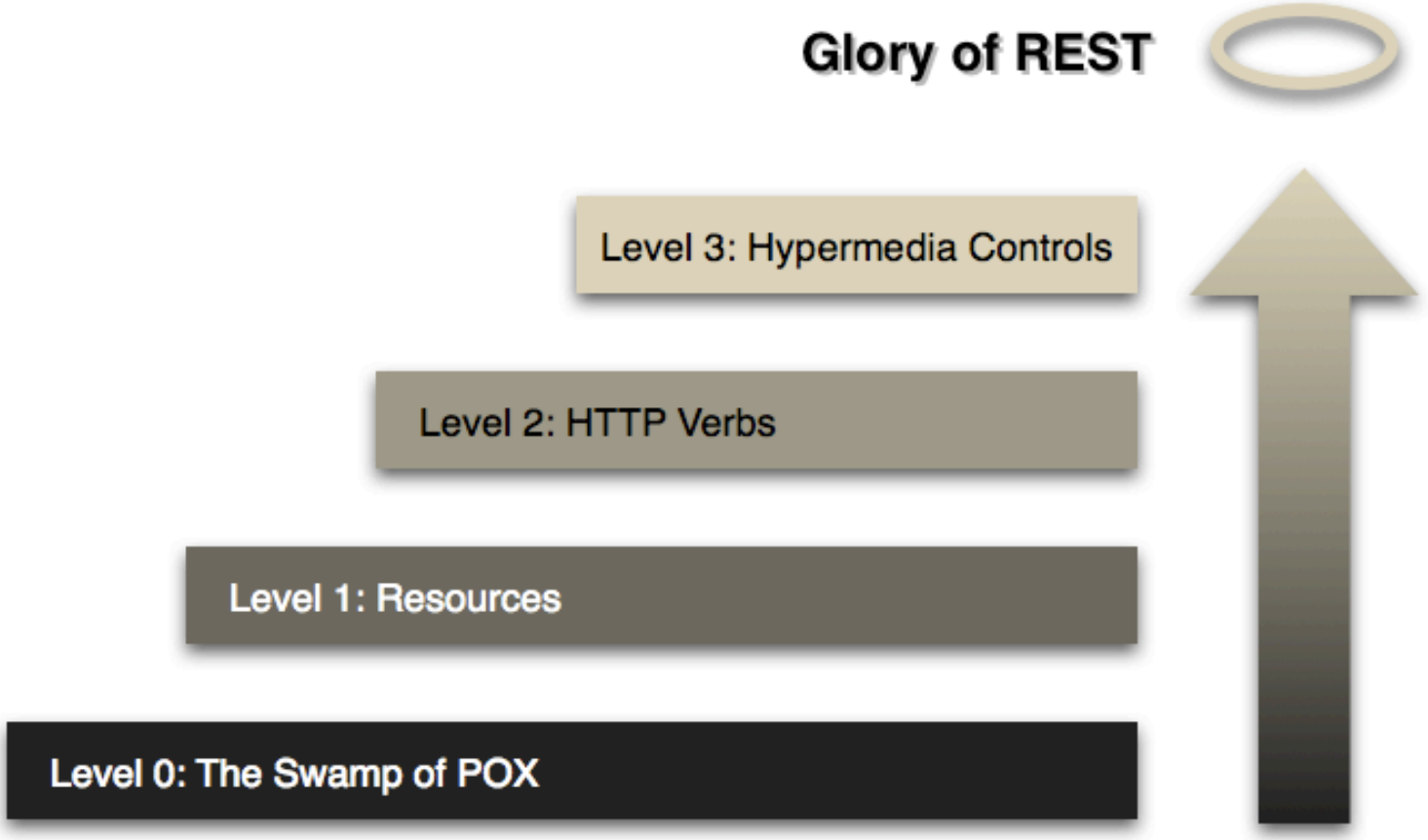
THE RICHARDSON MATURITY MODEL



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

http://restcookbook.com/Miscellaneous/richardsonmaturitymodel/

Richardson Maturity Model



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

http://restcookbook.com/Miscellaneous/richardsonmaturitymodel/

The Richardson Maturity Model - Level 0

- POX Swamp
  - To send an XML/JSON that contains everything: operation, arguments, options

```
POST /appointmentService HTTP/1.1
[various other headers]

<openSlotRequest date = "2010-01-04" doctor = "mjones"/>
```

```
<openSlotList>
  <slot start = "1400" end = "1450">
    <doctor id = "mjones"/>
  </slot>
  <slot start = "1600" end = "1650">
    <doctor id = "mjones"/>
  </slot>
</openSlotList>
```

The Richardson Maturity Model - Level 0

- POX Swamp
  - To send an XML/JSON that contains everything: operation, arguments, options

```
POST /appointmentService HTTP/1.1
[various other headers]

<appointmentRequest>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointmentRequest>
```

```
HTTP/1.1 200 OK
[various headers]

<appointmentRequestFailure>
  <slot doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
  <reason>Slot not available</reason>
</appointmentRequestFailure>
```

The Richardson Maturity Model - Level 1

- Multiple URI Based Resources and Single verbs

```
POST /doctors/mjones HTTP/1.1
[various other headers]

<openSlotRequest date = "2010-01-04"/>
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>
```

```
HTTP/1.1 200 OK
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

The Richardson Maturity Model - Level 1

- Multiple URI Based Resources and Single verbs

```
@Controller @RequestMapping(value = "/pets")
class PetController @Autowired constructor (val db: MongoDB) {
    @RequestMapping(value = "/add", method = arrayOf(RequestMethod.GET))
    public fun add(@RequestParam("ownerId") ownerIdParam: String, model: Model): String {
        db.withSession {
            val owner = Owners.find { id.equal(Id(ownerIdParam)) }.single()
            model.addAttribute("owner", owner)
            val petTypes = PetTypes.find().toList()
            model.addAttribute("petTypes", petTypes)
        }
        return "pets/add"
    }
```

The Richardson Maturity Model - Level 2

- Interaction with URI resources using different HTTP verbs

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

The Richardson Maturity Model - Level 2

- Interaction with URI resources using different HTTP verbs

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>
```

```
HTTP/1.1 201 Created
Location: slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

The Richardson Maturity Model - Level 2

- Interaction with URI resources using different HTTP verbs

```
POST /slots/1234 HTTP/1.1
[various other headers]

<appointmentRequest>
  <patient id = "jsmith"/>
</appointmentRequest>
```

```
HTTP/1.1 201 Created
Location: slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450"/>
  <patient id = "jsmith"/>
</appointment>
```

```
HTTP/1.1 409 Conflict
[various headers]

<openSlotList>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650"/>
</openSlotList>
```

The Richardson Maturity Model - Level 3

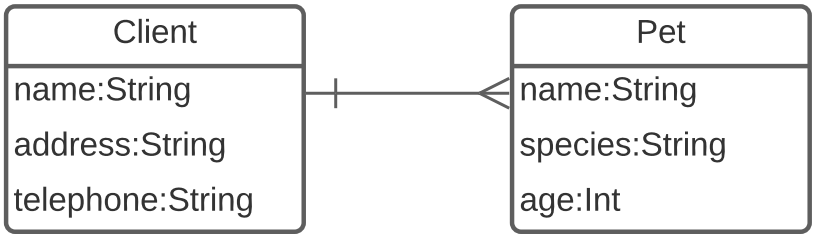
- Hypermedia Controls - HATEOAS
  - Resources are interconnected by links in the response, one entry point

```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```

```
HTTP/1.1 200 OK
[various headers]

<openSlotList>
  <slot id = "1234" doctor = "mjones" start = "1400" end = "1450">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/1234"/>
  </slot>
  <slot id = "5678" doctor = "mjones" start = "1600" end = "1650">
     <link rel = "/linkrels/slot/book"
           uri = "/slots/5678"/>
  </slot>
</openSlotList>
```

REST = Resource state transformation

- The resources that are provided by the API do not have to map the structure of the internal system state.
- Provided resources may have a nested structure that results from a relational structure of several database tables.




```
[{“name”:”joe”,
  “address”: “London, UK”,
  “telephone”:”555000222”,
  “pets”:[
          {“name”:“Max”,
           “species”:”Canis lupus familiaris”,
           ”age”:3},
          {“name”:“Max”,
           “species”:”Canis lupus familiaris”,
           ”age”:3}
        ]},
 {“name”,”mary”,
  “address”:…. }
```