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
- Microservices Patterns
  - API Gateway
  - Circuit-Breaker

Internet Applications Design and Implementation

(Lecture 3, Part 1 - 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

Partner Companies

This is a sample directory of partner companies.

company

Show/Hide | List Operations | Expand Operations

GET /companies — Get the list of all companies

Response Class (Status 200)

successful operation

Model | Example Value

```
[
  {
    "address": "Long Street",
    "email": "info@acme.com",
    "employees": [
      {
        "company": {
          "address": "Long Street",
          "email": "info@acme.com",
          "employees": [
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
search		Filter companies by name, description, or address	query	string

Online information about API

POST /companies — Add a new partner company to the collection

Parameters

Parameter	Value	Description	Parameter Type	Data Type
company	(required)	**Company object that needs to be added to the collection**	body	Model | Example Value

Parameter content type: application/json

```
{
  "address": "Long Street",
  "email": "info@acme.com",
  "employees": [
    {
      "company": {},
      "email": "john@acme.com",
      "id": 0,
      "jobs": "boss",
      "name": "John"
    }
```

Response Messages

HTTP Status Code	Reason	Response Model	Headers
200	OK		
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		
405	Invalid input		

Try it out!

Machine readable specification

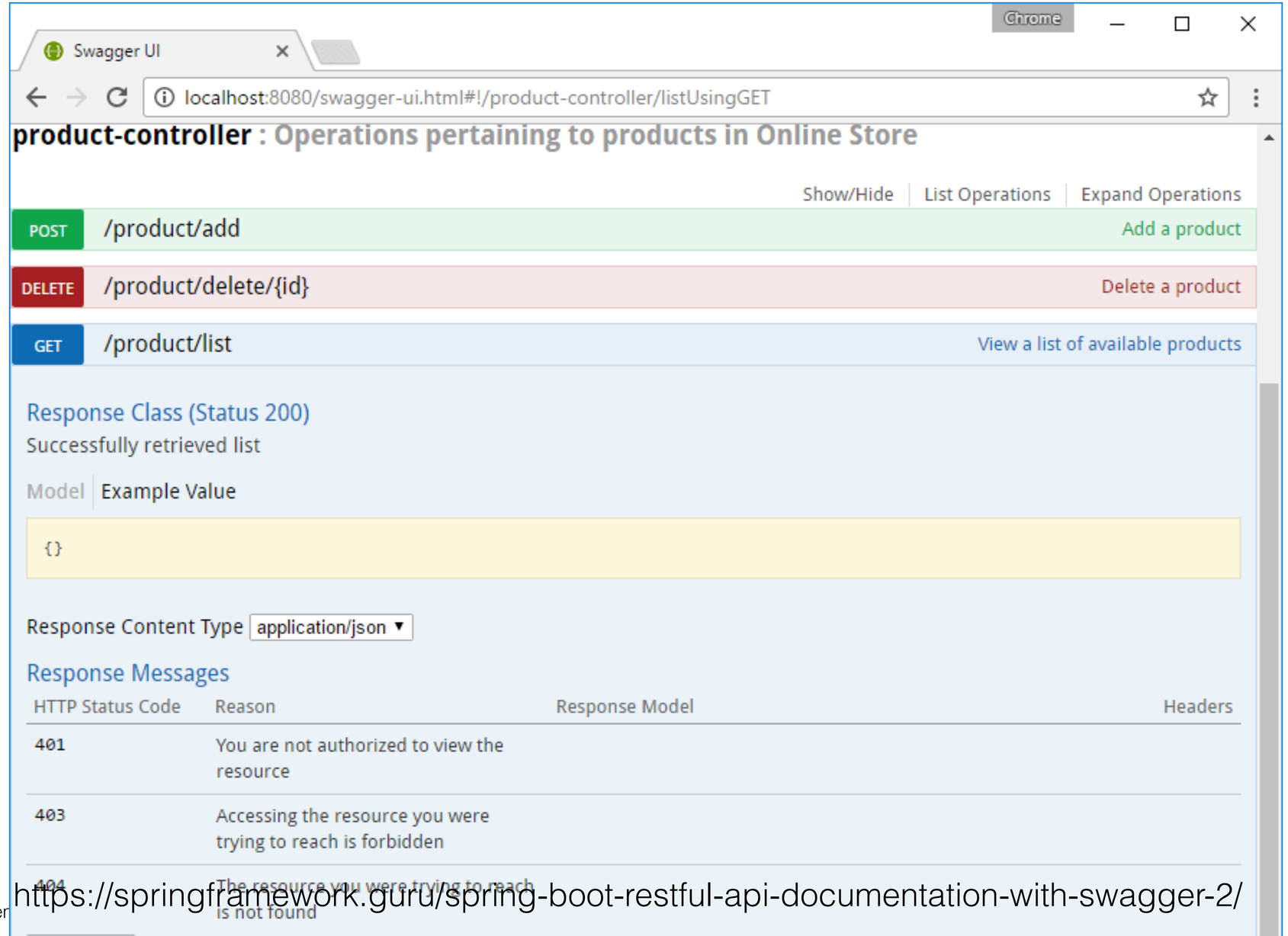
```
@RestController
@RequestMapping("/product")
@Api(value="onlinestore", description="Operations pertaining to products in Online Store")
public class ProductController {

    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



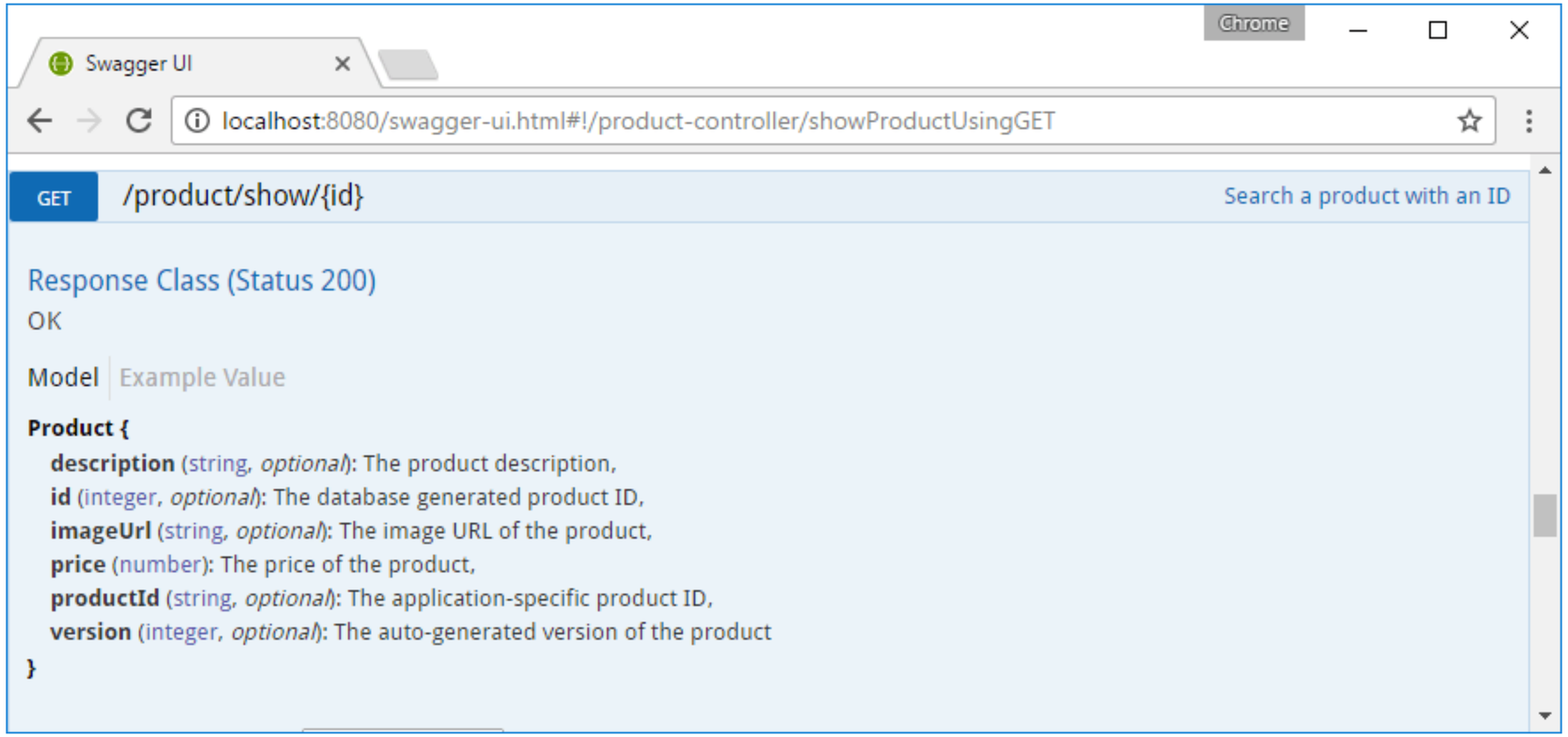
https://springframework.guru/spring-boot-restful-api-documentation-with-swagger-2/

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

(Lecture 3 - Part 2 - 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

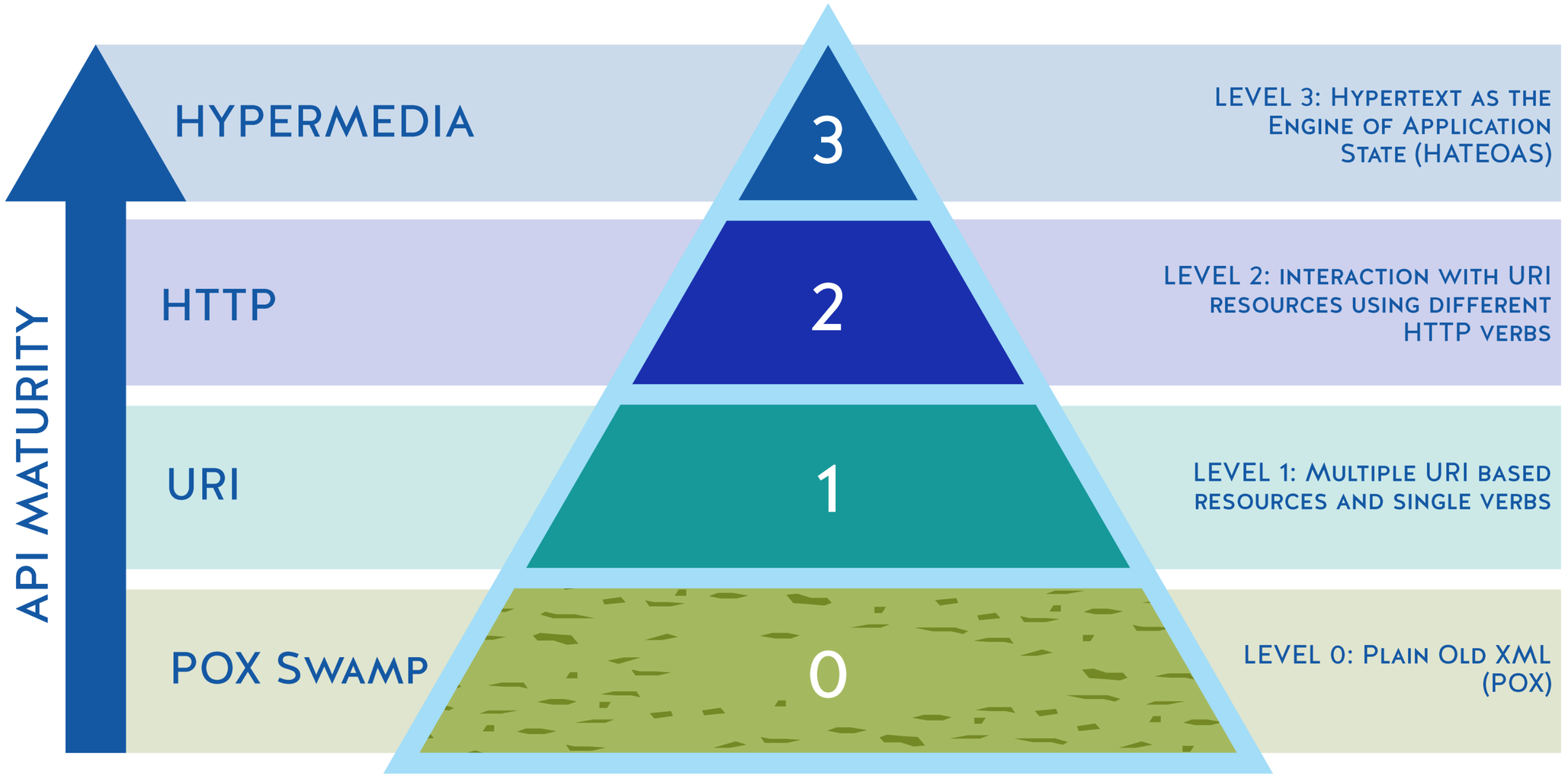
RECAP

- 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

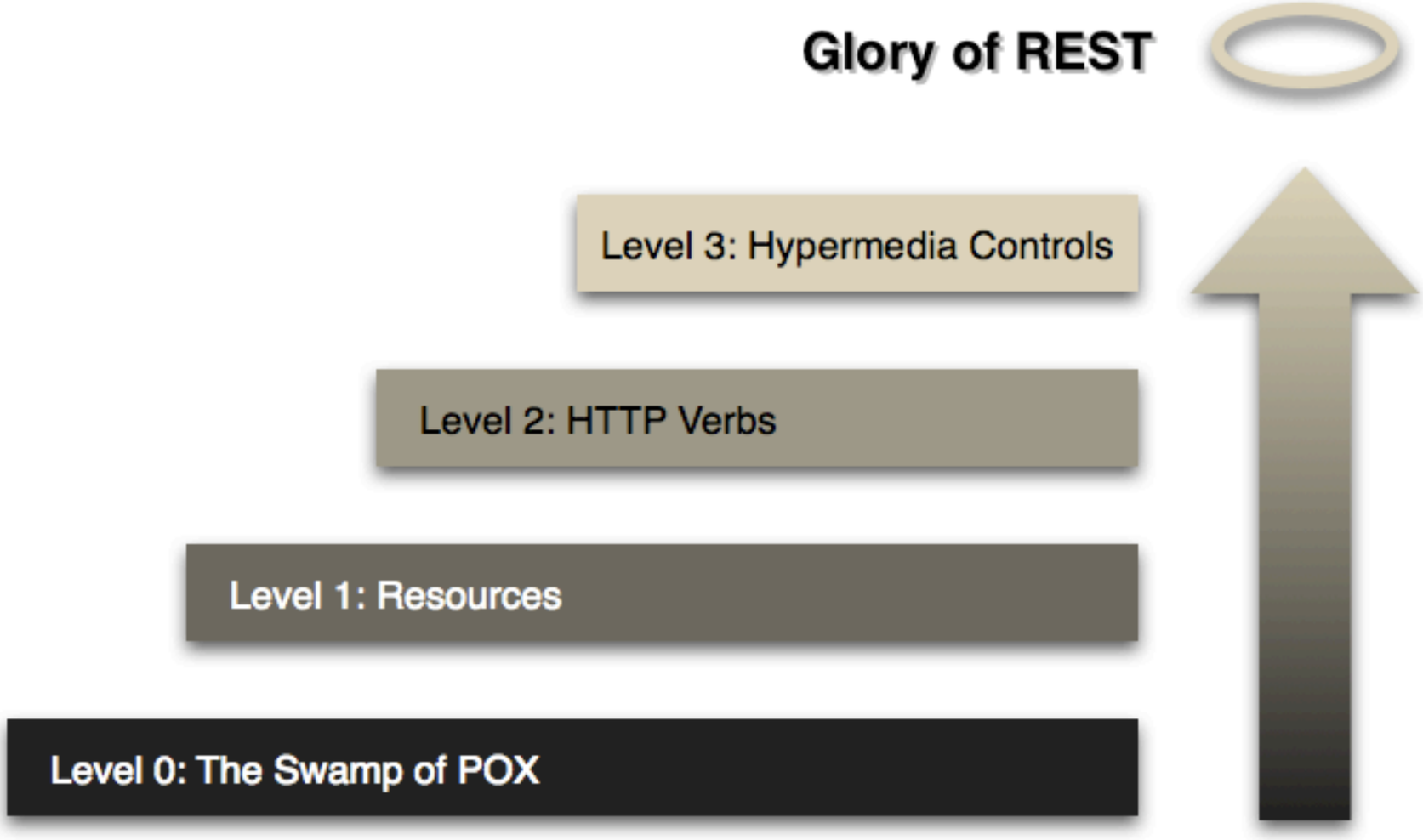
THE RICHARDSON MATURITY MODEL



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

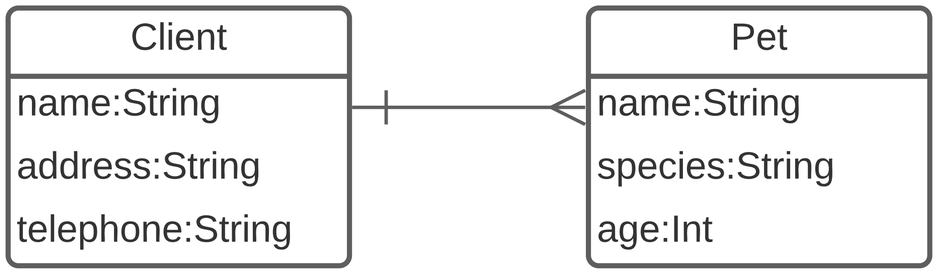
```
GET /doctors/mjones/slots?date=20100104&status=open HTTP/1.1
Host: royalhope.nhs.uk
```

```
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”:…. }
```

Internet Applications Design and Implementation 2020 - 2021

(Lecture 3 - Part 3 - Server-side MVC Architecture)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

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

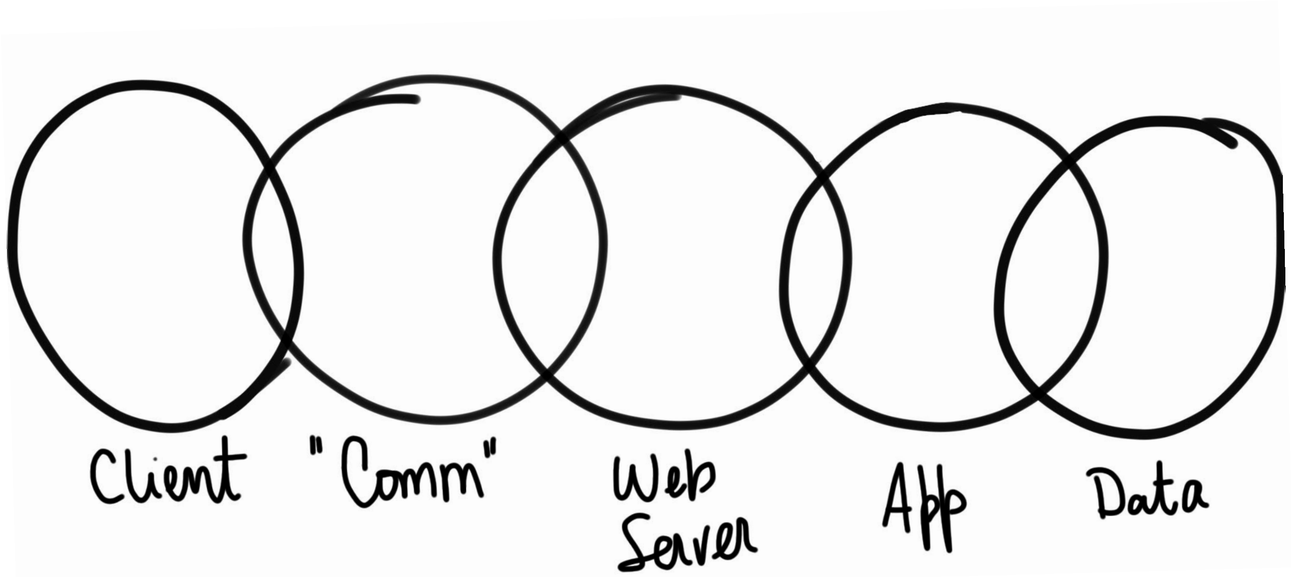
(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Web architectures, patterns and styles

RECAP

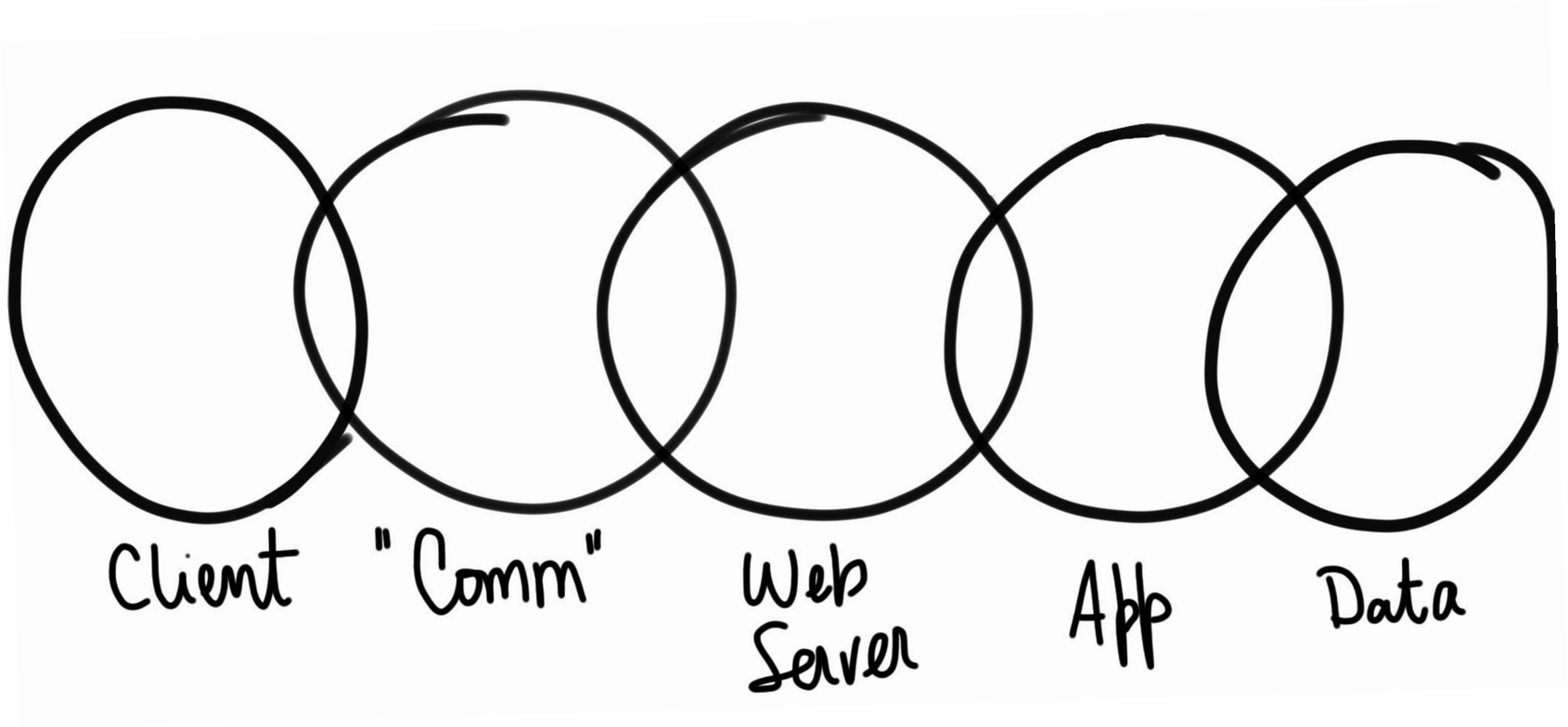
- Web applications usually follow a MVC architectural pattern.
  - Model layer - isolate the representation of persistent data and its operations, validations and conditions
  - Controller - contains the core application logic implementing the application interface (e.g. ad-hoc URL mapping, REST convention)
  - View - defines the way in which responses are formed (e.g. HTML, JSON)



Summary - Web Frameworks

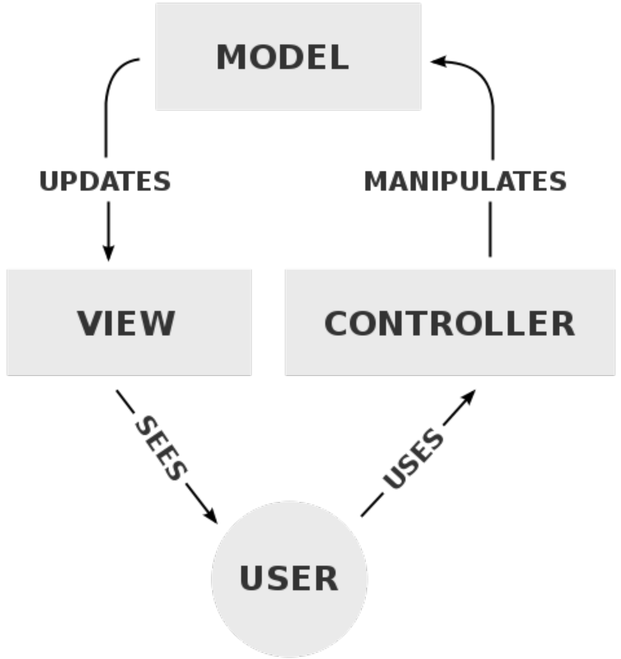
RECAP

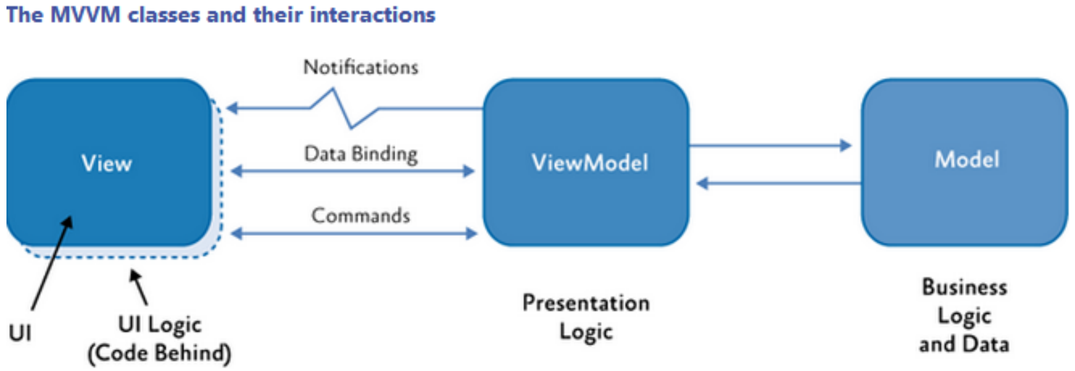
- Web Frameworks are “languages” that carry libraries and abstractions that get compiled to run on the “web virtual machine”.



The classic MVC design pattern

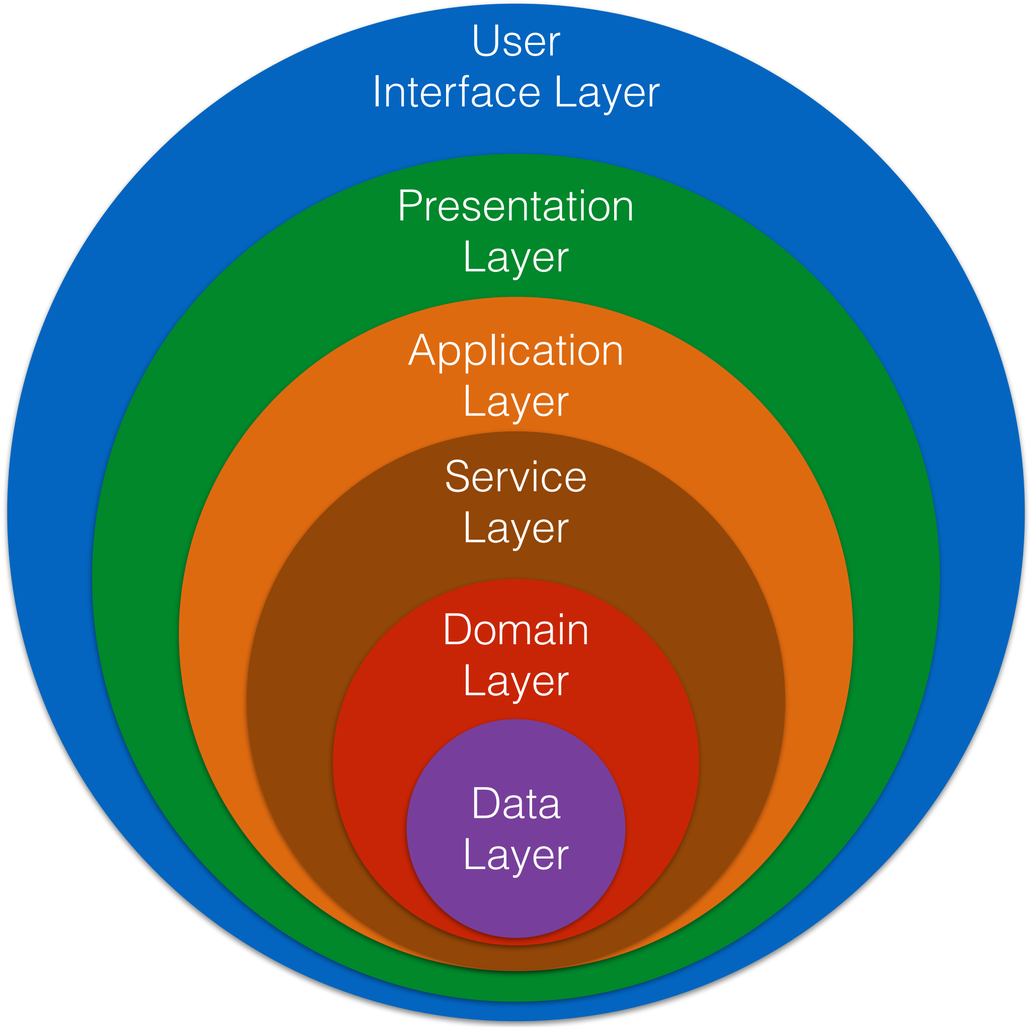
- The Model-View-Controller (Reenskaug'79, JOT'88)
  - designed to develop GUI
  - popular in web applications' context
- Variants of the MVC Architecture (Separation of Concerns)
  - MVP, PM (Fowler), MVVM (Microsoft)





https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/

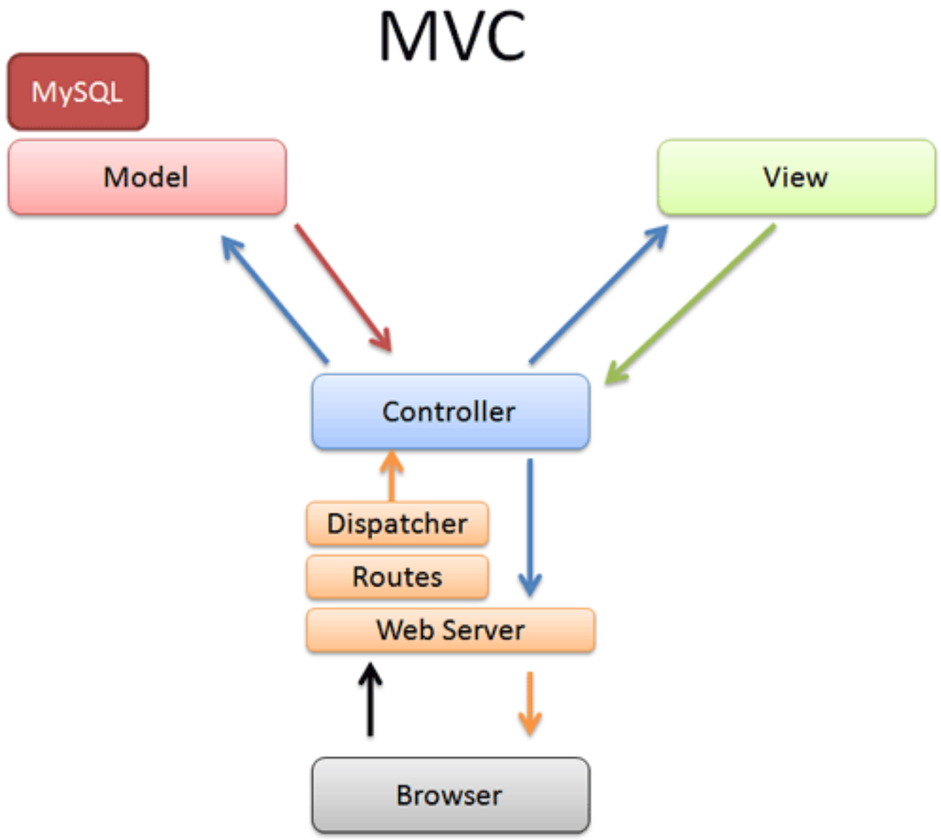
Internet Applications are Data-Centric



https://dzone.com/articles/layered-architecture-is-good

Frameworks and MVC Architecture

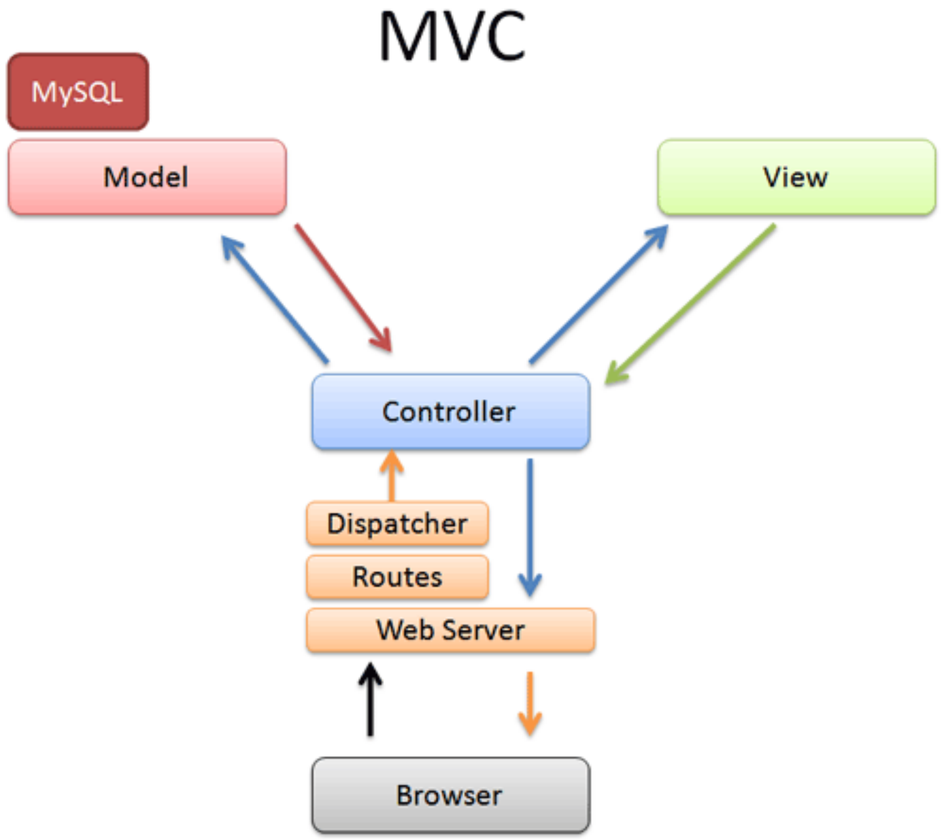
- Frameworks help to implement and maintain architectures.
- Rails (2005):
  - conventions on folder, file, and class names
  - A flexible OO prog language (Ruby) supports data sharing between model, controller, and view objects.



https://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

Frameworks and MVC Architecture

- Frameworks help to implement and maintain architectures.
- Django (2005):
  - views are controllers
  - templates are views
  - models are models



https://betterexplained.com/articles/intermediate-rails-understanding-models-views-and-controllers/

Frameworks and MVC Architecture

- Java Spring is a component-based programming framework (based on configuration).
- It does the “plumbing," and lets components implement the “logic” of applications.
- How spring implements the MVC pattern
  - Dependency Injection (inversion of control)
  - Aspect-Oriented Programming including Spring's declarative transaction management
  - Spring MVC web application and RESTful web service framework
  - Foundational support for JDBC, JPA, JMS
  - …

https://spring.io/guides/

Inversion of Control

- Design pattern where user-defined code fragment receives the flow of control from a generic framework.
  - Context: object-oriented programming
  - Found in: Frameworks, Event handlers, Callbacks
- Dependency Injection
  - An instance of inversion of control to build object networks
  - Centralised broker that maps types to implementations
  - Java Spring: Pool of beans/components, auto-wiring of object networks

Without inversion of Control

- Explicit initialisation of references

```
@RestController
@RequestMapping("/")
class EmpController(val employees:EmployeeService) {

   @GetMapping("/api/departments/{id}/employees")
    fun employeesOfDepartment(
            @PathVariable id:String,
            @RequestParam search:String?
    )
    = listOf(
                Employee("John Oliver",40,"New York"),
                Employee("John Gleese", 60, "London")
        )

   @GetMapping( "/api/projects/{id}/team")
    fun teamMembersOfProject(
            @PathVariable id:String
    )
    = employees.teamMembersOfProject(id)


}
```

```
@Service
class EmployeeService(val employees:EmployeeRepository) {
    fun teamMembersOfProject(id:String) = employees.findAll()
}

interface EmployeeRepository : CrudRepository<Employee, Long>
```

```
fun someMethod() {
    EmpController(
            EmployeeService(
                    EmployeeRepositoryImp(
                            DBConnection("...")
                    )
            )
    )
}
```

Without inversion of Control

- Explicit initialisation of references

```
package pt.unl.fct.demo.controllers;

import org.springframework.web.bind.annotation.*;
import pt.unl.fct.demo.model.Company;
import pt.unl.fct.demo.services.CompaniesService;

@RestController
@RequestMapping(value="/companies")
public class CompaniesController {

    CompaniesService companies;

    public CompaniesController(CompaniesService companies) {
        this.companies = companies;
    }

    @GetMapping("")
    Iterable<Company> getAllCompanies(@RequestParam(required=false) String search) {
        // Do some extra checking on the request, and then...
        return companies.getAllCompanies(search);
    }

    @PostMapping("")
    void addNewCompany(@RequestBody Company company) {
        // Do some extra checking on the request, and then...
        companies.addCompany(company);
    }
}
```

Spring uses annotations to indicate the kind of class, and where to plug it in

Without inversion of Control

- Explicit initialisation of references

```
package pt.unl.fct.demo.controllers;

import org.springframework.web.bind.annotation.*;
import pt.unl.fct.demo.model.Company;
import pt.unl.fct.demo.services.CompaniesService;

@RestController
@RequestMapping(value="/companies")
public class CompaniesController {

    CompaniesService companies;

    public CompaniesController(CompaniesService companies) {
        this.companies = companies;
    }

    @GetMapping("")
    Iterable<Company> getAllCompanies(@RequestParam(required=false) String search) {
        // Do some extra checking on the request, and then...
        return companies.getAllCompanies(search);
    }

    @PostMapping("")
    void addNewCompany(@RequestBody Company company) {
        // Do some extra checking on the request, and then...
        companies.addCompany(company);
    }
}
```

Instead of doing it explicitly ->>

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;

public class DemoServ extends HttpServlet{
    public void doGet(HttpServletRequest req,
                      HttpServletResponse res)
            throws ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();

        String name=req.getParameter("name");
        pw.println("Welcome "+name);

        pw.close();
    }
}
```

from: https://www.javatpoint.com/servletrequest

Without inversion of Control

- Explicit initialisation of references

```
package pt.unl.fct.demo.controllers;

import org.springframework.web.bind.annotation.*;
import pt.unl.fct.demo.model.Company;
import pt.unl.fct.demo.services.CompaniesService;

@RestController
@RequestMapping(value="/companies")
public class CompaniesController {

    CompaniesService companies;

    public CompaniesController(CompaniesService companies) {
        this.companies = companies;
    }

    @GetMapping("")
    Iterable<Company> getAllCompanies(@RequestParam(required=false) String search) {
        // Do some extra checking on the request, and then...
        return companies.getAllCompanies(search);
    }

    @PostMapping("")
    void addNewCompany(@RequestBody Company company) {
        // Do some extra checking on the request, and then...
        companies.addCompany(company);
    }
}
```

resources in ruby'nrails
a whole DSL to define >>
routes

```
resources :photos
```

creates seven different routes in your application, all mapping to the `Photos` controller:

HTTP Verb	Path	Controller#Action	Used for
GET	/photos	photos#index	display a list of all photos
GET	/photos/new	photos#new	return an HTML form for creating a new photo
POST	/photos	photos#create	create a new photo
GET	/photos/:id	photos#show	display a specific photo
GET	/photos/:id/edit	photos#edit	return an HTML form for editing a photo
PATCH/PUT	/photos/:id	photos#update	update a specific photo
DELETE	/photos/:id	photos#destroy	delete a specific photo

https://guides.rubyonrails.org/routing.html

Using Dependency Injection

- Explicitly tell what things are, let spring do the wiring

```
@RestController
@RequestMapping("/")
class EmpController() {

    @Autowired
    lateinit var employees:EmployeeService

    @GetMapping("/api/departments/{id}/employees")
    fun employeesOfDepartment(
            @PathVariable id:String,
            @RequestParam search:String?
    )
```

Declare dependencies explicitly to be initialised by Spring

Using Dependency Injection

- Explicitly tell what things are, let spring do the wiring

```
@RestController
@RequestMapping("/")
class EmpController(val employees:EmployeeService) {

    @GetMapping("/api/departments/{id}/employees")
    fun employeesOfDepartment(
            @PathVariable id:String,
            @RequestParam search:String?
    )
```

Declare constructor dependencies and let Spring initialise correctly

Frameworks and MVC Architecture

- Java Spring is a component-based programming framework (based on configuration).
- It does the “plumbing," and lets components implement the “logic” of applications.
- How spring implements the MVC:

```
@SpringBootApplication
class McqApplication

fun main(args: Array<String>) {
    runApplication<McqApplication>(*args)
}
```

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC: (in Java)

```java
@Configuration
@EnableAutoConfiguration
@EnableWebMvc
@ComponentScan
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

`@Configuration`

Annotation specifies that the class has Bean definition methods

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC

```
@Configuration
@EnableAutoConfiguration
@EnableWebMvc
@ComponentScan
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

`@EnableAutoConfiguration`

Attempts to guess and configure beans that you are likely to need (doc.spring.io)

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC

```
@Configuration
@EnableAutoConfiguration
@EnableWebMvc
@ComponentScan
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

`@ComponentScan`

Configures component scanning.If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC (Here the view is an HTML (thymeleaf) template)

```
@Controller
public class GreetingController {

    private static final String template = "Hello, %s! %d”;
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public String greeting(@RequestParam(value="name", defaultValue="World") String name,
                           Model model) {
        long c = counter.incrementAndGet();
        model.addAttribute(“message”, String.format(template, name, c));
        return “greeting”;
    }
}
```

Frameworks and MVC Architecture

- Java Spring is a configuration and programming framework.
- It does the “plumbing”, and lets the components implement the “logic” of applications.
- How spring implements the MVC (Here the view is a JSON object formatter)

```
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name", defaultValue="World") String name) {
        return new Greeting(counter.incrementAndGet(),
                            String.format(template, name));
    }
}
```

Add-ons to the MVC Framework

- Resource control: DB connection & transactions

```
@Component
public class BookingService {

    private final static Logger logger = LoggerFactory.getLogger(BookingService.class);

    private final JdbcTemplate jdbcTemplate;

    public BookingService(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Transactional
    public void book(String... persons) {
        for (String person : persons) {
            logger.info("Booking " + person + " in a seat...");
            jdbcTemplate.update("insert into BOOKINGS(FIRST_NAME) values (?)", person);
        }
    }

    public List<String> findAllBookings() {
        return jdbcTemplate.query("select FIRST_NAME from BOOKINGS",
                (rs, rowNum) -> rs.getString("FIRST_NAME"));
    }

}
```

Add-ons to the MVC Framework

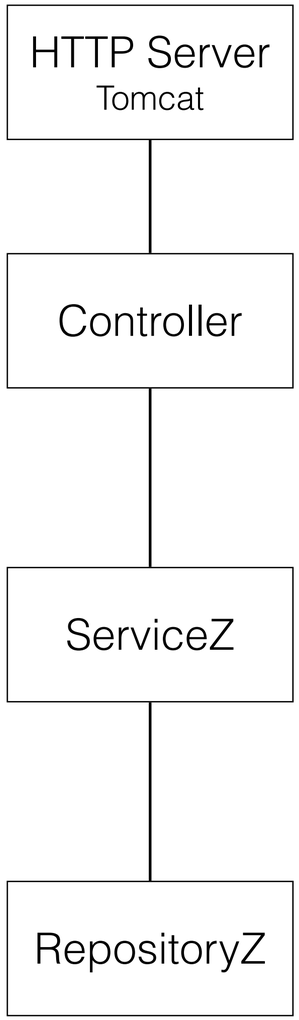
- Across application concerns: security

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
            .authorizeRequests()
                .antMatchers("/", "/home").permitAll()
                .anyRequest().authenticated()
                .and()
            .formLogin()
                .loginPage("/login")
                .permitAll()
                .and()
            .logout()
                .permitAll();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth
            .inMemoryAuthentication()
                .withUser("user").password("password").roles("USER");
    }
}
```

Example of an Architecture built with Spring

RECAP



- Spring is a component framework
- Resolves component dependencies by dependency injection
- Uses annotations to configure components

```
@RestController
@RequestMapping("/")
class EmpController(val employees:EmployeeService) {

   // http GET :8080/api/projects/2/team
    @GetMapping( "/api/projects/{id}/team")
    fun teamMembersOfProject(
            @PathVariable id:String
    )
    = employees.teamMembersOfProject(id)


}

@Service
class EmployeeService(val employees:EmployeeRepository) {
    fun teamMembersOfProject(id:String) = employees.findAll()
}

interface EmployeeRepository : CrudRepository<Employee, Long>
```

Architecture to the rescue of testers

- Unit tests should test components in isolation
- Defining the context for a component (correctly) is laborious and error prone
- Difficult to do with persistent data (must prepare tests)
- Impossible to do in tightly coupled structures
- Component frameworks allow mocking of dependencies

```
@SpringBootTest
@AutoConfigureMockMvc
open class RESTApplicationTests() {

    @Autowired lateinit var mvc: MockMvc
    @MockBean  lateinit var questions:QuestionRepository

    @Test
    fun `basic REST test`() {
        Mockito.`when`(questions.findAll()).thenReturn(l)

        mvc.perform(get(questionsURL))
                .andExpect(status().isOk)
    }
```

Architecture to the rescue of testers

```
@SpringBootTest
@AutoConfigureMockMvc
open class RESTApplicationTests() {

    @Autowired lateinit var mvc: MockMvc
    @MockBean  lateinit var questions:QuestionRepository

    @Test
    fun `basic REST test`() {
        Mockito.`when`(questions.findAll()).thenReturn(l

        mvc.perform(get(questionsURL))
                .andExpect(status().isOk)
    }
```

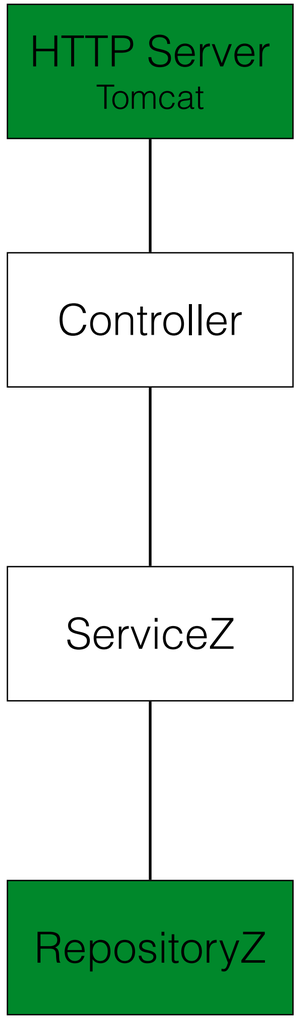
Replaces web server

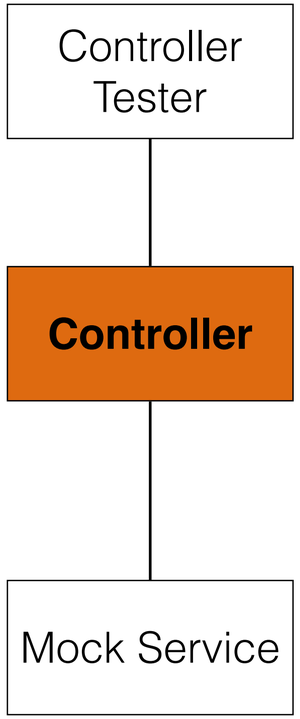
Replaces component with Mock object

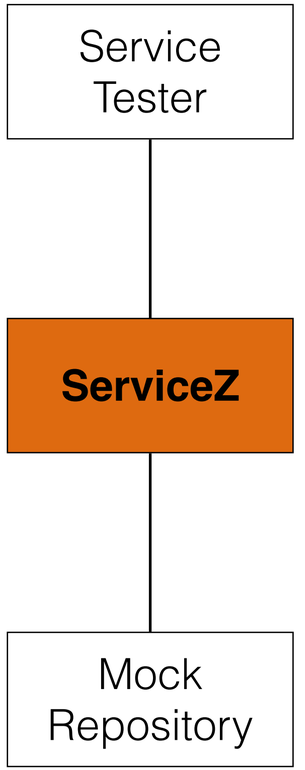
Replaces call results with expected values

Architecture to the rescue of testers

- Unit tests should test components in isolation
- Unit tests simulate inputs and compare outputs
- Mock components create a controlled context for each test or set of tests.







Internet Applications Design and Implementation 2020 - 2021

(Lecture 3 - Part 4 - MicroService Architecture: Service Discovery)

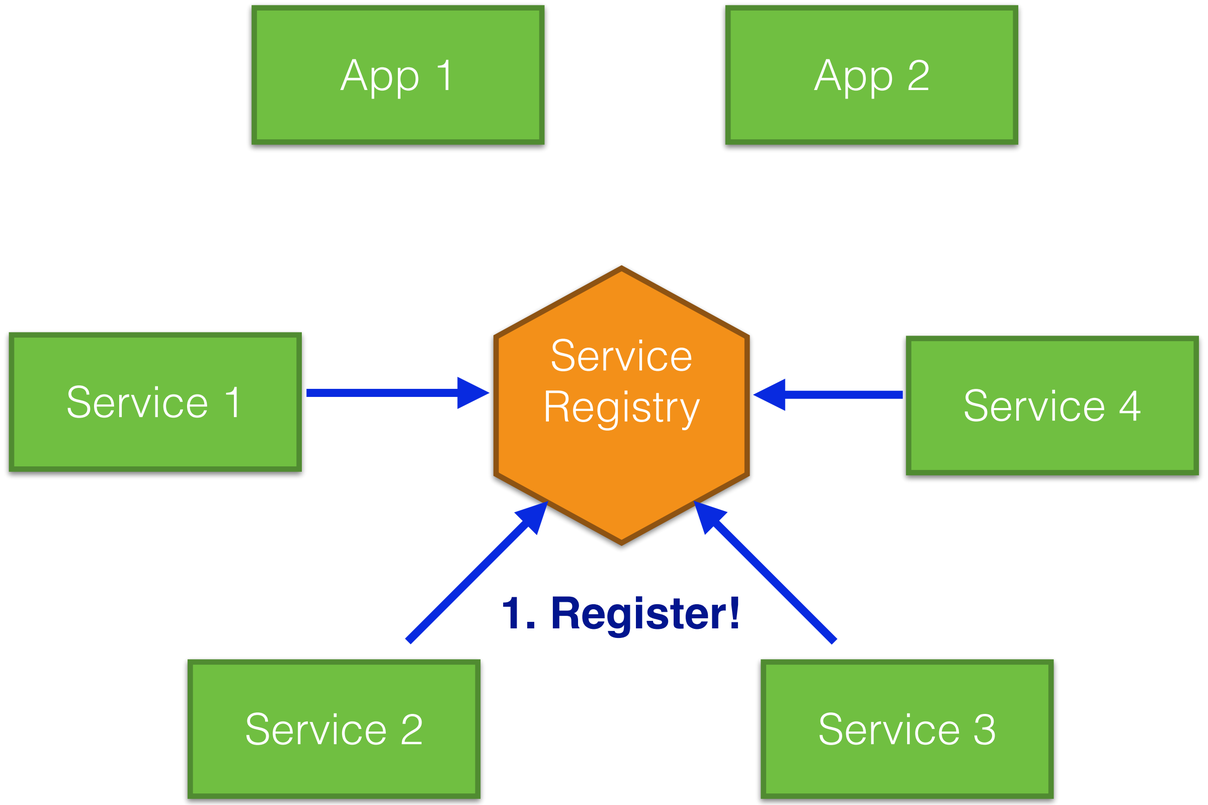
MIEI - Integrated Master in Computer Science and Informatics
Specialization block

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

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

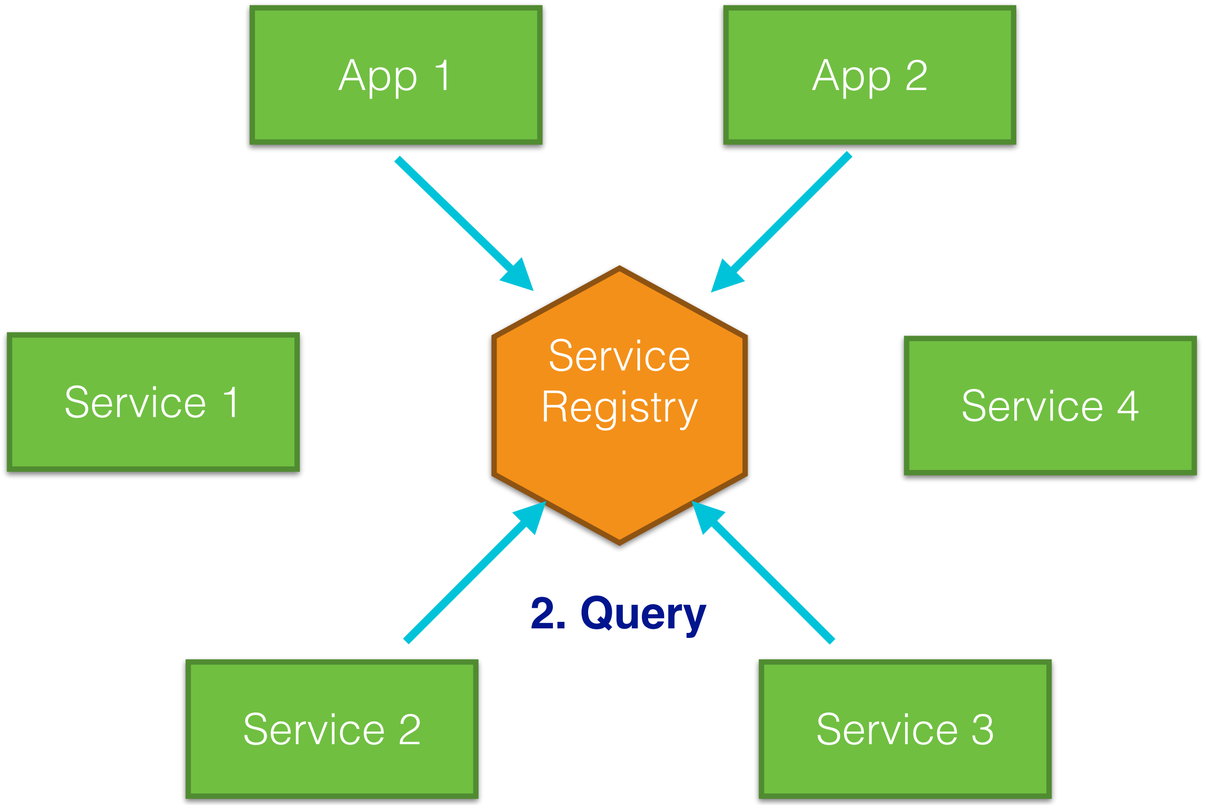
A Cloud Pattern: Service Discovery

- Example following Netflix Eureka
  - Provides services for Service Registry and Service Discovery
  - It relies on a single fixed point, which is the Eureka Server
- Netflix Eureka was reimplemented in Spring Cloud projects



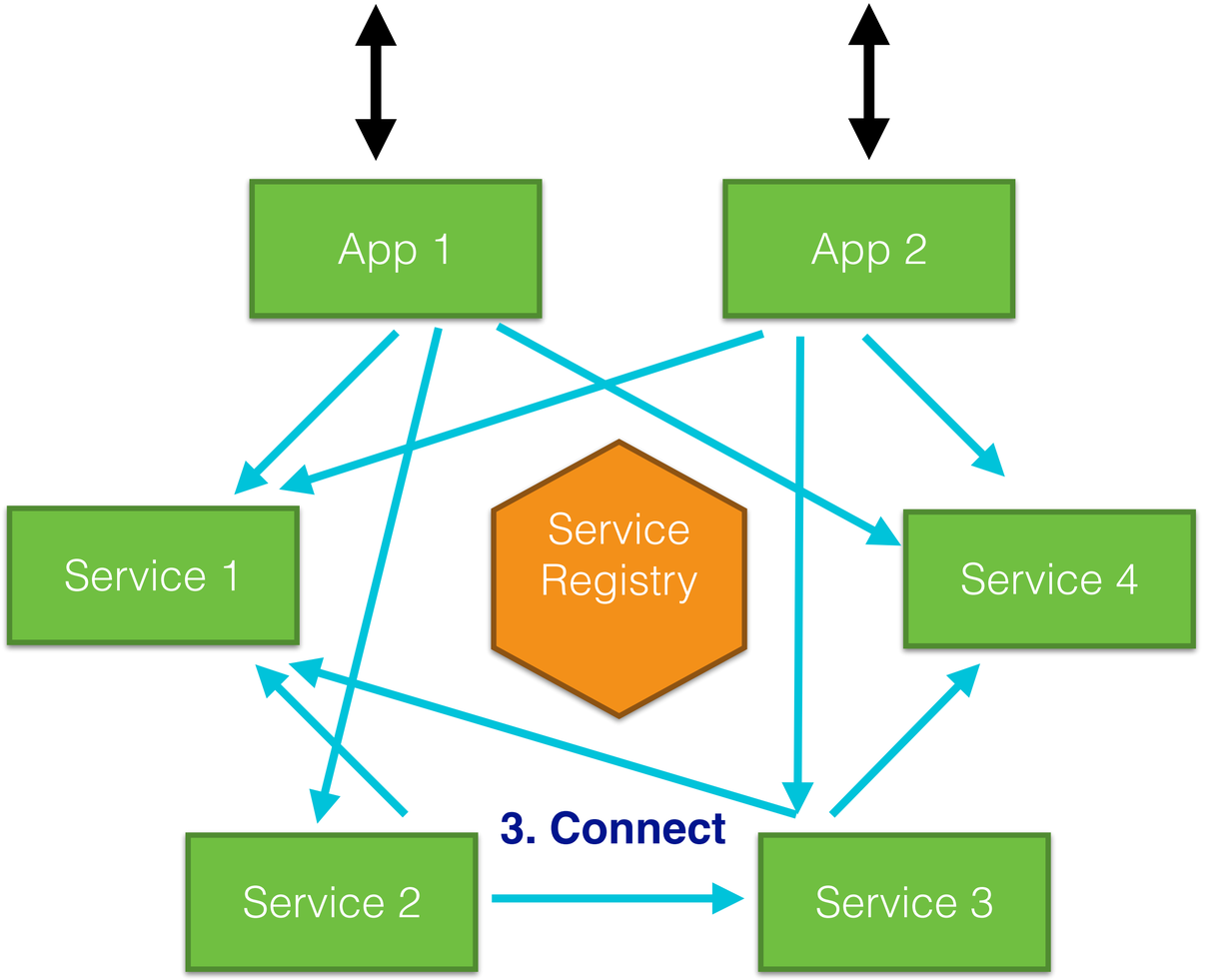
A Cloud Pattern: Service Discovery

- Example following Netflix Eureka
  - Provides services for Service Registry and Service Discovery
  - It relies on a single fixed point, which is the Eureka Server
- Netflix Eureka was reimplemented in Spring Cloud projects



A Cloud Pattern: Service Discovery

- Example following Netflix Eureka
  - Provides services for Service Registry and Service Discovery
  - It relies on a single fixed point, which is the Eureka Server
- Netflix Eureka was reimplemented in Spring Cloud projects



Spring Cloud 2023.0.3

Spring Cloud provides tools for developers to quickly build some of the common patterns in distributed systems (e.g. configuration management, service discovery, circuit breakers, intelligent routing, micro-proxy, control bus, short lived microservices and contract testing). Coordination of distributed systems leads to boiler plate patterns, and using Spring Cloud developers can quickly stand up services and applications that implement those patterns. They will work well in any distributed environment, including the developer's own laptop, bare metal data centres, and managed platforms such as Cloud Foundry.

Features

Spring Cloud focuses on providing good out of box experience for typical use cases and extensibility mechanism to cover others.

- Distributed/versioned configuration
- Service registration and discovery
- Routing

Eureka Server

- Add the dependency to the project (change to spring-boot 3.2.8).

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

- Enable the server in a configuration class

```
@SpringBootApplication
@EnableEurekaServer
class EurekaServerApplication
```

Needs a dep. manager for spring cloud

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-parent</artifactId>
            <version>2023.0.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Eureka Server

- Add the dependency to the project (change to spring-boot 3.2.8).

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

- Enable the server in a configuration class

```
@SpringBootApplication
@EnableEurekaServer
class EurekaServerApplication
```

Needs a configurations application.properties

```
spring.application.name=Eureka-server

server.port=8761

eureka.client.register-with-eureka= false
eureka.client.fetch-registry= false
```

Eureka Server HTTP API (Nor really REST)

Operation	HTTP action	Description
Register new application instance	POST /eureka/v2/apps/**appID**	Input: JSON/XML payload HTTP Code: 204 on success
De-register application instance	DELETE /eureka/v2/apps/**appID**/**instanceID**	HTTP Code: 200 on success
Send application instance heartbeat	PUT /eureka/v2/apps/**appID**/**instanceID**	HTTP Code: * 200 on success
Query for all instances	GET /eureka/v2/apps	HTTP Code: 200 on success Output: JSON/XML
Query for all **appID** instances	GET /eureka/v2/apps/**appID**	HTTP Code: 200 on success Output: JSON/XML
Query for a specific **appID**/**instanceID**	GET /eureka/v2/apps/**appID**/**instanceID**	HTTP Code: 200 on success Output: JSON/XML
Query for a specific **instanceID**	GET /eureka/v2/instances/**instanceID**	HTTP Code: 200 on success Output: JSON/XML
Take instance out of service	PUT /eureka/v2/apps/**appID**/**instanceID**/status?value=OUT_OF_SERVICE	HTTP Code: * 200 on success
Move instance back into service (remove override)	DELETE /eureka/v2/apps/**appID**/**instanceID**/status?value=UP (The value=UP is optional, it is used as a suggestion for the fallback status	HTTP Code: * 200 on success
Update metadata	PUT /eureka/v2/apps/**appID**/**instanceID**/metadata?key=value	HTTP Code: * 200 on success
Query for all instances under a particular **vip address**	GET /eureka/v2/vips/**vipAddress**	* HTTP Code: 200 on success Output: JSON/XML * 404 if the **vipAddress** does not exist.
Query for all instances under a particular **secure vip address**	GET /eureka/v2/svips/**svipAddress**	* HTTP Code: 200 on success Output: JSON/XML * 404 if the **svipAddress** does not exist.

Eureka Server UI

spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2024-10-06T17:59:13 +0100
Data center	default	Uptime	01:11
		Lease expiration enabled	true
		Renews threshold	8
		Renews (last min)	16

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
ORDERSERVICE	n/a (1)	(1)	UP (1) - joaos-air.lan:OrderService:0
PRODUCTSERVICE	n/a (1)	(1)	UP (1) - joaos-air.lan:ProductService:0
SOCIALAPP	n/a (1)	(1)	UP (1) - joaos-air.lan:SocialApp:8080
USERSERVICE	n/a (1)	(1)	UP (1) - joaos-air.lan:UserService:0

Eureka Services

- Add the dependency to the project (change to spring-boot 3.2.8).

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

- Configure to find the Eureka Server

```
spring.application.name=UserService
server.port=0
eureka.client.serviceUrl.defaultZone=${EUREKA_URI:http://localhost:8761/eureka/}
eureka.instance.prefer-ip-address=true
```

Eureka Services

- Implement the API as usual, isolate API in an interface

```
interface HelloAPI {
    @GetMapping("/hello")
    fun hello(): String
}


@RestController
class UserController(var eurekaClient: EurekaClient) : HelloAPI {
    @Value("\${spring.application.name}")
    var appName: String? = null

    override fun hello(): String {
        return "Hello World! from ${eurekaClient.getApplication(appName).getName()}"
    }
}
```

Eureka Clients (Feign - a declarative client)

- Add the dependency to the project (change to spring-boot 3.2.8).

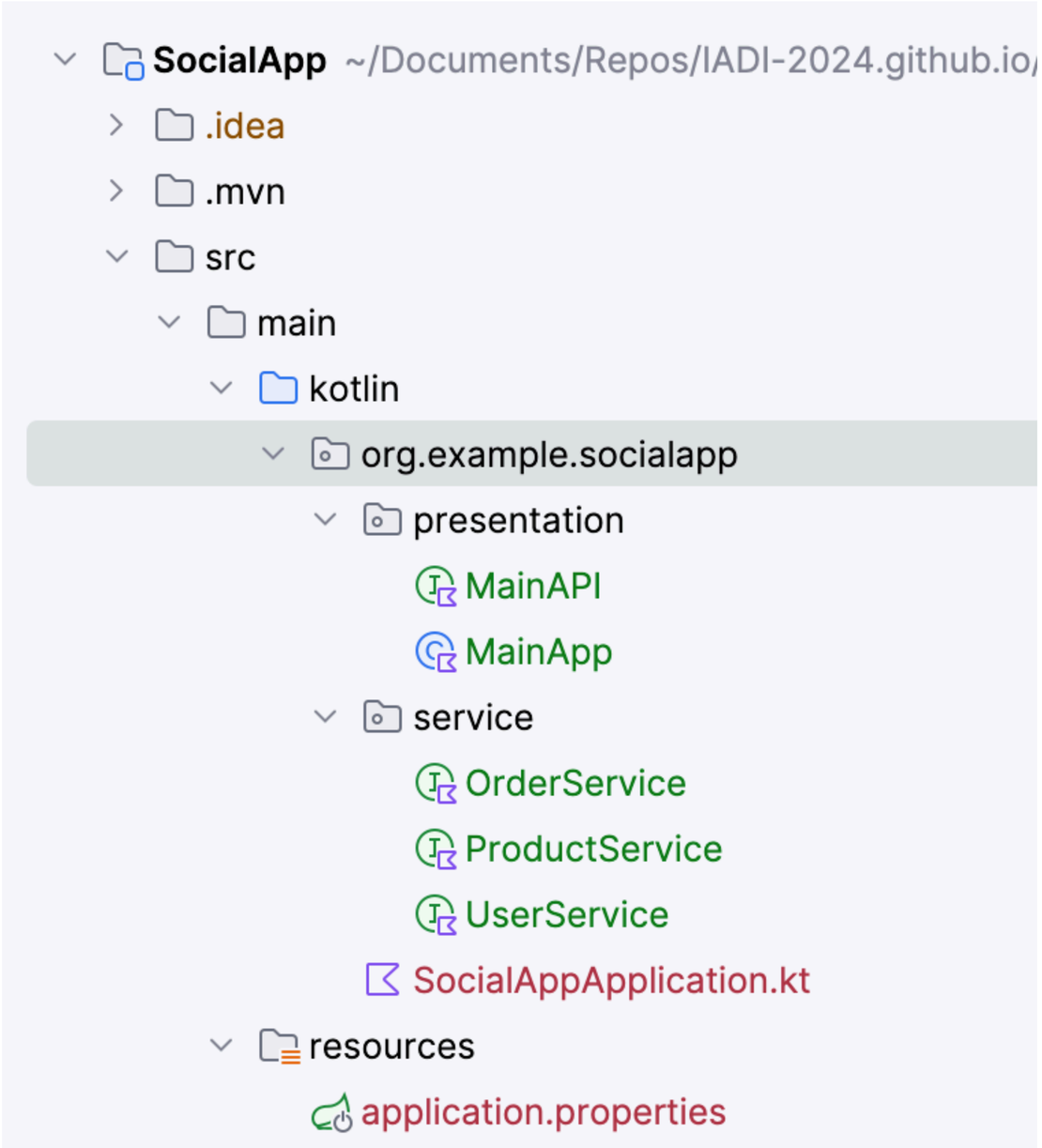
```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

- Configure to find the Eureka Server and provide an API in a fixed port (8080)

```
spring.application.name=SocialApp

server.port=8080
eureka.client.serviceUrl.defaultZone=${EUREKA_URI:http://localhost:8761/eureka}
```

Eureka Clients

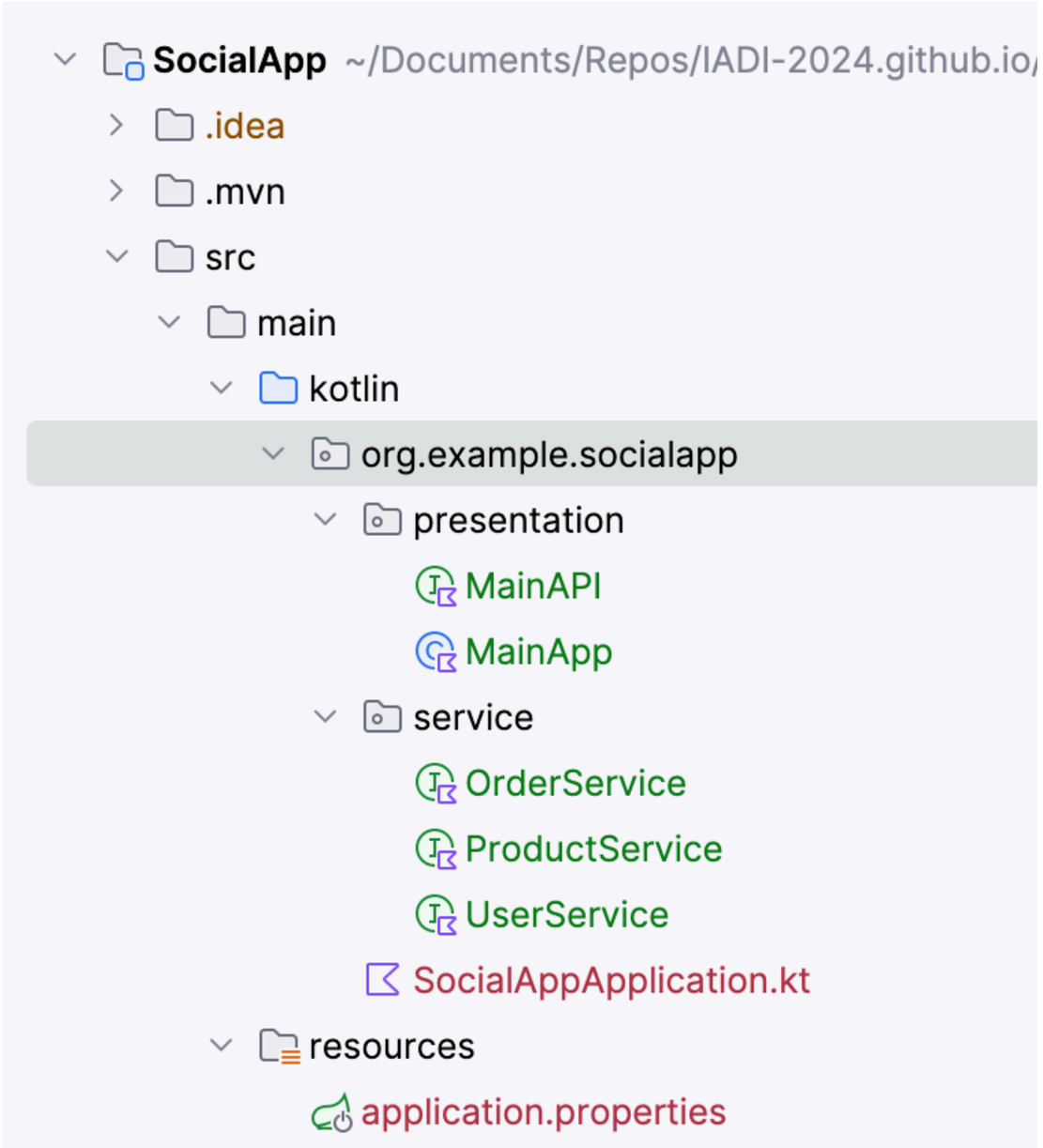


Implement the application in a layered architecture as usual.

```
@RestController
class MainApp(
    val userClient: UserService,
    val orderService: OrderService,
    val productService: ProductService
) : MainAPI {

    override fun hello() =
        listOf(userClient.hello(),
               orderService.hello(),
               productService.hello())
}
```

Eureka Clients



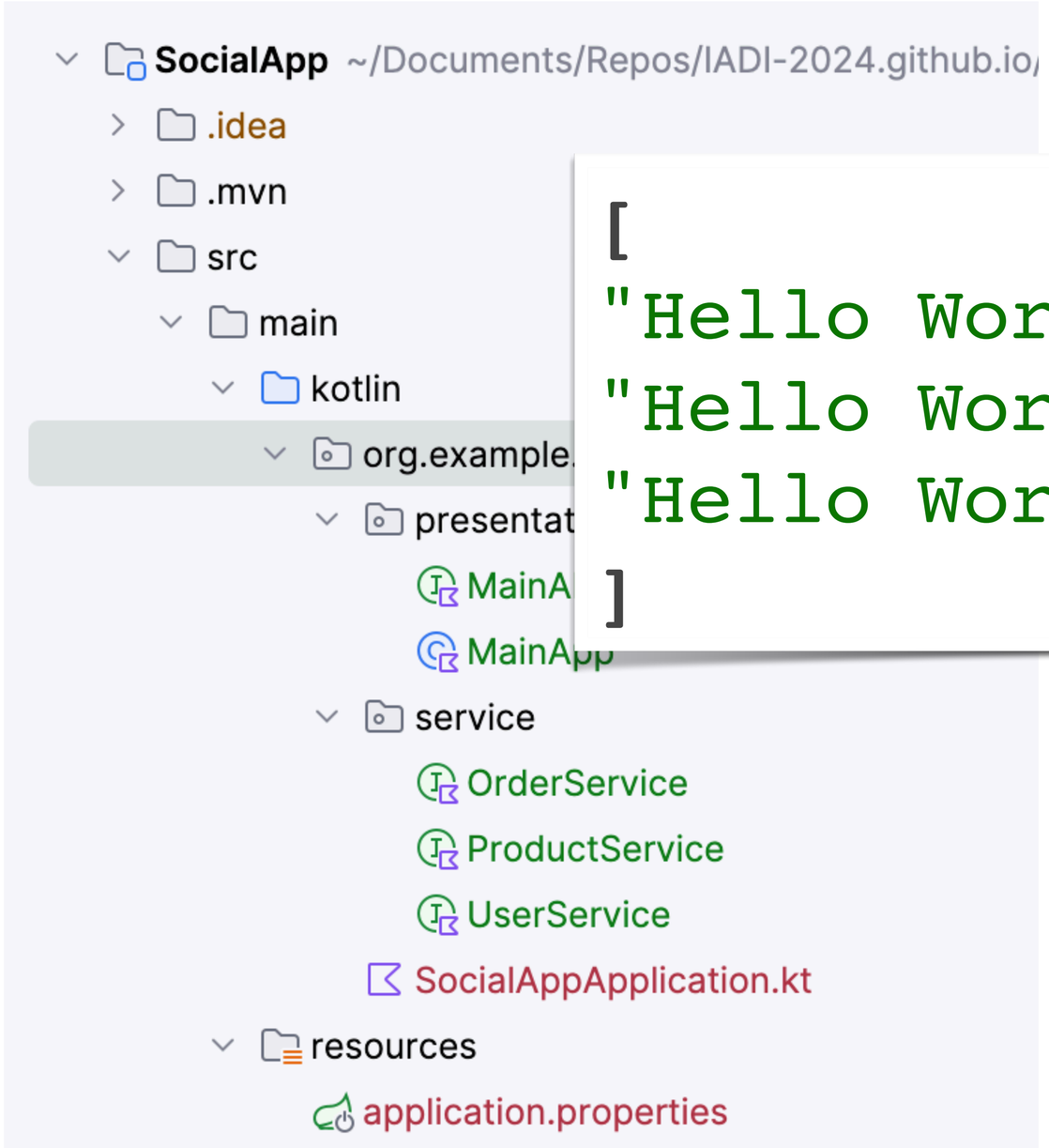
Reuse the interfaces to make a connector.

```kotlin
@FeignClient("UserService")
interface UserService {
    @GetMapping("/hello")
    fun hello(): String
}

@FeignClient("OrderService")
interface OrderService {
    @GetMapping("/hello")
    fun hello(): String
}

@FeignClient("ProductService")
interface ProductService {
    @GetMapping("/hello")
    fun hello(): String
}
```

Eureka Clients



Reuse the interfaces to make a connector.

```
[
"Hello World! from USERSERVICE",
"Hello World! from ORDERSERVICE",
"Hello World! from PRODUCTSERVICE"
]
```

```
    @GetMapping("/hello")
    fun hello(): String
}

@FeignClient("ProductService")
interface ProductService {
    @GetMapping("/hello")
    fun hello(): String
}
```