Internet Applications Design and Implementation
2020 - 2021

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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

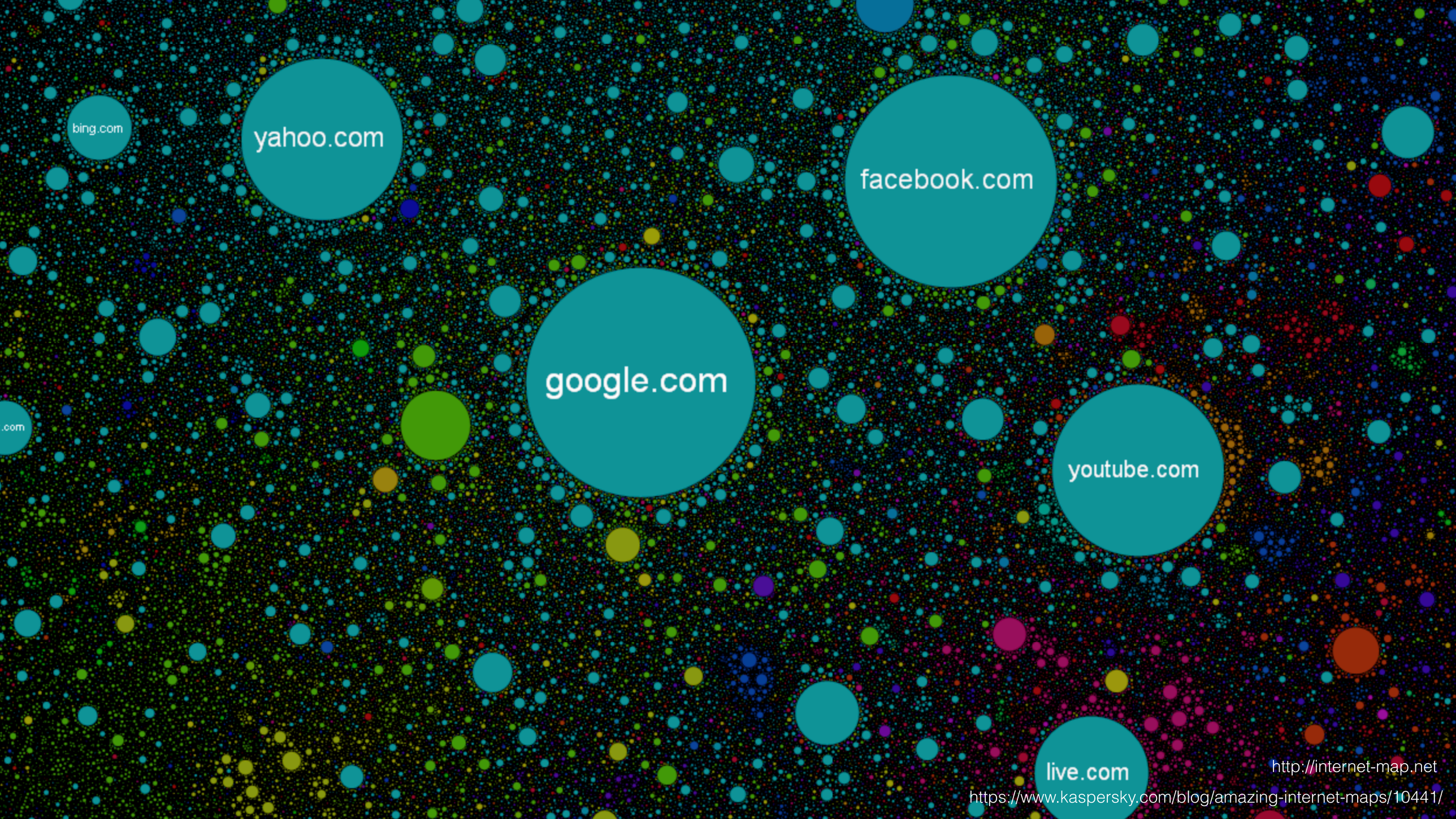
Internet Applications Design and Implementation
2020 - 2021
(Introductory Video and Logistics)

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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA


bing.com
yahoo.com
facebook.com
google.com
youtube.com
live.com
http://internet-map.net
https://www.kaspersky.com/blog/amazing-internet-maps/10441/

Internet Applications Design and Implementation

Internet application | ˈɪntənɛt aplɪˈkeɪʃ(ə)n |, noun

1. Any application that uses the internet to consume and/or provide services and data.

Internet Applications Design and Implementation

Web application | wɛb aplɪ'keɪʃ(ə)n |, noun

1. An internet application that runs on a browser and obtains HTML and data pages from a web server.

Mobile application | 'məʊbʌɪl aplɪ'keɪʃ(ə)n |, noun

2. An application installed and running in a mobile device, usually dependent on web services as data sources.

Web service | wɛb 'səːvɪs |, noun

3. A computational procedure available on the Internet to provide services and data to other apps and services. Usually associated to binding technologies like SOAP or REST.

Internet Applications Design and Implementation

Cloud application | klaʊd aplɪˈkeɪʃ(ə)n |, noun

1. An internet application deployed on an independent hosting service, providing computation and additional resources and features like replication and storage.

Service-based architecture | ˈsəːvɪs beɪs ˈɑːkɪtɛktʃə |, noun

2. A conceptual structure and logical organisation of web services, usually implemented using orchestration languages and tools.

Data-Centric Applications | Dataˈdeɪtə-ˈsɛntrɪk ˌæplɪˈkeɪʃ(ə)nz |, noun

3. Applications that are developed primarily based on resource-based rules, making their data available to others through well defined interfaces

Internet Applications Design and Implementation

The challenge is to **design**, **specify**, and **implement**

modular,
loosely-coupled,
large-scale applications,

so that the software development process is more efficient and the longevity of the applications is longer.

Internet Applications Design and Implementation

- Goals:
  - Faster development cycles,
  - functionally correct applications,
  - applications that can easily be used by other systems,
  - heterogeneous development and execution environments,
  - applications are easily maintainable (corrections and extensions)
  - secure systems
  - reliable and available systems
  - performant applications

Internet Applications Design and Implementation

- Goals:
  - Faster development cycles,
  - functionally correct applications
  - secure systems
  - reliable and available systems
  - performant applications

Many of these goals are specific of Internet Applications and should be attained using specific methods, tools and techniques.

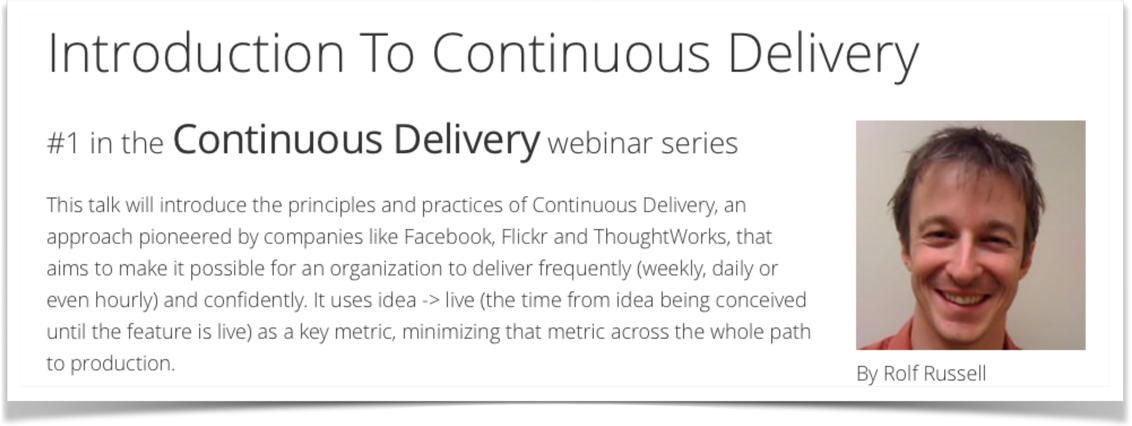
What about Internet Applications?

Features of Internet Applications

- Everything is (inter)connected and developing software for interconnection requires special skills and methods
- Internet Apps can be:
  - Standalone / Desktop with native interfaces (RPC) or http connections
  - Web, via http, accessible through browsers
  - Mobile native or PWAs (connected via http of native, w/ offline capabilities)
  - Compound and orchestrated services
  - Mash-up interfaces
    (e.g. google maps + rentals, friends, …http://mashable.com/2009/10/08/top-mashups/)

Skills for Internet Applications

- Special skills because
  - software evolves fast, we need to develop extensible and maintainable software.
  - data and code have different wills, we need to develop software in sync with data.
  - software is “open” and connected (the world changes), we need to develop software that is loosely coupled, robust and follows standard API conventions.
  - Internet Apps may grow to have a huge user base (milions of requests)…
  - It also needs to be scalable, secure, replicated, reliable, concurrent, safe…

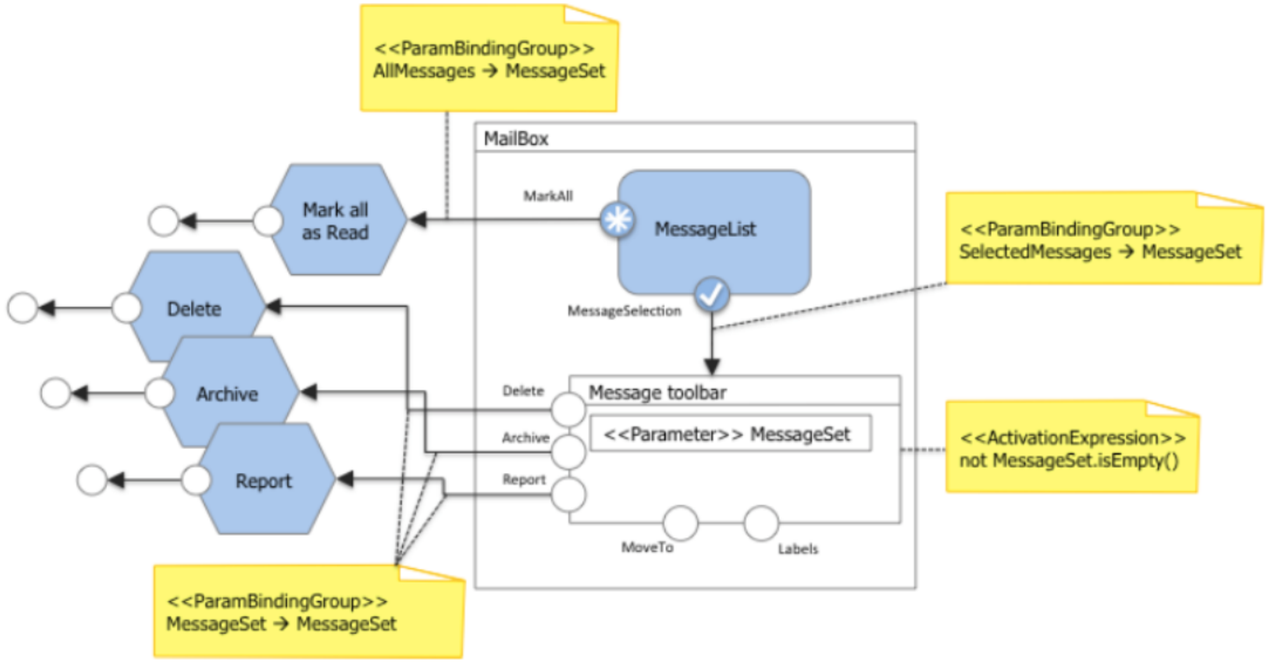


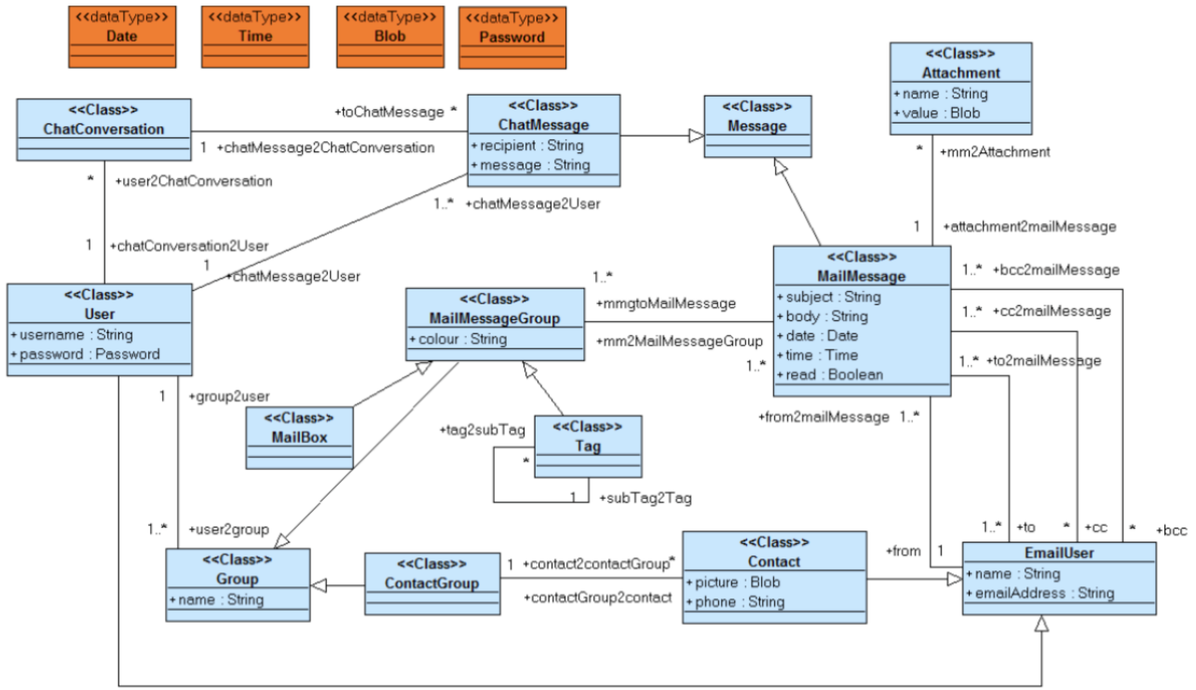
“Tools” for Internet Applications

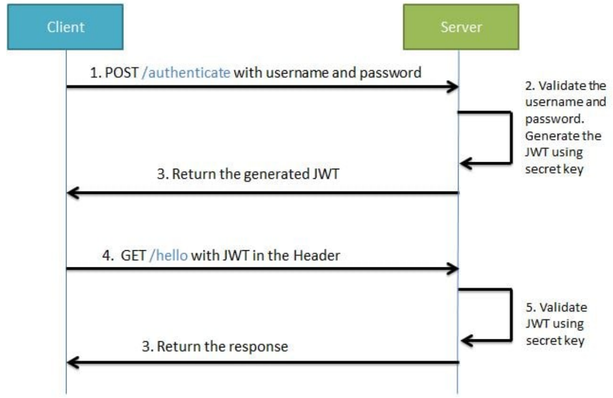
- Software Specification (UML, OpenAPI, IFML)
- Software Architecture (Web MVC, SOA, Micro Services)
- Software Patterns (Observer, Chain of Respons., Facade, Builder, …)
- Programming Languages (Statically Typed, Dynamic, Concurrent, …)
- Software Frameworks (Web, Component, Reactive, Data-Abstraction,…)
- Development Methods (Agile, TDD, BDD, Continuous Integration, C. Delivery)
- Deployment Tools (Cloud Managers, Containers,)

Software Specification for Internet Applications

- Data schema specification
- Behaviour specification
- Interface-flow specification
- Security specification

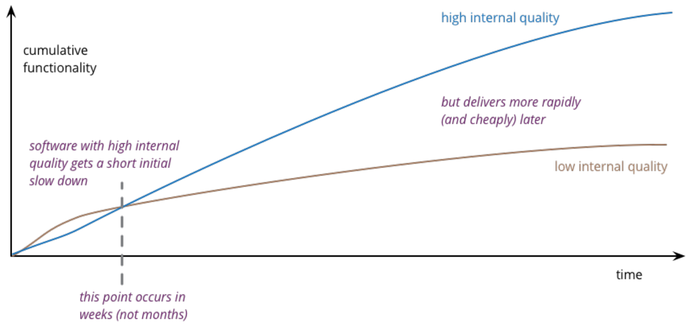




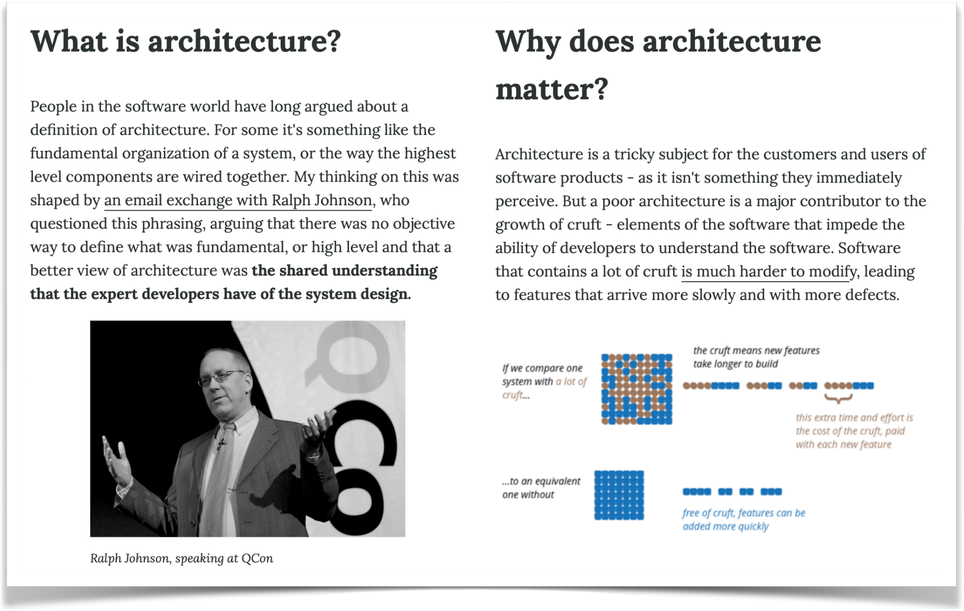


Software Architecture for Internet Applications

- Architecture refers to the internal design of a software system
- It describes the way the highest level components are wired together.
- A good architecture allows a system to be expanded with new capabilities
- A good architecture pays off in quality



in martinfowler.com/architecture/



What is architecture?

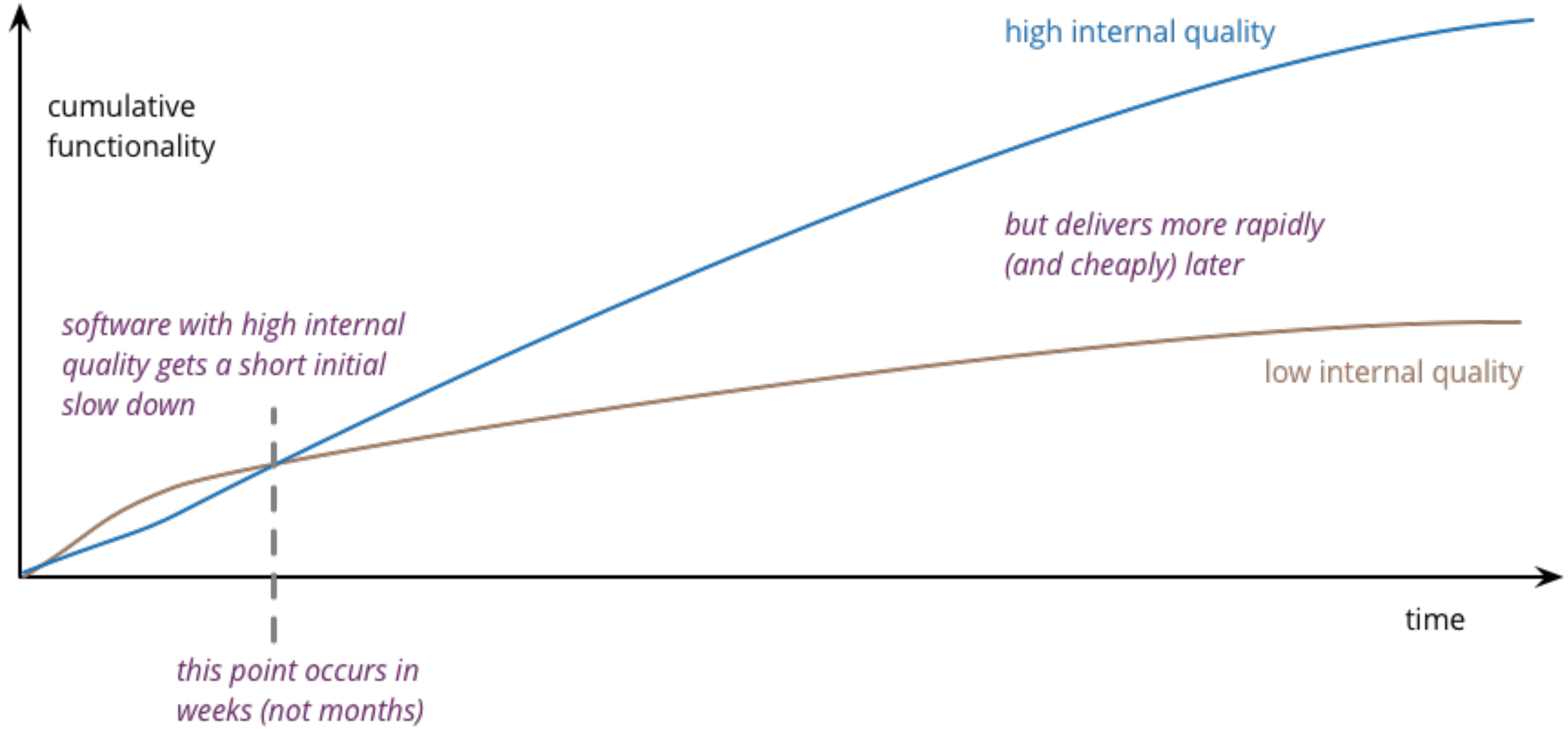
People in the software world have long argued about a definition of architecture. For some it's something like the fundamental organization of a system, or the way the highest level components are wired together. My thinking on this was shaped by an email exchange with Ralph Johnson, who questioned this phrasing, arguing that there was no objective way to define what was fundamental, or high level and that a better view of architecture was **the shared understanding that the expert developers have of the system design.**

Ralph Johnson, speaking at QCon

Why does architecture matter?

Architecture is a tricky subject for the customers and users of software products - as it isn't something they immediately perceive. But a poor architecture is a major contributor to the growth of cruft - elements of the software that impede the ability of developers to understand the software. Software that contains a lot of cruft is much harder to modify, leading to features that arrive more slowly and with more defects.

Software Architecture for Internet Applications



in martinfowler.com/architecture/

Software Architecture for Internet Applications

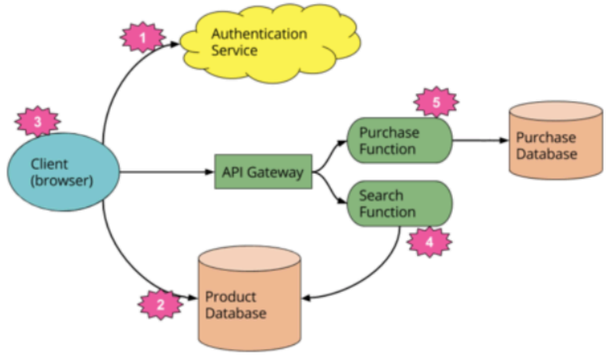
Application Boundary

One of the undecided problems of software development is deciding what the boundaries of a piece of software is. (Is a browser part of an operating system or not?) Many proponents of Service Oriented Architecture believe that applications are going away - thus future enterprise software development will be about

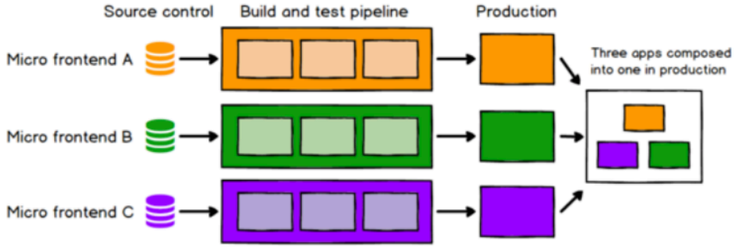
Microservices Guide

The microservice architectural pattern is an approach to developing a single application as a

Serverless Architectures



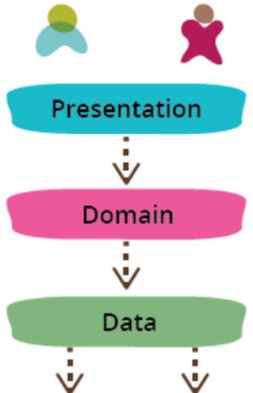
Micro Frontends



GUI Architectures

In the mid 2000s I was pursuing a couple writing projects that could have turned into books, but haven't yet made it. One was on the architecture of user interfaces. As part of this work, I drafted a description of how GUI architectures evolved, comparing the default approach of Forms and Controls with the the Model-View-Controller (MVC) pattern. MVC is one of the most ill-understood patterns in the software world

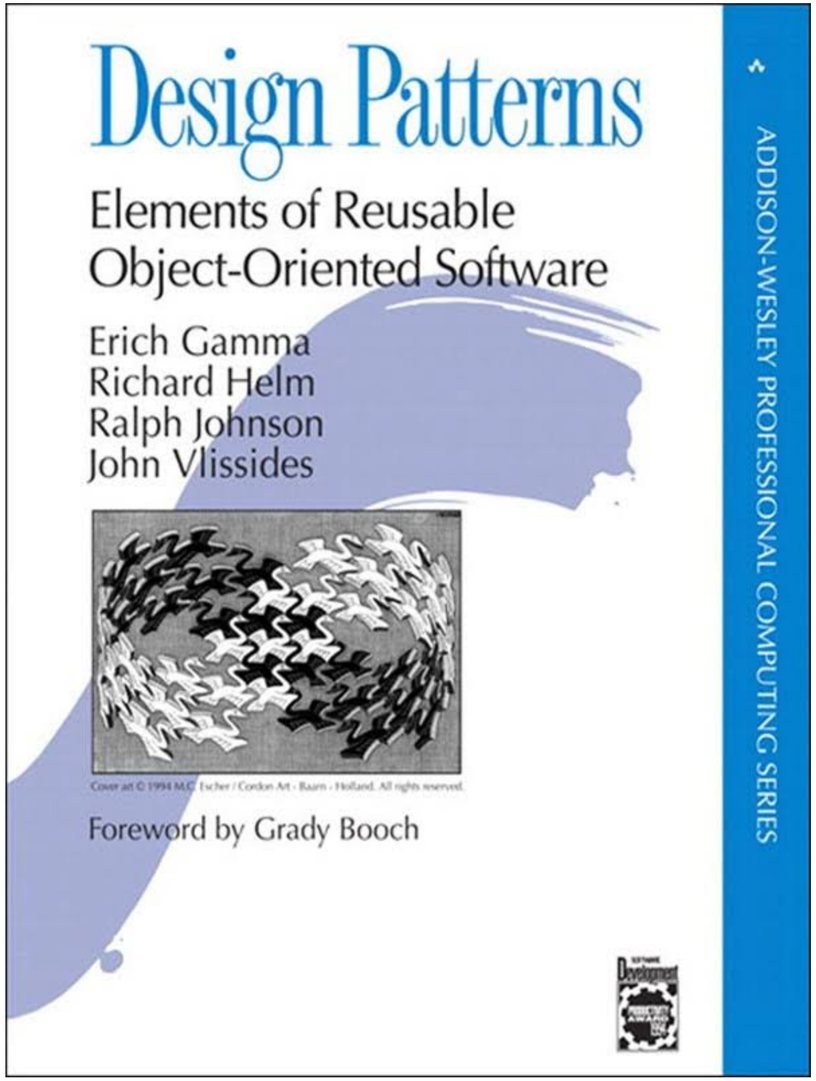
Presentation Domain Data Layering



in martinfowler.com

(Design) Patterns for Internet Applications

- Observer (Web Controllers, Reactive web)
- Chain of Responsibility (Security filters on requests)
- Facade (Service objects, REST interfaces, ORMs)
- Builders (inner to outer representations of data)
- …
- Frameworks implement design and architectural patterns and styles
  - Layered Architecture
  - Model View Controller
  - Inversion of control
  - REST interfaces



Software Frameworks for Internet Applications

- provide **re-usable code** (as libraries)
- provides **controlled and managed resources**
  - e.g. transactions, log, security
- introduce **abstraction layers**
  - to hide complexity of concrete execution scenarios (hardware/software configuration)
  - Sometimes by scaffolding code (does not evolve well)
- support **test driven development** and **code evolution**
- They work by explicit **configuration** or by **conventions**
- **Improve the overall quality! (correction, maintainability, extensibility)**

Spring Initializr
Bootstrap your application

Github Twitter Help

Project: **Maven Project** | Gradle Project

Language: Java | **Kotlin** | Groovy

Spring Boot: 2.2.0 M5 | 2.2.0 (SNAPSHOT) | 2.1.9 (SNAPSHOT) | **2.1.8**

Dependencies

Start projects with

Developer Tools

Spring Boot DevTools
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok
Java annotation library which helps to reduce boilerplate code.

Spring Configuration Processor
Generate metadata for developers to offer contextual help and “code completion” when working with custom configuration keys (ex.application.properties/.yml files).

Web

Spring Web
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default

Spring Reactive Web
Build reactive web applications with Spring WebFlux and Netty.

Rest Repositories
Exposing Spring Data repositories over REST via Spring Data REST.

Software Frameworks for Internet Applications

- provide **re-usable code** (as libraries)
- provides **controlled and managed resources**
  - e.g. transactions, log, security

```
UserTransaction utx = entityManager.getTransaction();

try {
    utx.begin();
    businessLogic();
    utx.commit();
} catch(Exception ex) {
    utx.rollback();
    throw ex;
}
```

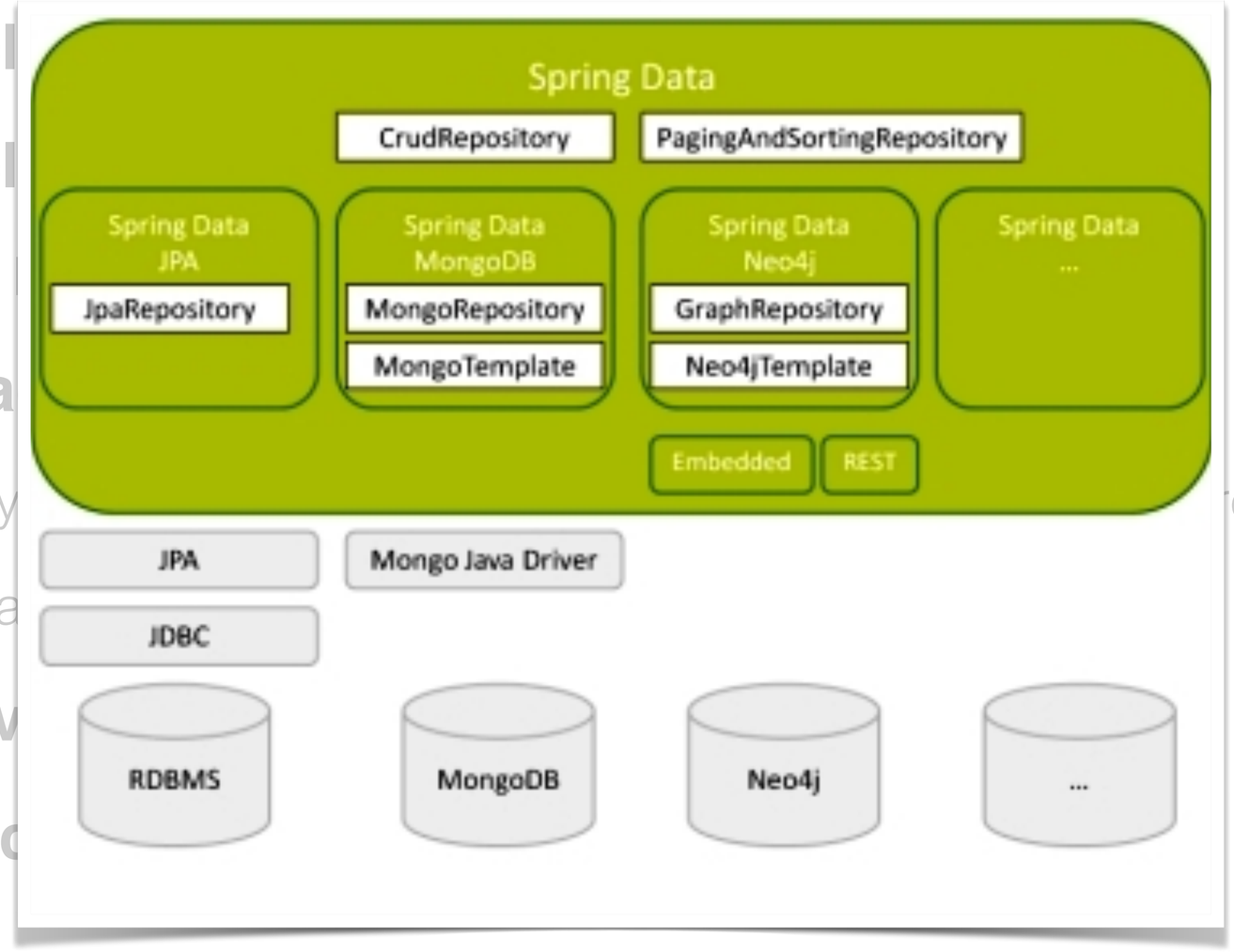
```
@Transactional
public void businessLogic() {
... use entity manager inside a transaction ...
}
```

- it's repetitive and error prone
- any error can have a very high impact
- errors are hard to debug and reproduce
- this decreases the readability of the code base
- What if this method calls another transactional method?

https://dzone.com/articles/how-does-spring-transactional

Software Frameworks for Internet Applications

- provide **re-usabl**
- provides **control**
  - e.g. transactions,
- introduce **abstra**
  - to hide complexity
  - Sometimes by sca
- support **test driv**
- work by explicit **c**



e configuration)

https://www.infoq.com/articles/spring-data-intro/

A spectrum of frameworks for a spectrum of domains

- Web Frameworks
- Data manipulation Frameworks
- Resource Oriented Frameworks (REST)
- Process Oriented Frameworks
- Client-side frameworks

Software Frameworks for Internet Applications

- Examples of Web Frameworks:
  - Ruby on Rails, Django and Python, Spring and Java, Cake and PHP, nodeJS, Meteor, Revel and Go
- Examples of Client Frameworks:
  - AngularJS, React, Vue, Lightweight form
- Examples of Data Frameworks:
  - LINQ, Hibernate (JPA), …
- Examples of Web Service Frameworks/Languages:
  - WS-BPEL…
- Examples of Multi-tier Languages:
  - Ocaml (w/ Ocsigen), Elm, LINKS, UrWeb, Loom

Software Frameworks for Web Applications

- Python (+Django)
  - Easy-to-learn programming language
  - Easy-to-use data structures
  - Available libraries

Software Frameworks for Web Applications

- Based on Ruby (dynamically typed language)
  - Implements the MVC architectural pattern
  - Pattern components assembled by **conventions** on folders, filenames, and language identifiers
- Very flexible programming language
  - Everything is “re-programmable”
- Rails is a versatile tool
  - Support for scaffolding, migrating code and data, and TDD
- Convention over configuration (even more)
- Big community
  - Big pool of top-of-the-line gems

Software Frameworks for Web Applications

- Java + J2EE / Spring / Play
- Most used programming language
  - Statically typed and dynamically assembled
  - Well know
  - Easy to start web development
- Huge amount of ready-to-use libraries (Beans)
- Industrial grade efficient implementations
  - Beans framework (MVC is just a module - webmvc)
  - Patterns are assembled **programatically**, or by XML **configuration** files

Software Frameworks for Web Applications

- Elixir + Phoenix is a web development framework written in Elixir which implements the server-side Model View Controller (MVC) pattern.
- Provides high developer productivity and high application performance
- Scaffolding tools like Rails and Django
- Provides LiveView
  - based on web sockets
  - efficiently handle cross border events
  - efficiently refresh web pages based on server side templates

Software Frameworks for Web Applications

- Spring + Kotlin
- Advantages:
  - It's Completely Interoperable With Java
  - It's (way) More Concise Than Java
  - Safer Code
  - It Comes With a Smarter and Safer Compiler
  - It's Easier to Maintain
  - It's Been Created to Boost Your Productivity
  - It “Spoils” You with Better Support for Functional Programming
  - It Has Null in Its Type System

Ktor

VERT.X

https://dzone.com/articles/what-are-the-biggest-advantages-of-kotlin-over-jav

https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0

Jobs for Internet Applications

A **software architect** is a **software** developer expert who makes high-level design choices and dictates technical standards, including **software** coding standards, tools, and platforms.

Software architect - Wikipedia

https://en.wikipedia.org › wiki › Software_architect

About Featured Snippets Feedback

The **2019 Roadmap** To Fullstack **Web Development**

Gift This Course Wishlist

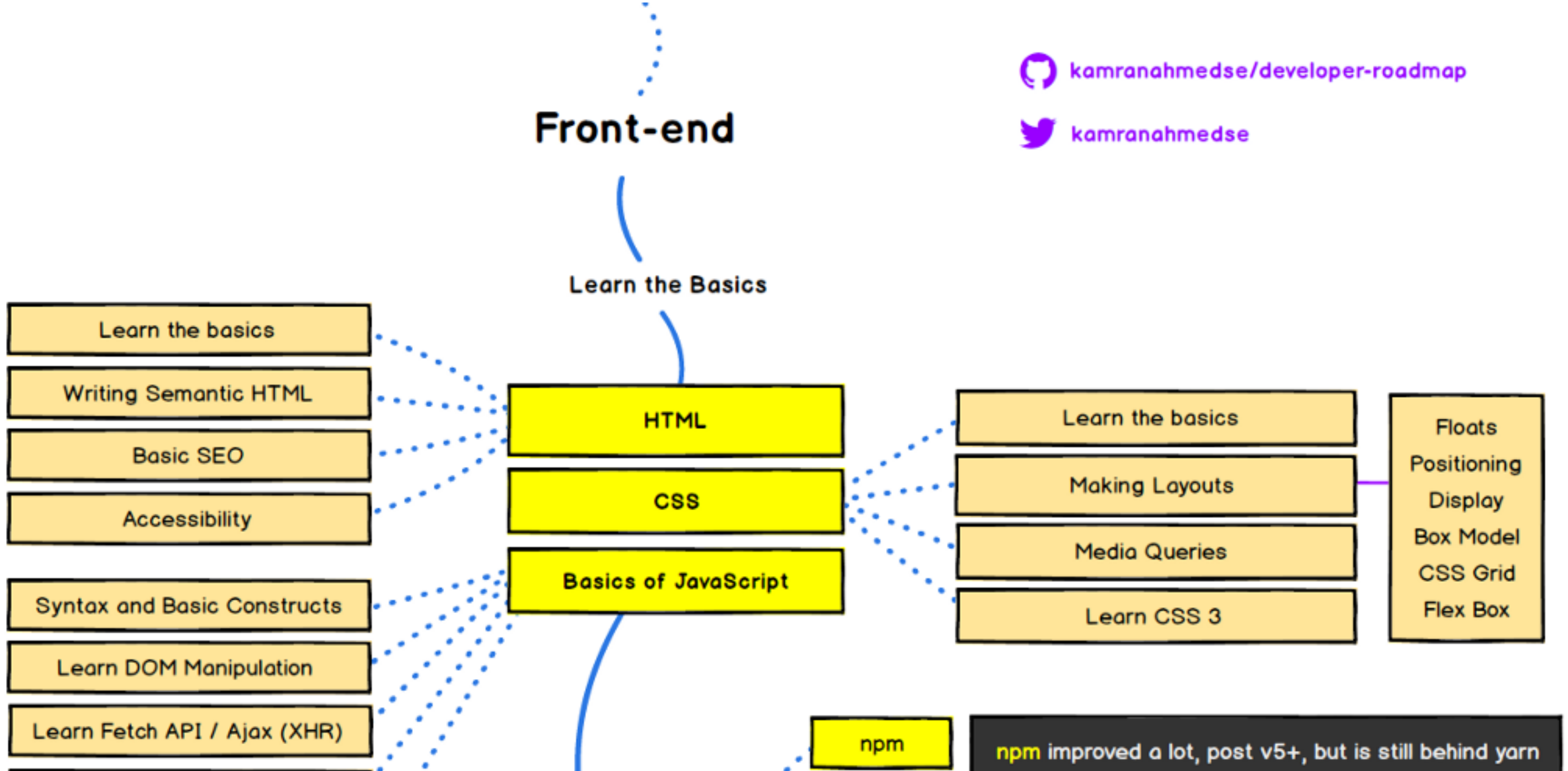
Go Full Stack with Spring Boot and React

Build Your First Full Stack Application with React and Spring Boot. Become a Full Stack Web Developer Now!

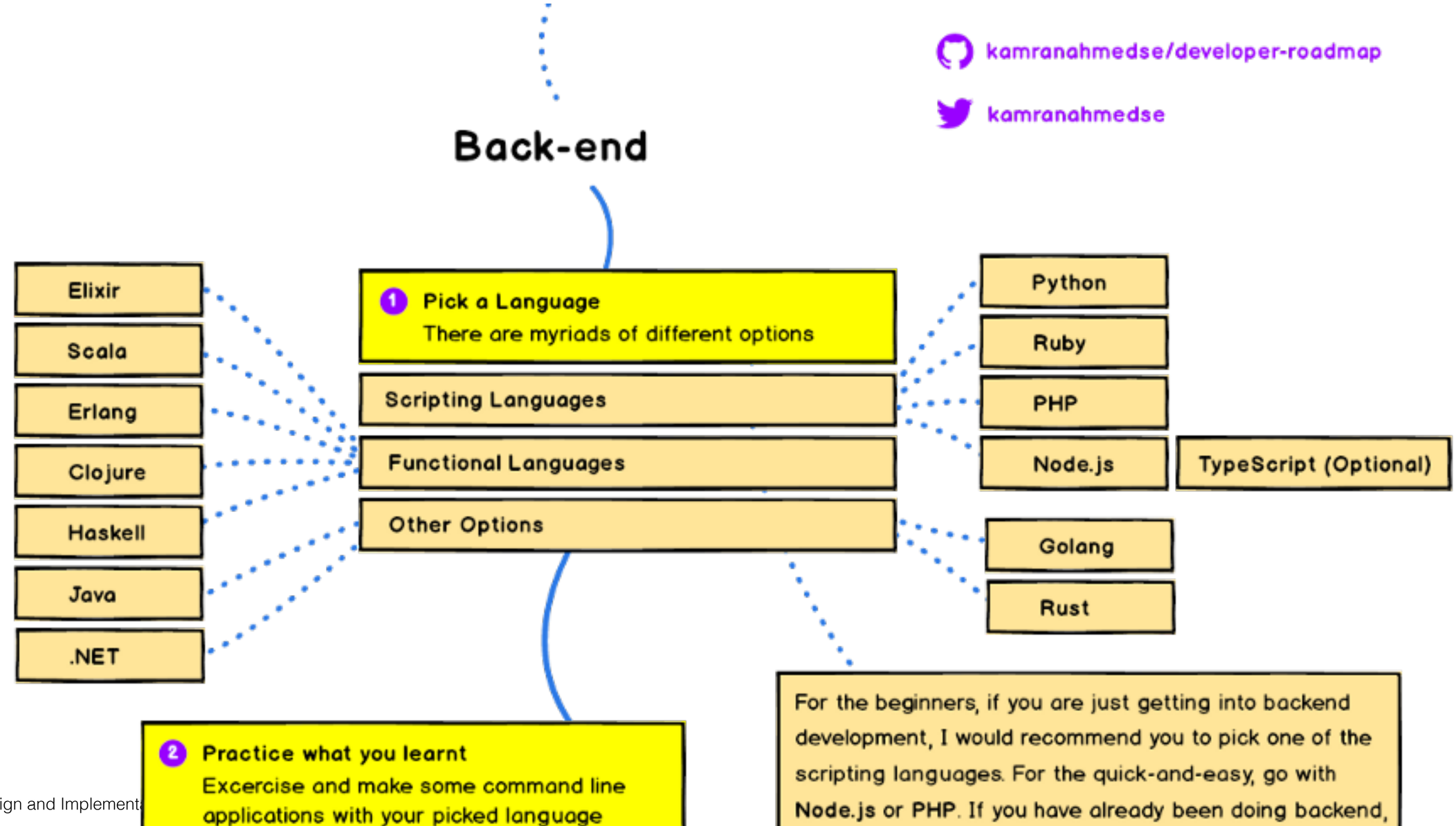
4.5 (378 ratings) 1,918 students enrolled

Created by in28Minutes Official Last updated 8/2019 English English

Programming Languages for Internet Applications



Programming Languages for Internet Applications



Development Methods for Internet Applications

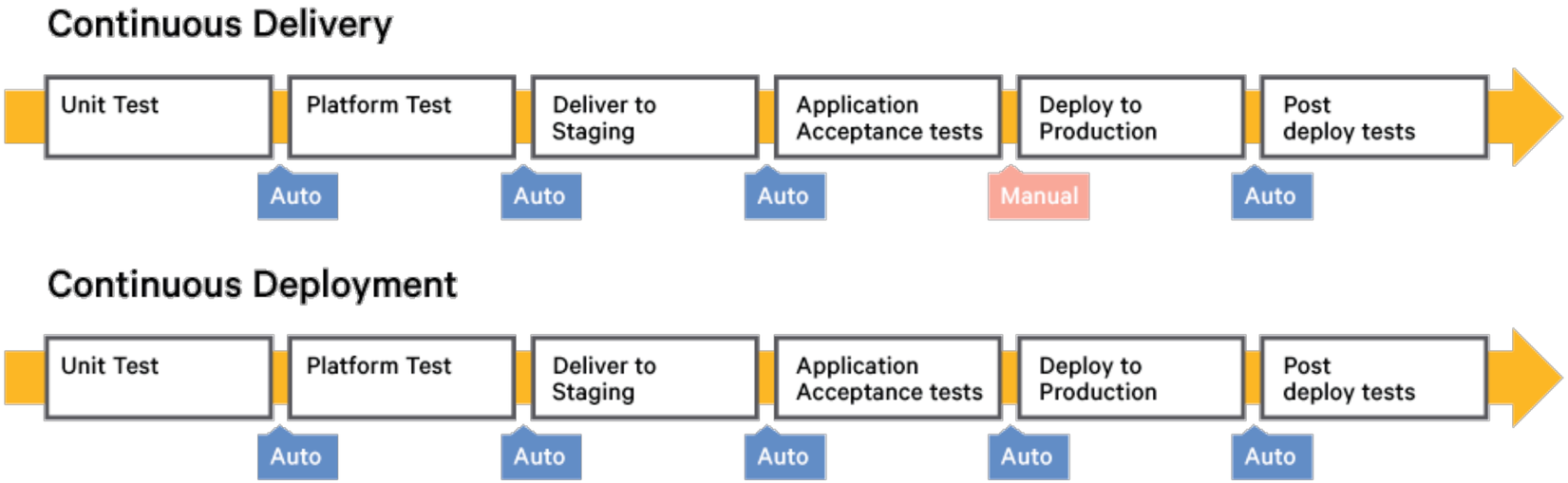
- Test Driven Development
  - Founded as part of the “extreme programming (XP)” methodology (1999)
  - Part of the Agile methodology
  - Steps:
    Add a test; Run all tests and see if the new test fails; Write the code; Run tests; Refactor code; Repeat
- Behaviour Driven Development
  - Tests are complete examples
- Generic Tests and Test Generation
  (e.g. QuickCheck)

Deployment Methods for Internet Applications

- Software development is increasingly competitive
- Any mistake can be extremely expensive
- Pressure is on to deliver fast and change even faster
- Companies deploy software at an astonishing pace:
  - Amazon: “every 11.7 seconds”
  - Netflix: “thousands of times per day”
  - Facebook: “bi-weekly app updates”

Deployment Methods for Internet Applications

- Processes and Methods for software construction and software deployment (DevOps)
- Specification and development methods
- Testing tools and toolchains
- Validation and Verification techniques



Summary

Course Overview

Internet Applications Design and Implementation

We focus on the **principles and concepts on the development of Internet applications**.

The syllabus follows an approach based on the fundamentals of software development based on **web and service oriented architectural patterns**, **advanced modularity mechanisms**, **data persistency abstractions**, **good development practices**, **performance concerns**, and **validation techniques**.

Lectures run along **practical assignments** and the **development of a running project** using frameworks, languages, and programming tools for Internet Applications that ensure the safety and compliance of the solution with relation to a specification

Goals: To Know

- Essential aspects of **architectural patterns** for inversion of control and software architectures specific for Internet Applications.
- **Principles of the development** of web applications and single page web applications.
- Mechanisms of **specifying and implementing web services** and web service orchestrations.
- **Internal structure** of an Internet **browser** and its **client applications**.
- Principles of **data-centric** and **user-centric development** in the context of Internet applications.
- Main **data abstraction** mechanisms used in Internet applications.
- Major performance **pitfalls** of Internet applications and their workarounds.
- Main specification and implementation mechanisms for **security policies in Internet Applications**.

Goals: To Do

- **Use development frameworks** that implement architectural styles for Internet applications.
- **Specify and build** web and cloud **applications** to support thin, flat, and native clients.
- **Specify and build** client **applications with reactive and rich behaviour**.
- **Implement authentication mechanisms** and **specify** the core **security rules** of an Internet Application
- Specify and efficiently use abstraction data layers such as **Object Relational Mappings** in Internet applications.
- **Design and deploy** Internet Applications that are efficient and maintainable.

Team

João Costa Seco

Eduardo Geraldo

Logistics (Plan)

- Weekly pre-recorded YouTube videos (approximately 1h per week)
- Interactive discussions (Tue. 15h-16h) — must see videos first
- Running project developed along the semestre (partial deliveries)
- Online Labs (Introductory Video + Q&A on Discord/Zoom)
- One-on-one Online Hours (by appointment)
- Resources
  - Youtube (lecture and demo videos)
  - Bitbucket (slides, assignments, started code, submission system, pipelines)
  - Discord (direct contact, conversation, questions)
  - Piazza (structured answers, announcements) — Check Piazza for all the links needed.

Evaluation

- Written evaluation component (50%) — 2 Tests or Exam
- Laboratory work component (50%)
  - TDD development of a sample Internet Application (with pipelines)
  - mandatory deliverables: (Data Model & API, Server, IFML Spec, Final)
  - a deployed bitbucket project with tags for all delivered versions (tags will be provided)

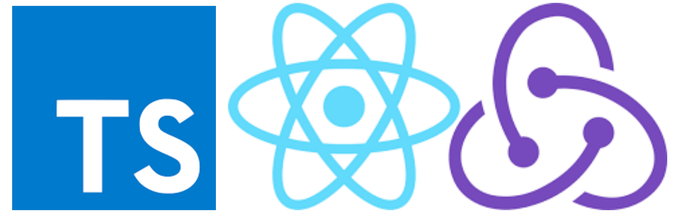
    If you don’t know how to operate with git, go learn TODAY!
  - work is evaluated for functionality, performance, method, etc.
  - teams of **3** members (named commits maybe used to distinguish between members)
  - deliveries include a written report, the final one includes a presentation with demo (20m)

Lecture Plan

Week	Lecture	Lab	Project
1	Overview and Logistics	Presentation of project, data modeling	
2	Software Archirecture. Service based architectures and their specification. Specification of RESTful APIS and beyond.	Implementing a webservice with a REST controller	
3	Frameworks for web and service-based applications	Designing a RESTful API using OpenAPI	RESTful API specification (9 Oct)
4	Data Abstraction. Data Access Patterns.	Implementing a Layered Architecture (Part 1)	
5	Data Abstraction in Spring (SpringData)	Implementing a Layered Architecture (Part 2)	
6	Security in Internet Applications	Implementing a Security Layer	
7	Test Driven Development	Q&A about project development	Server Component (30 Oct)
8	Guest Lecture about micro-service architectures		
9	Client Applications	Design of a pure HTML/JS client	
10	Client Frameworks and Languages	Implementing a client application using OpenAPI	
11	Specification and Implementation of Interactive Systems - IFML	Interface design with IFML	IFML design (4 Dec)
12	Model transformations	Implementation of a UI	
13	State Management	Q&A about project development	
14	Guest Lecture about client frameworks	Q&A about project development	The complete system (22 Dec)

Lab Exercises

- Tech Stack (mandatory):
  - Server-side: SpringBoot + Kotlin + Swagger + JUnit5 + MySQL
  - Client-side: TypeScript + React + Redux
- Tools stack
  - git (bitbucket — mandatory), with pipelines (optional)
  - IntelliJ
  - httpie/curl/postman



Project Deliveries

- A single project from the beginning:
  - Data-driven development of service based application, specification of data model and API
  - User-driven development of a progressive/single page client application, specification of user interaction
- A single bitbucket (mandatory) repository with client and server code, documentation, presentation and report
- Reports will include UserStories, UML/ER, IFML, OpenAPI spec, security spec
- Unit and Integration Test suite (including security aspects)
- Evaluation will cover both documentation, funcional correctness, code construction, coverage, security and performance issues.

Project assignment 2021

Grant application system

Project assignment 2021 — Grant application system

The project that you will implement this semester is a **management system for student grant applications**.

You will implement the process of submitting and evaluating **grant applications**, managed by the system and interfacing with several participants.

Project assignment 2021 — Grant application system

The system's behaviour starts by the **creation of grant calls**, and declaration of which **data items** should be submitted in each application. A grant application to a grant call is submitted by a **student of an institution,** giving its details. The needed items include personal info and curriculum vitæ items.

Reviewers, coming from institutions, are part of one or more **evaluation panels**, commanded by a **panel chair**. Reviewers submit **evaluations** to grant proposals once the submission deadline is passed and they are open for evaluation.

(more details on the system and the process as we go along…)

Bibliography

spring

TypeScript

learn play download interact

tutorial handbook samples language spec

Kotlin

Bibliography

- Marco Brambilla and Piero Fraternali. Interaction Flow Modeling Language – Model-Driven UI Engineering of Web and Mobile Apps with IFML. Morgan Kaufmann. 2014
- Martin L. Abbott and Michael T. Fischer, The Art of Scalability: Scalable Web Architecture, Processes and Organizations for the Modern Enterprise, Addison-Wesley Professional. 2009
- Martin Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley. 2002
- Software Architecture in Practice (3rd ed). Len Bass, Paul Clements, and Rick Kazman. Addison-Wesley 2015.
- Craig Walls. Spring in Action (4th edition). Manning. 2015

spring.io facebook.github.io/react/ typescriptlang.org kotlinlang.org


applications
web
service
mobile
techniques
development
Internet
services
mechanisms
orchestrate
patterns
data
modularity
clients
MVC
specific
advanced
tools
single
providers
page
pages
oriented
architectural
resources
software
Clients
validation
native
abstractions
methods
compound
Components
collaborative
online
reactive
result
practices
persistency
performance
challenges
concerns
good
Applications
fundamentals
integration
Model
Spring
REST
Service
Hibernate
TDD
Web
Beans
Aspects
JPA
Search
Relational
Transactions
Processes
Persistency
control
View
cycle
model
term
crawling
viewstate
Multi-language
cookies
Ajax
User
Concurrency
Security
engines
state
Access
Reactivity
Representational
Object
WS-BPEL
NoSQL
Database
Sessions
experience
analysis
Control
CMS
Life
Parameters
transfer
Short

Temas de Dissertação de Mestrado

- Contexto Académico
  - **Advances of SNITCH in Spring**. Desenvolvimento de ferramentas de controlo de segurança para Java e Java Spring (continuação de 2 teses de mestrado) — (Equipa: João Costa Seco, Eduardo Geraldo)
  - **Linguagem para processos de negócio** implementação de um ambiente de programação para uma linguagem baseada em grafos de processos. Colaboração com a universidade de Copenhaga (continuação de 1 tese de mestrado).
- Contexto Académico-Empresarial (OutSystems)
  - diversos temas relacionados com o desenvolvimento de ferramentas de programação (abordagens baseadas em linguagens de programação: ICL, CVS). Temas sobre transformação de modelos e desenvolvimento de User-Interfaces.
- Contexto Académico-Empresarial (VORTEX/Altran)
  - diversos temas relacionados com o desenvolvimento de ferramentas de verificação de código de sistemas de tempo real.

Internet Applications Design and Implementation

2020 - 2021

(Lab class 1)

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)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Just start to model the database

- Use lucidchart to do a ER that covers the assignment cases
- Discuss it with the instructor, submission due in 2 weeks together with a RESTful API (starts next week).
- Establish a team of 3 students, create a bitbucket repository and share it with the instructors (joao.seco@fct.unl.pt and e.geraldo@campus.fct.unl.pt)
- Commit a pdf and tag it (DataModel)…

Project assignment 2021 — Grant application system

The project that you will implement this semester is a **management system for student grant applications**.

You will implement the process of submitting and evaluating **grant applications**, managed by the system and interfacing with several participants.

Project assignment 2021 — Grant application system

The system's behaviour starts by the **creation of grant calls**, and declaration of which **data items** should be submitted in each application. A grant application to a grant call is submitted by a **student of an institution,** giving its details. The needed items include personal info and curriculum vitæ items.

Reviewers, coming from institutions, are part of one or more **evaluation panels**, commanded by a **panel chair**. Reviewers submit **evaluations** to grant proposals once the submission deadline is passed and they are open for evaluation.

(more details on the system and the process as we go along…)

Internet Applications Design and Implementation 2020 - 2021

(Lecture 2 - Software 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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outline

- Software Architecture - Introduction
- Software Architecture for Internet Applications
  - Three-tier architecture
  - Service-based architectures
  - Microservice-based architectures
- Frameworks at the service of Software Architecture
- The architectural style REST to instantiate webservices
- Specifying webservices with OpenAPI and Spring

Internet Applications Design and Implementation 2020 - 2021

(Lecture 2, Part 1 - Software Architecture - Introduction)

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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Software Architecture

Introduction

based on the book “Software Architectures in Practice”

Software Architecture

- What is software architecture?
- What are the benefits of using one?

Software Architecture

- What is software architecture?

  “The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.” (in Software Architectures in Practice)

- Structures can be:
  - the module decomposition structure, which divide computational responsibilities and work assignment (among teams). Identify modules or components
  - runtime structures (connectors) that deal with the communication between components (eg services)
  - organisational structures. How components are developed, tested, deployed

Software Architecture…

… is a form of Abstraction
(different views over the same system)

… is in every software system

(from caos to tidy)

… includes Behaviour
(names and connectors have semantics)

Not All Architectures Are Good Architectures

Software Architecture (Structures)

- Module structures
  - What is the primary functional responsibility assigned to each module?
  - What are the dependencies to other software elements?
  - What modules are each module related to? by generalisation or specialisation.
- Component and connector structures
  - How do modules interact?
  - What are the shared data stores?
  - What is the data flow in the system?
  - Can the structure change? how?
  - What are the security requirements? performance bottlenecks?
- Allocation structures
  - What is the hardware/cloud infrastructure? module ownership? which are the regressions tests? etc.

Architectural Patterns

- Pre-determined compositions of architectural elements provide strategies for solving common problems in software systems.
- Examples:
  - Layered pattern — Linear (unidirectional) dependencies between multiple elements
  - Shared-data pattern — Components and connectors to create and manipulate persistent data, connectors are languages like SQL.
  - Client-server pattern — Components: Clients and servers; Connectors: protocols and languages
  - Multi-tier pattern — A generic deployment structure of components in different infrastructures
  - Competence center — Work assignment division by expertise (eg. departments)

Internet Applications Design and Implementation 2020 - 2021

(Lecture 2, Part 2 - Software Architecture - Internet Applications)

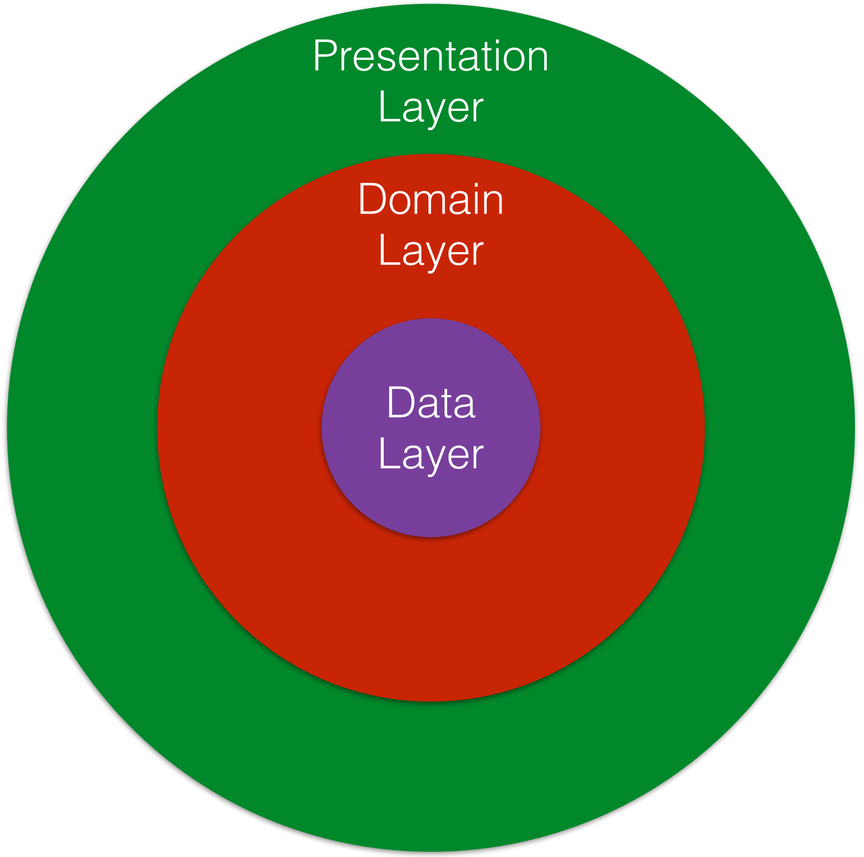
MIEI - Integrated Master in Computer Science and Informatics
Specialization block

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))

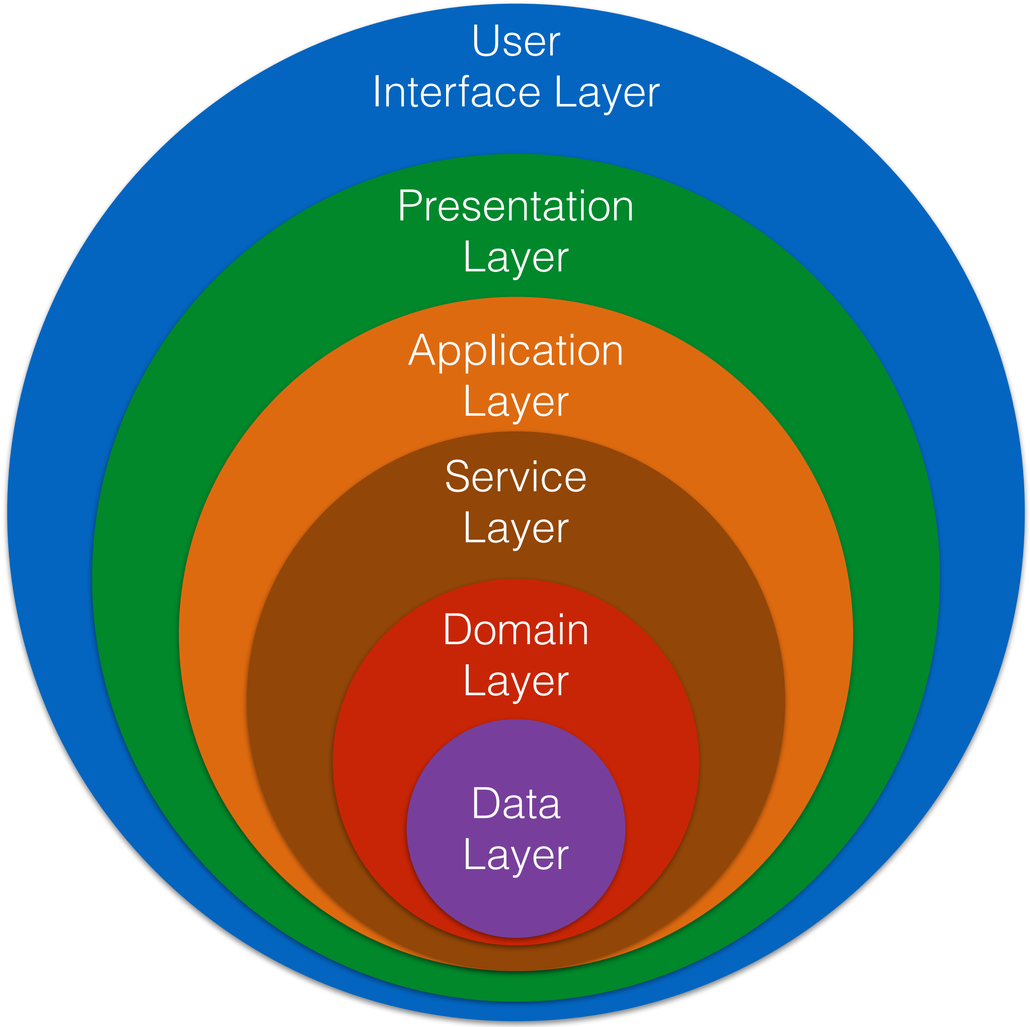
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Internet Applications are Data-Centric



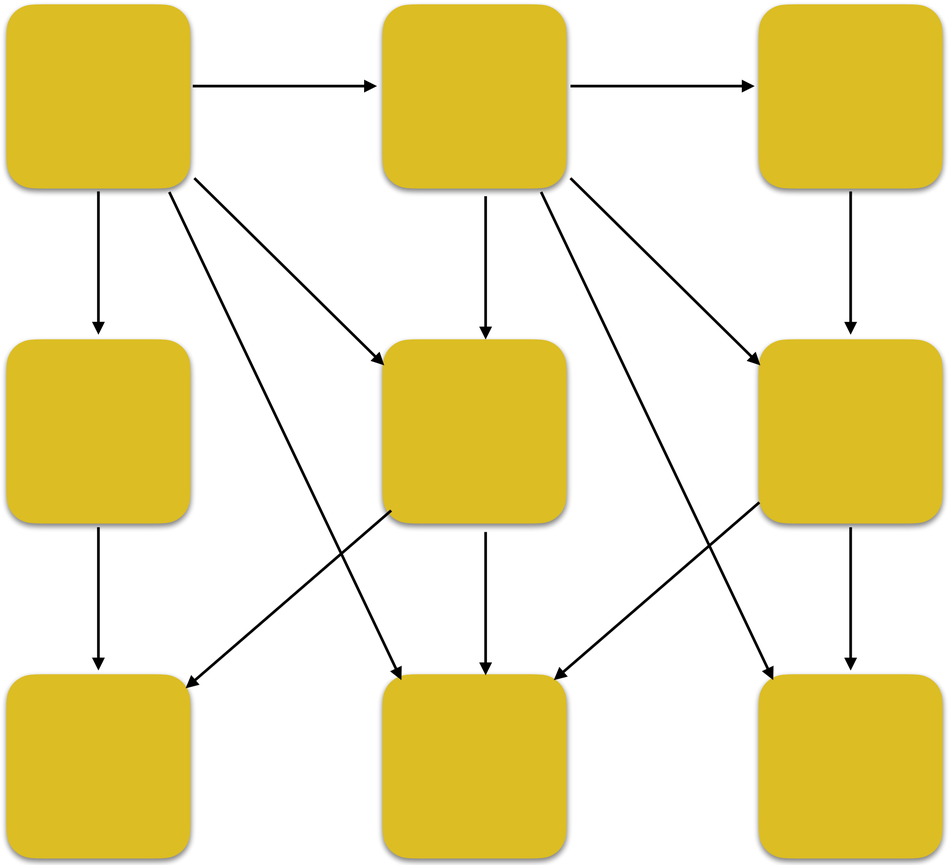
Patterns of Enterprise Application Architecture

Internet Applications are Data-Centric



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

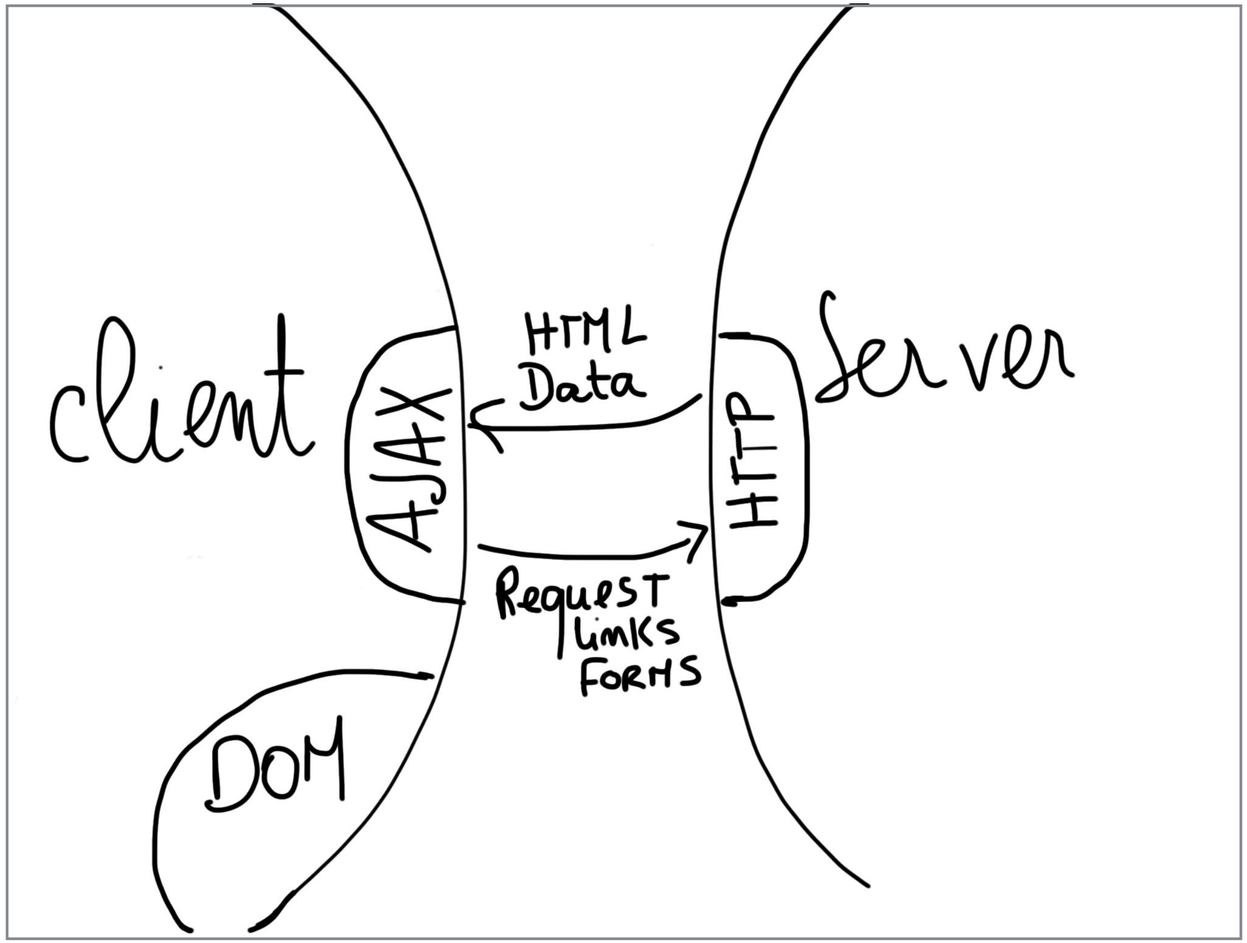
Internet Applications are also Decentralised



https://dzone.com/articles/introduction-to-microservices-part-1

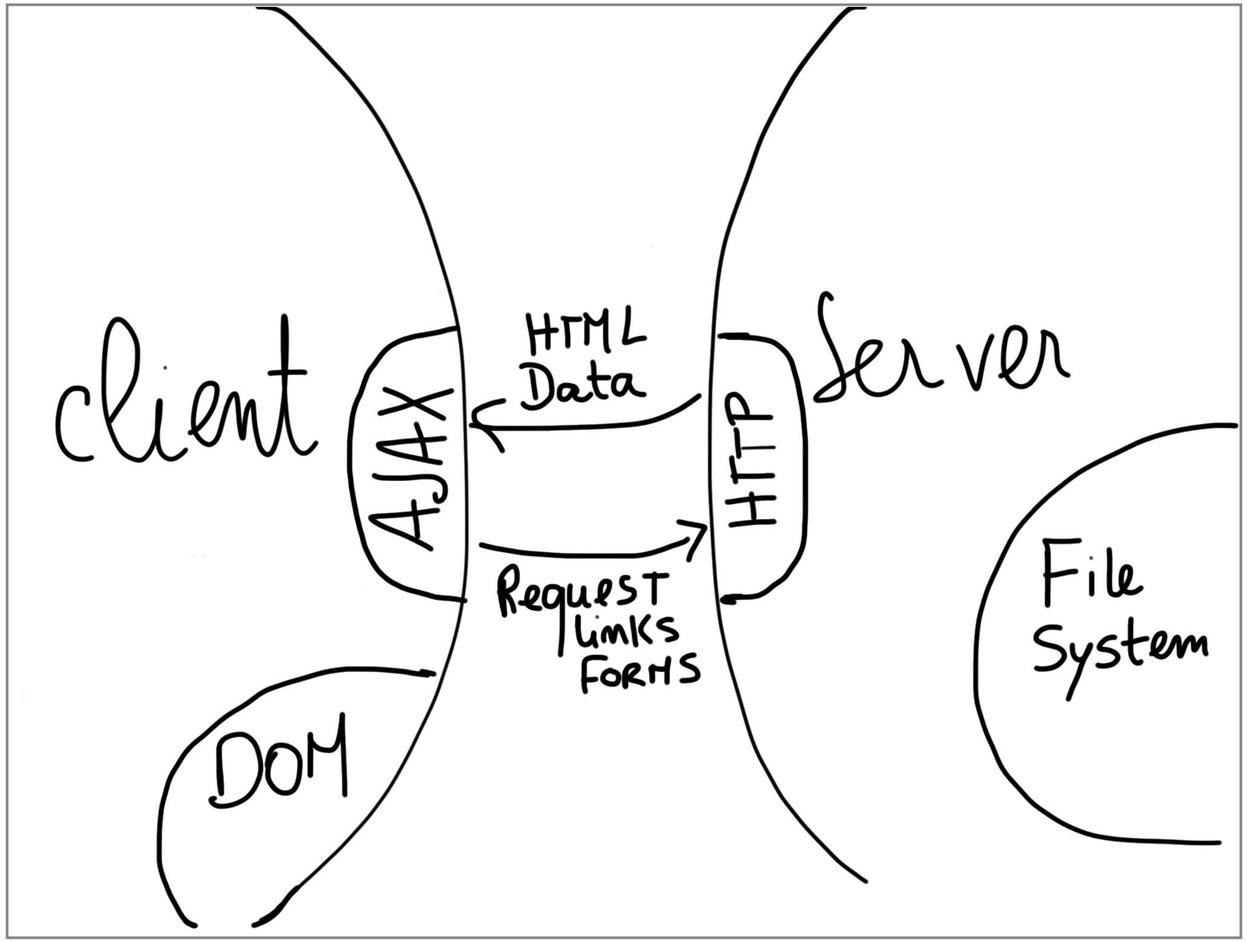
(Logical) Architecture of Web applications

- Tech Details: Interconnection based on the HTTP protocol
  - Method, path, arguments, (a)synchrony, multi-typed response



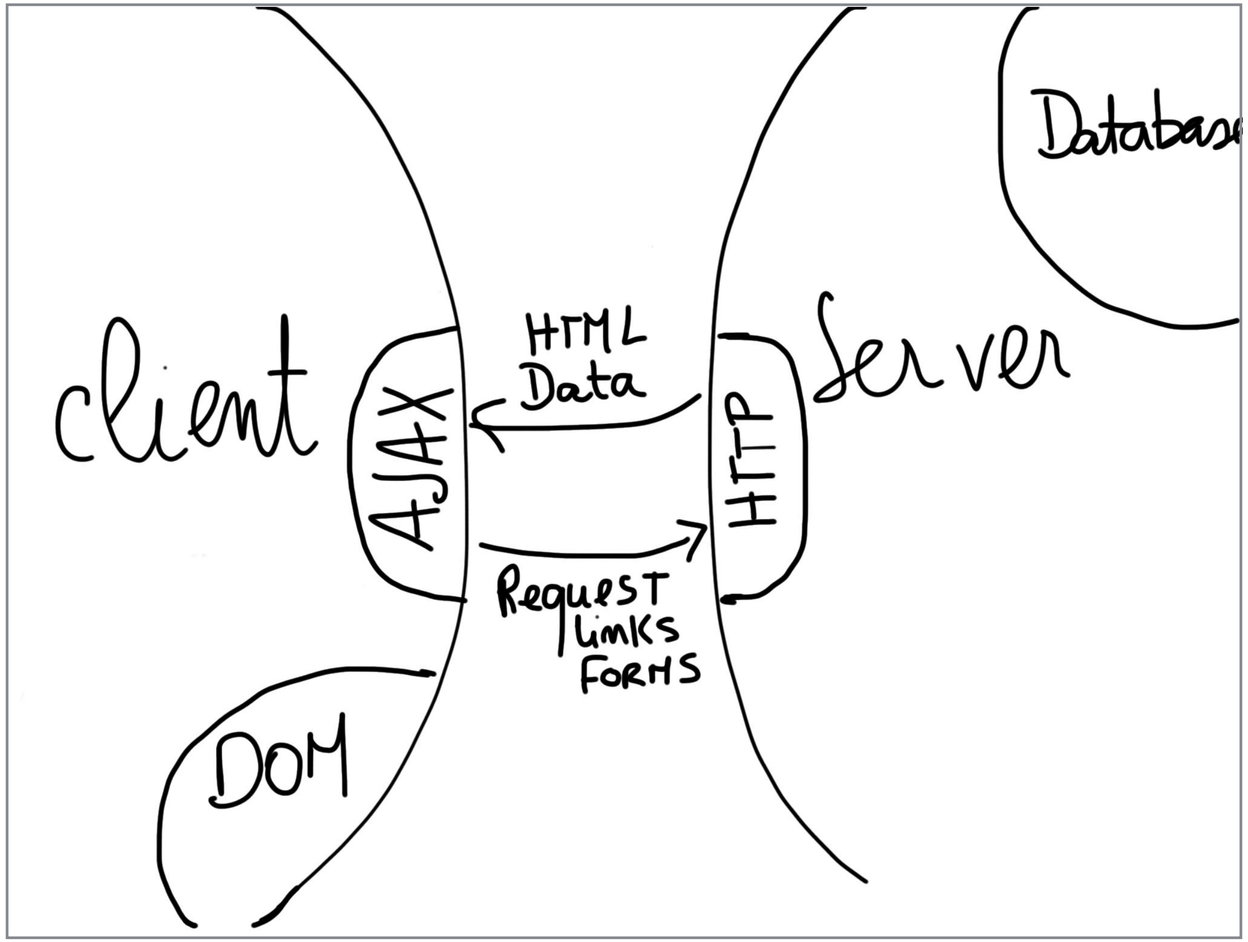
Architecture of Web applications

- Static HTML pages stored in the file system



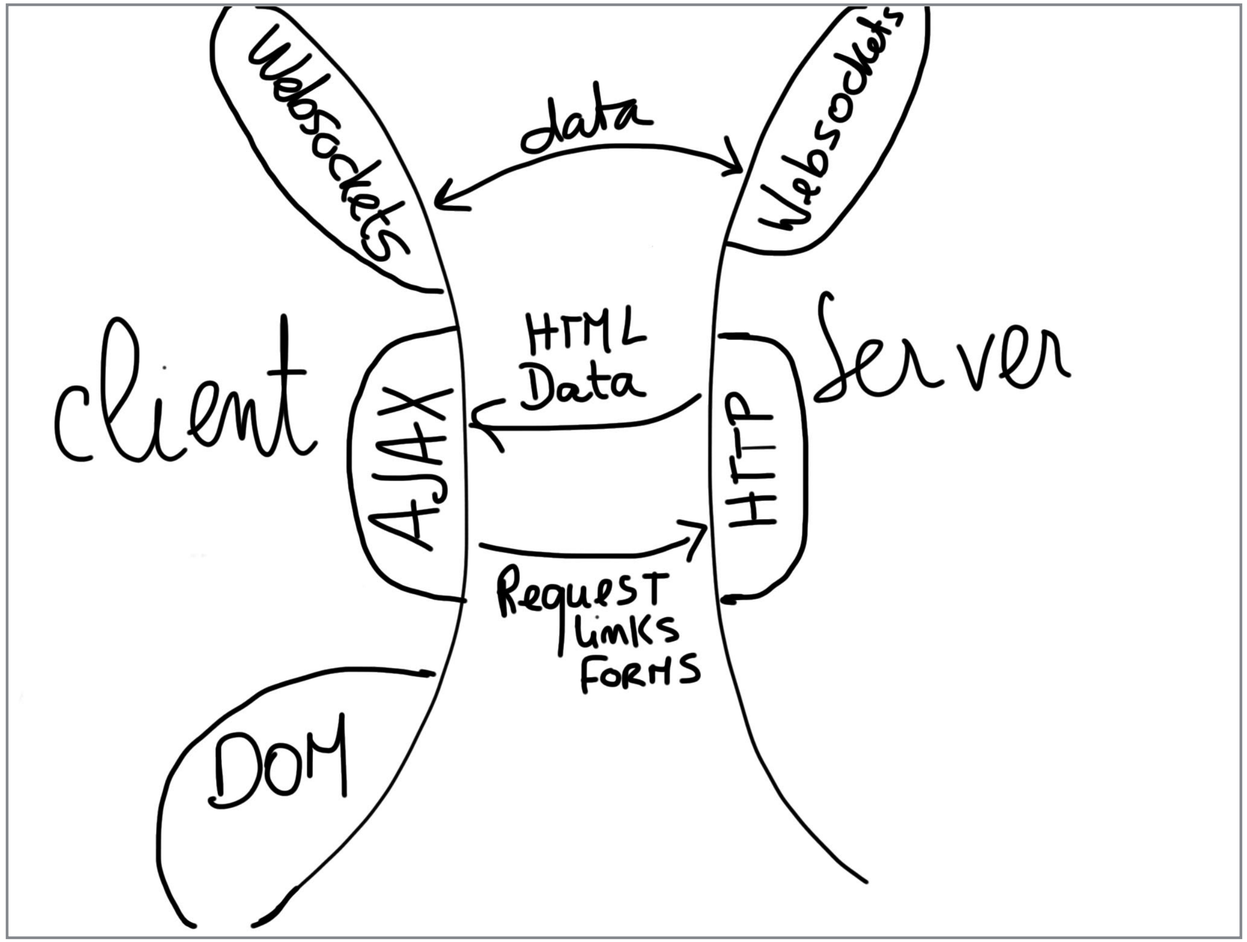
Architecture of Web applications

- Dynamic HTML pages based on data stored in some database.

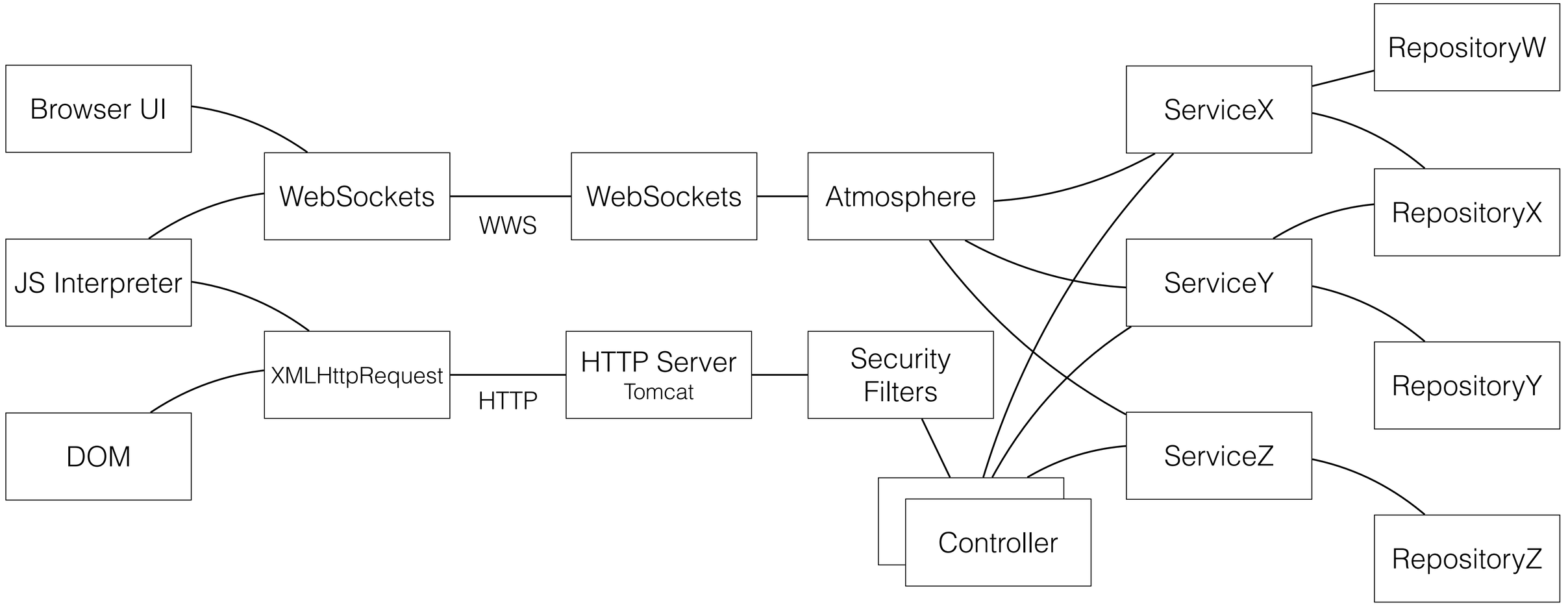


Architecture of Web applications

- Dynamic HTML pages with permanent connection with the server to exchange data.

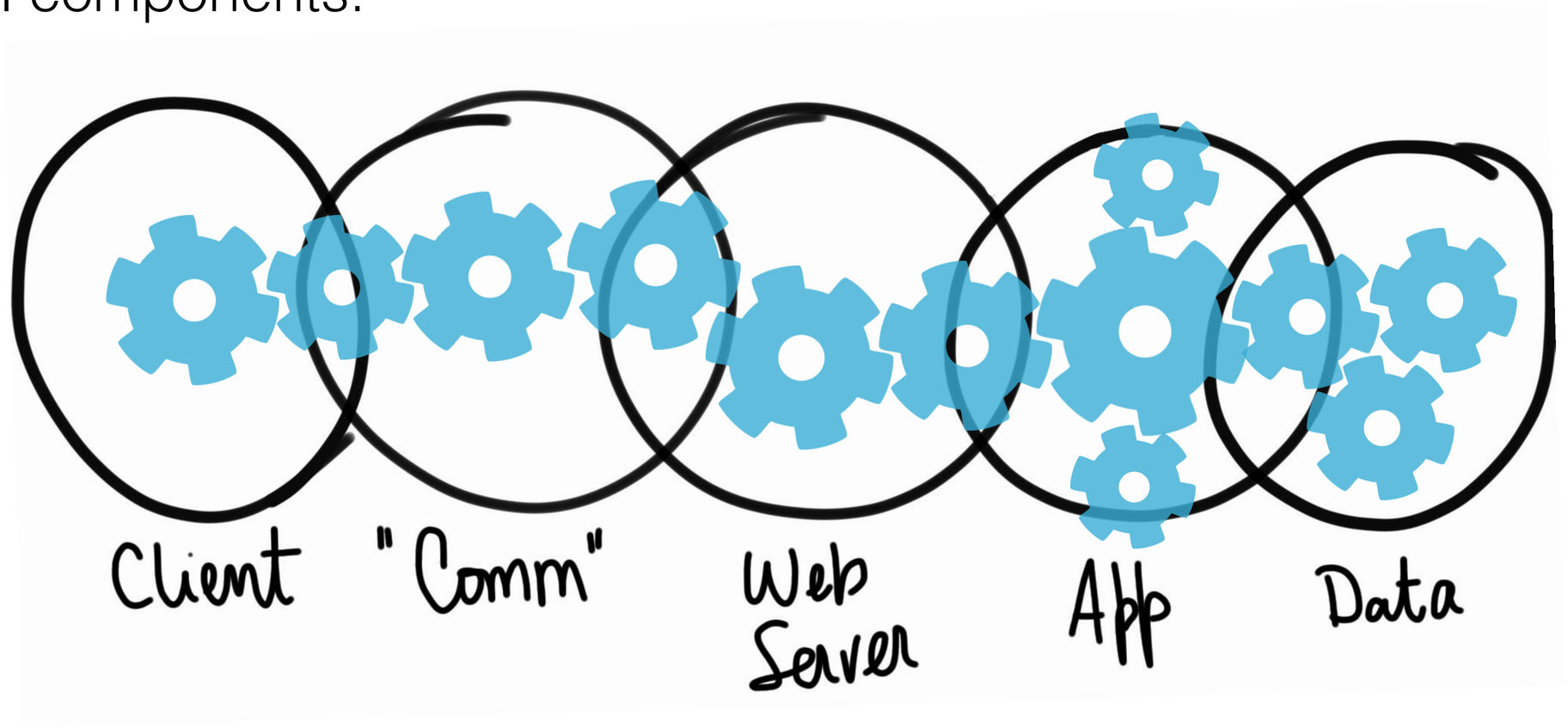


Architecture of Web applications



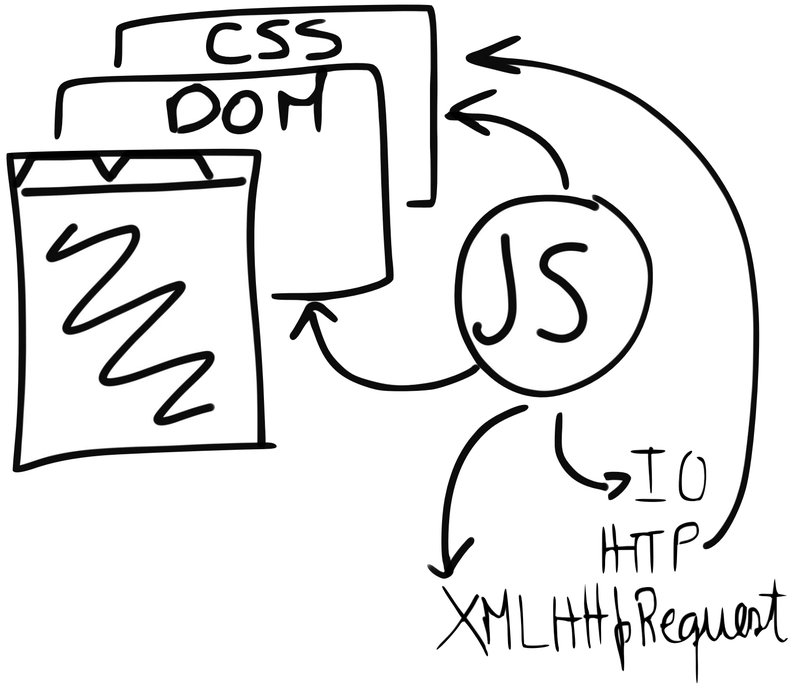
Architecture of Web applications - The Big Picture

- A web application is made of code deployed to the independent and different physical components.



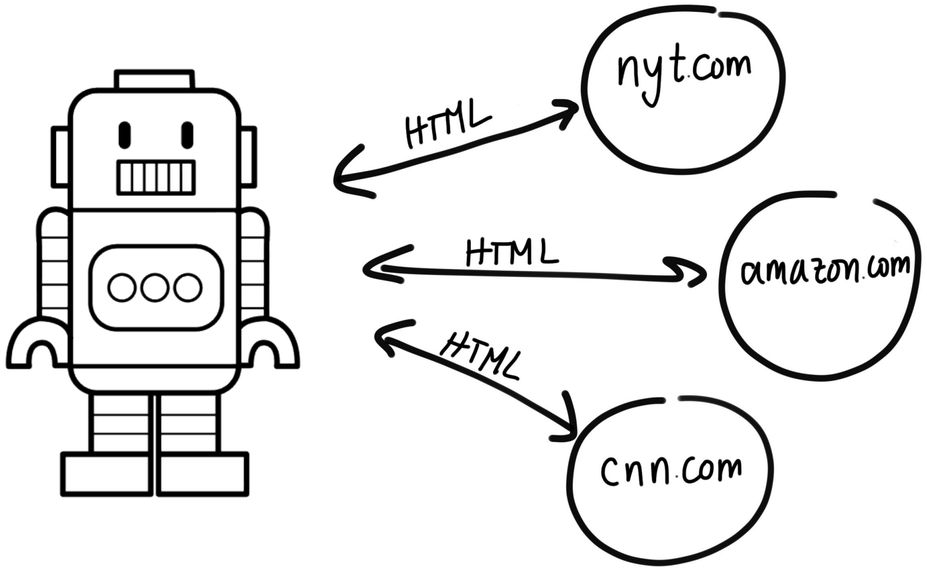
Web client architecture

- Browser (HTML5)
  - HTML (structure and semantics)
  - CSS (style and UI behaviour)
  - JS + AJAX,
    Socket interfaces (behaviour)
  - DOM
    (the supporting data structure)
  - UI Events & callbacks
    (mechanism for dynamic structuring of behaviour)



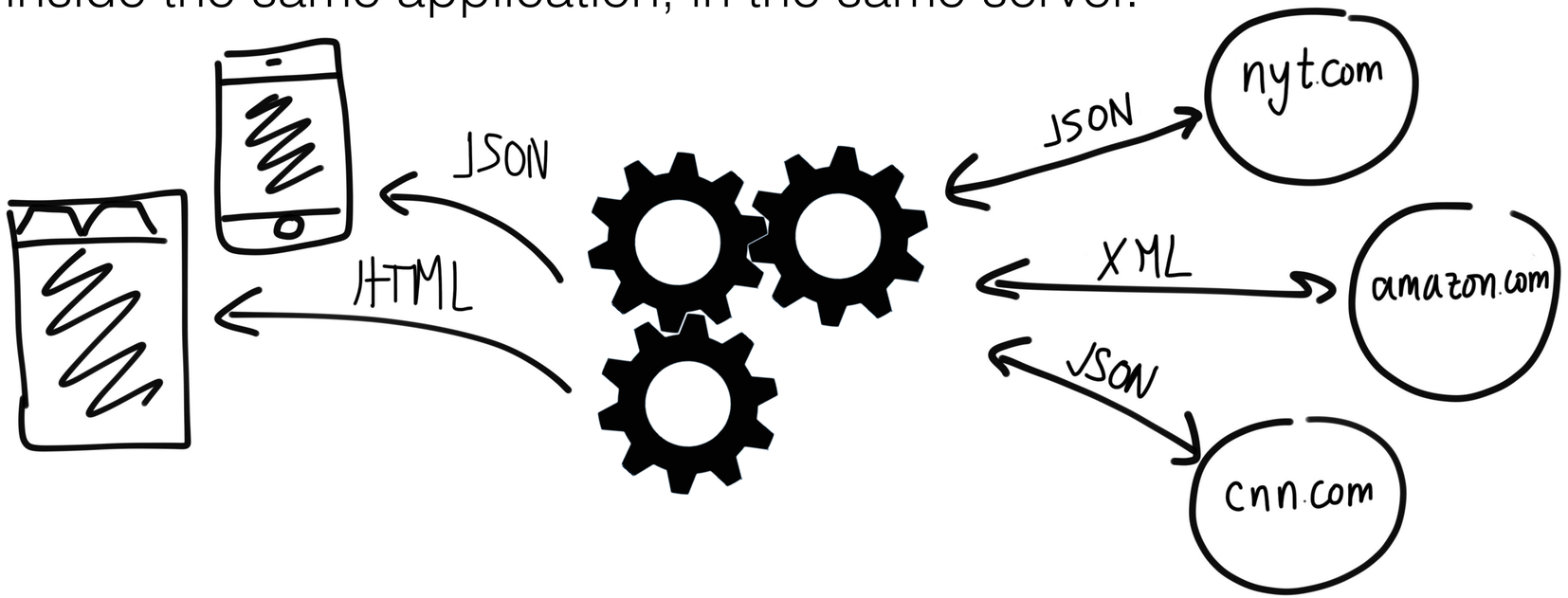
Web client architecture

- Other kind of web clients (asking for HTML)
  - simulate web requests and sessions with web-servers
  - parse/crawl the results to extract information
  - should be built with web-services instead (if possible).



Service based web client architecture

- Provide web content based on data providing services
  - Data can be exchanged in “machine-friendly” formats.
    (e.g. JSON, XML)
  - Combined and refurbished in HTML or data formats.
  - Even reused inside the same application, in the same server.



Interconnection layer (low-level support)

- HTTP/1.1 (*Hypertext Transfer Protocol*) Protocol
  - Method: GET, HEAD, POST, PUT, DELETE (3 more)
  - Arguments (query string and body)
  - Multi-typed message body
  - Cookies
  - Return codes: 1XX, 2XX, 3XX, 4XX, 5XX
- Websockets (standard RFC6455 2011 - ws:// wss://)
  - supported by browsers and web servers to allow two way data communication between client and servers
- HTTP/2 approved May 2015.
  - Faster, compressed, and cyphered transmission of data
- TLS (successor of SSL)
    - Provides cryptographic support for web communications (handshake+symmetric crypto)

https://httpstatuses.com/

1×× Informational
100 Continue
101 Switching Protocols
102 Processing

2×× Success
200 OK
201 Created
202 Accepted
203 Non-authoritative Information
204 No Content
205 Reset Content
206 Partial Content
207 Multi-Status
208 Already Reported
226 IM Used

3×× Redirection
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
305 Use Proxy
307 Temporary Redirect
308 Permanent Redirect

4×× Client Error
400 Bad Request
401 Unauthorized
402 Payment Required
403 Forbidden
404 Not Found
405 Method Not Allowed
406 Not Acceptable
407 Proxy Authentication Required
408 Request Timeout
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Payload Too Large
414 Request-URI Too Long
415 Unsupported Media Type
416 Requested Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot
421 Misdirected Request
422 Unprocessable Entity
423 Locked
424 Failed Dependency
426 Upgrade Required
428 Precondition Required
429 Too Many Requests
431 Request Header Fields Too Large
444 Connection Closed Without Response
451 Unavailable For Legal Reasons
499 Client Closed Request

5×× Server Error
500 Internal Server Error
501 Not Implemented
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
505 HTTP Version Not Supported
506 Variant Also Negotiates
507 Insufficient Storage
508 Loop Detected
510 Not Extended
511 Network Authentication Required
599 Network Connect Timeout Error

Web server / App server architecture

- Web servers handle HTTP requests, map URLs to local files, execute local scripts (e.g. CGI, PHP), or locally bound code (e.g. Servlets).
- App Servers modularly manage bound code, and associated resources (e.g. sessions, context, connections).
  - One web server, many applications.
  - Allows the assembly of components of applications (controllers, views, models)

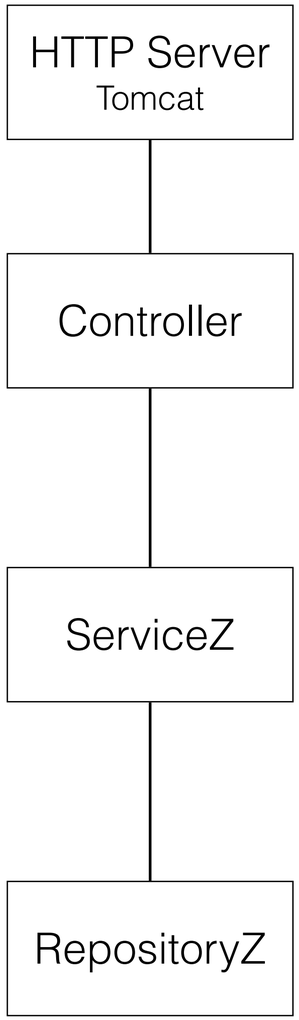
Web architectures, patterns and styles

- Increase the level of abstraction, reusability, and maintainability
  - Software Architectures
  - Architectural and design patterns
  - Architectural styles

- Software frameworks implement some, and complement with libraries and tools.
- User-defined pieces are required to specify the “core” logic, and configure general purpose code.
- All implement the “inversion of control” pattern.

An architecture built with Spring

Example of an Architecture built with Spring



- 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>
```

Service Based Architectures

Service Oriented Architectures

- Are the technological and methodological basis for building open-ended Internet Applications.
- Provide a model of distributed computation based on loosely coupled interactions.
- Define heterogeneous ecosystems of service implementations.
- Use implementation independent data formats (JSON, XML)
- Allows independent development of services by different vendors and technologies
- A service:
  - Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)
  - Is self-contained
  - May be composed of other services
  - Is a “black box” to consumers of the service

https://web.archive.org/web/20160819141303/http://opengroup.org/soa/source-book/soa/soa.htm

Web architectures, patterns, and styles

- Web services are usually defined over HTTP protocol

- **SOAP** (*Simple Object Access Protocol*)
  - Operation based protocol (on HTTP) to implement web services (XML as format)
- **REST** (*Representational State Transfer*)
  - Resource based architectural style to implement web services over HTTP (or another connection protocol)

Micro Service Architectures

- Are an extreme interpretation of service based architectures.
- Have smaller grained services and interfaces.
- Provide independent and lightweight deployment.
- Allow flexible management (e.g. replicas).
- Based on a clear service ownership model (team responsible for all stages of dev&ops).
- Isolated persistent state (sometimes bad).

https://martinfowler.com/microservices/

Amazon's API Mandate (by Jeff Bezos, 2002)

- All teams will henceforth expose their data and functionality through service interfaces.
- Teams must communicate with each other through these interfaces.
- There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
- It doesn't matter what technology you use.
- All service interfaces, without exception, must be designed from the ground up to be externalizeable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
- The mandate closed with: Anyone who doesn't do this will be fired. Thank you; have a nice day!

Internet Applications Design and Implementation 2020 - 2021

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

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

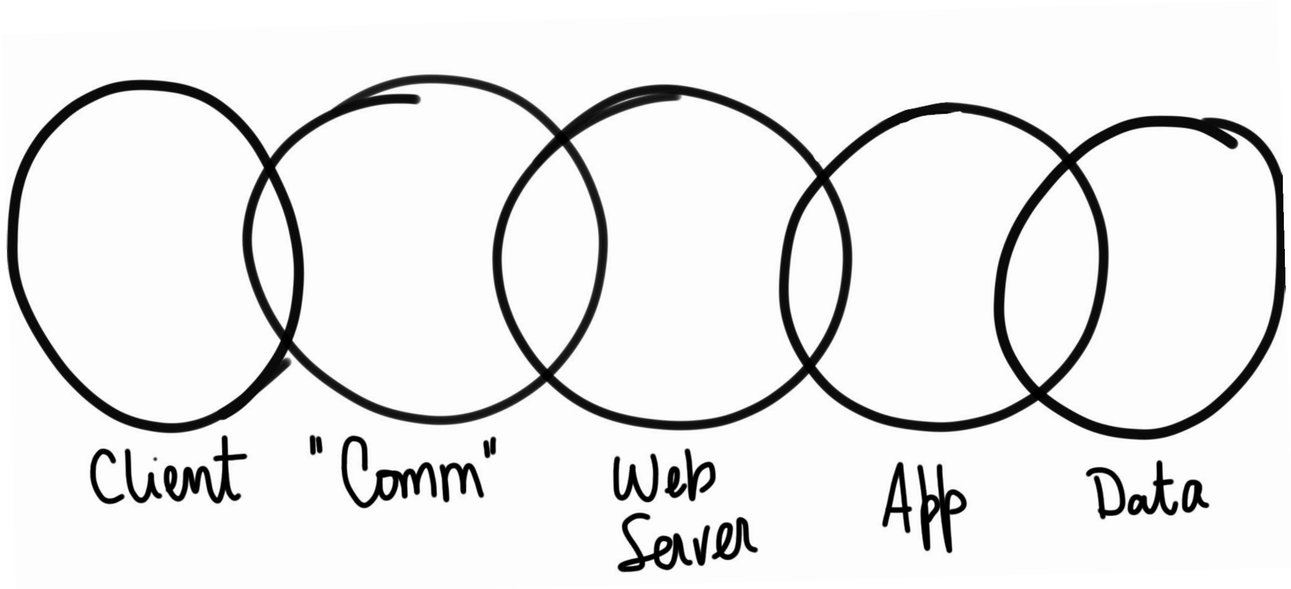
João Costa Seco (joao.seco@fct.unl.pt)

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

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

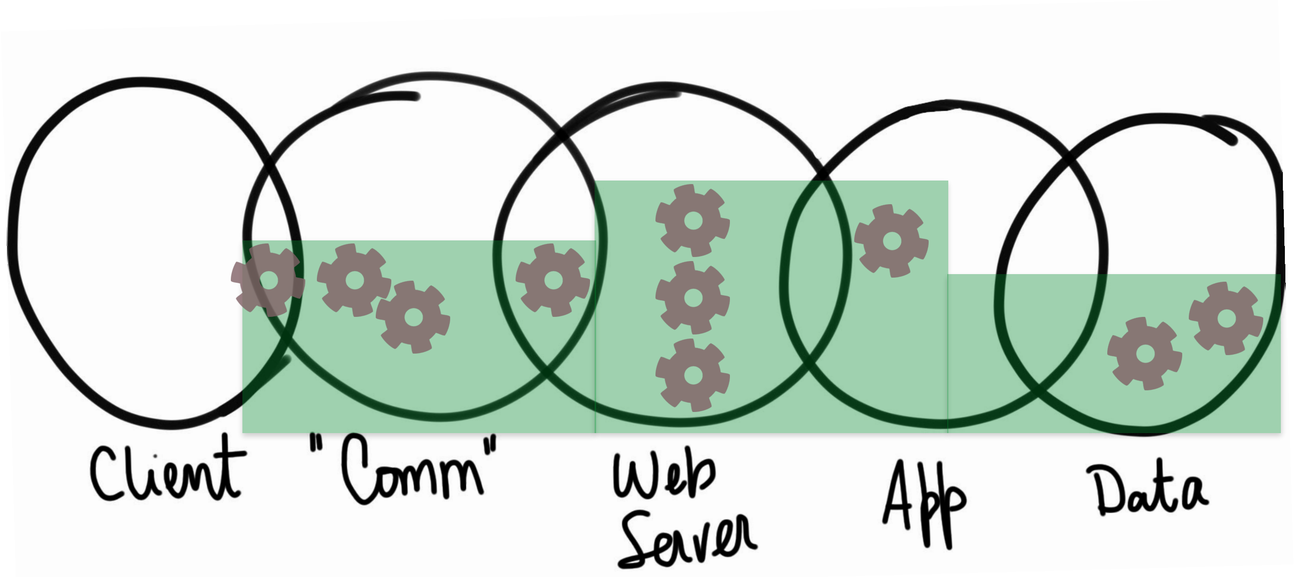
Web architectures, patterns and styles

- Most common web applications follow the 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)



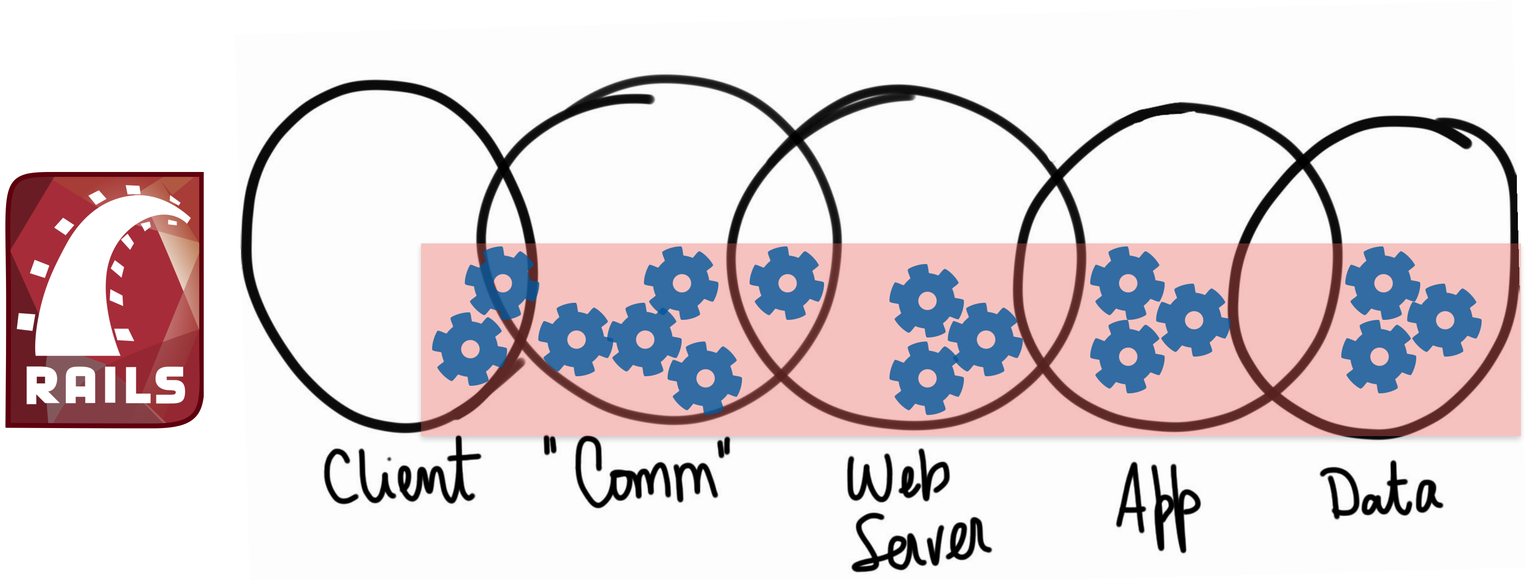
Web architectures, patterns and styles

- Most common web applications follow the 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)



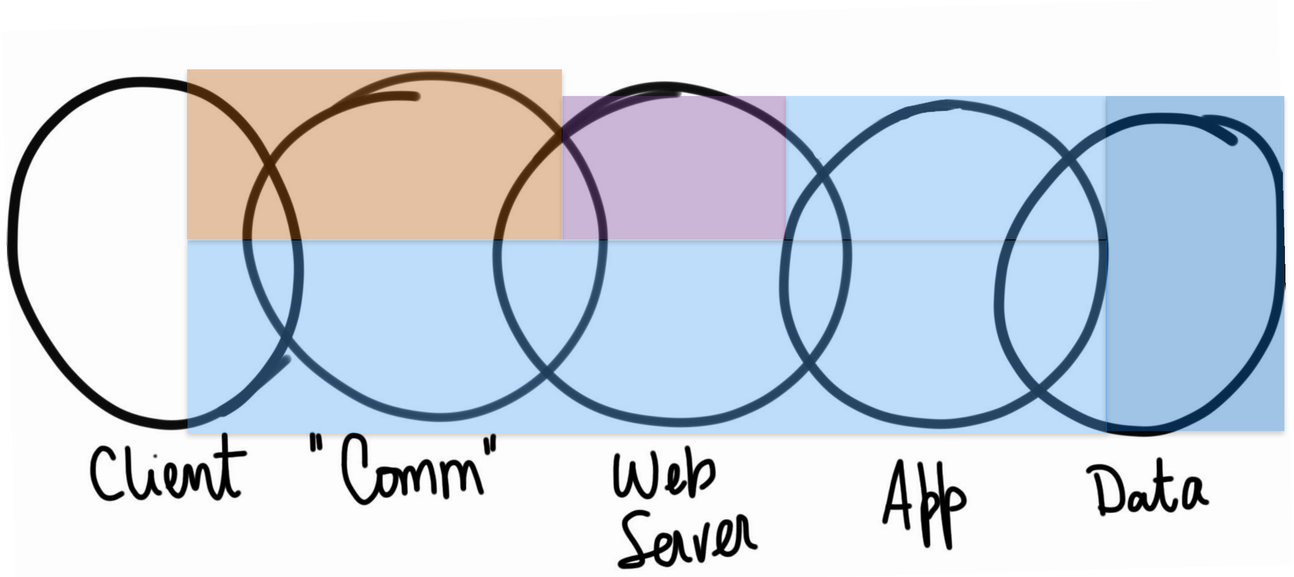
Web architectures, patterns and styles

- Most common web applications follow the 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)



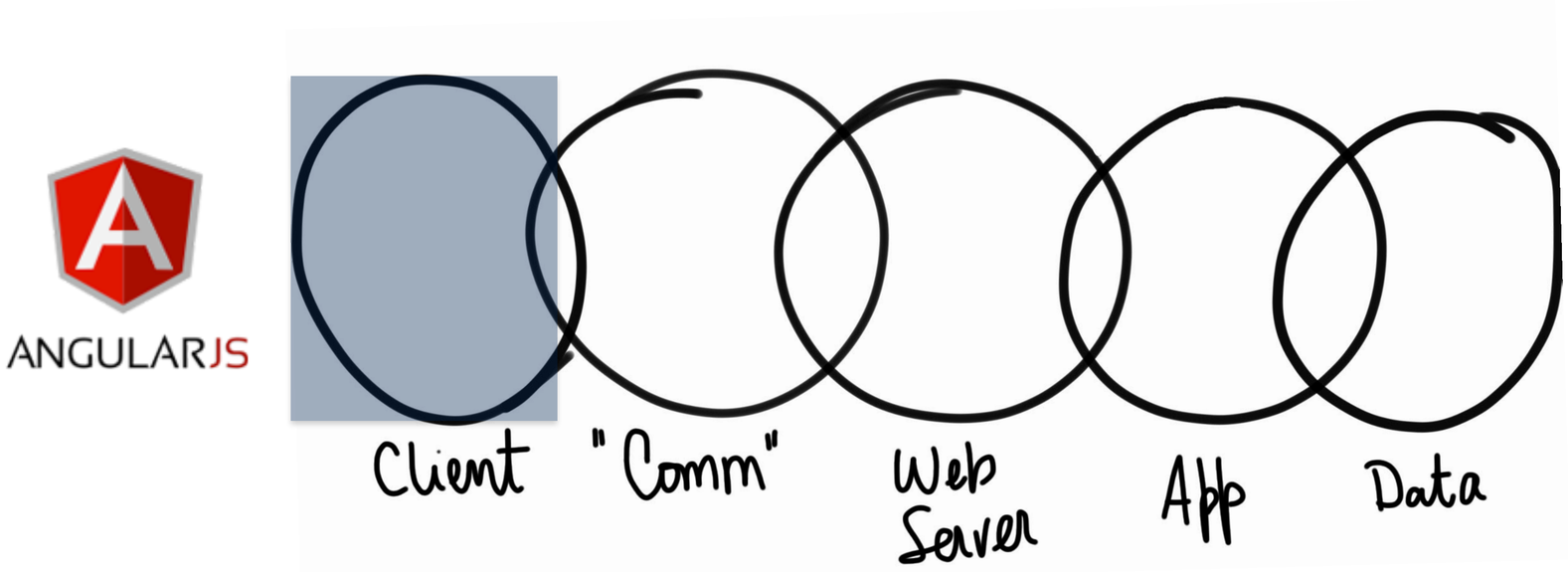
Web architectures, patterns and styles

- Most common web applications follow the 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)



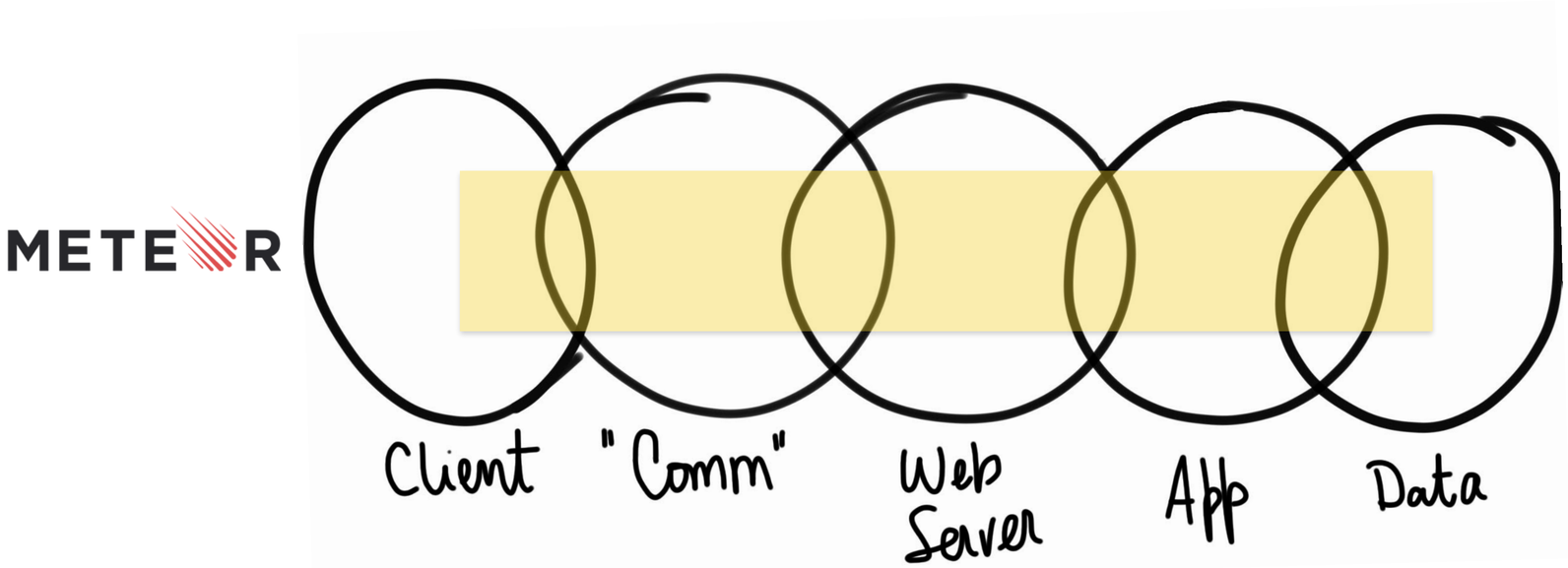
Web architectures, patterns and styles

- Most common web applications follow the 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)



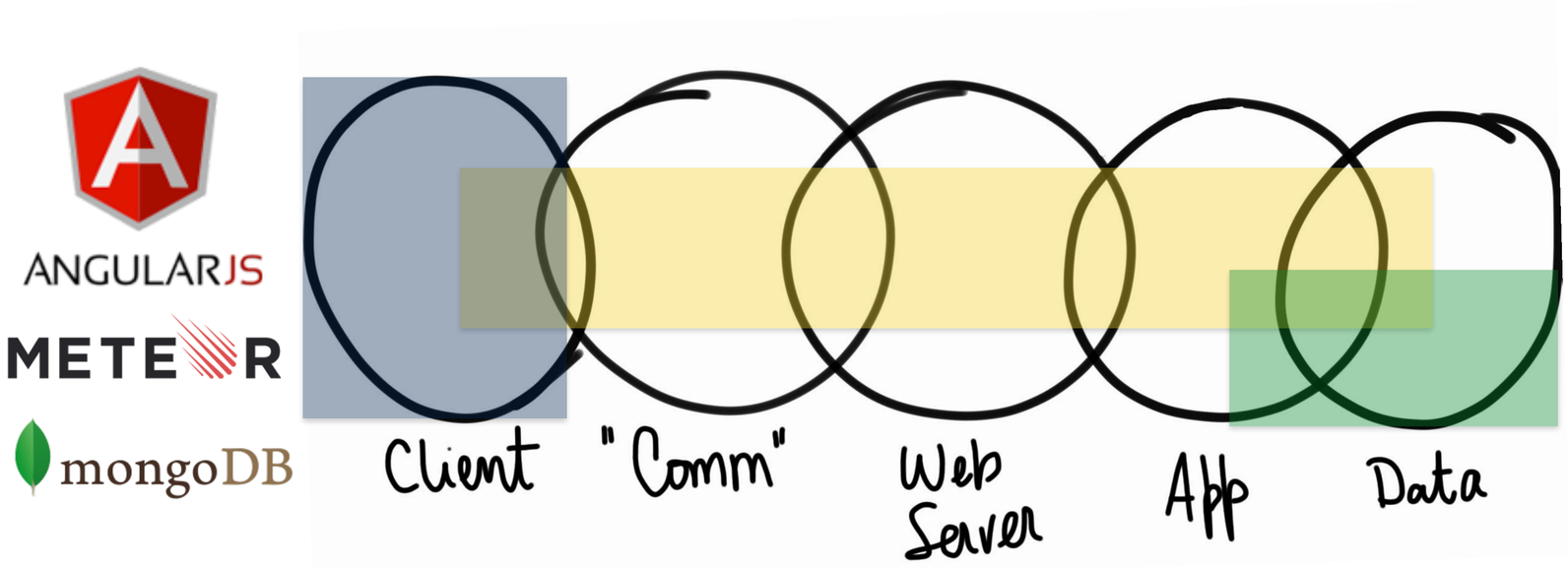
Web architectures, patterns and styles

- Most common web applications follow the 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)



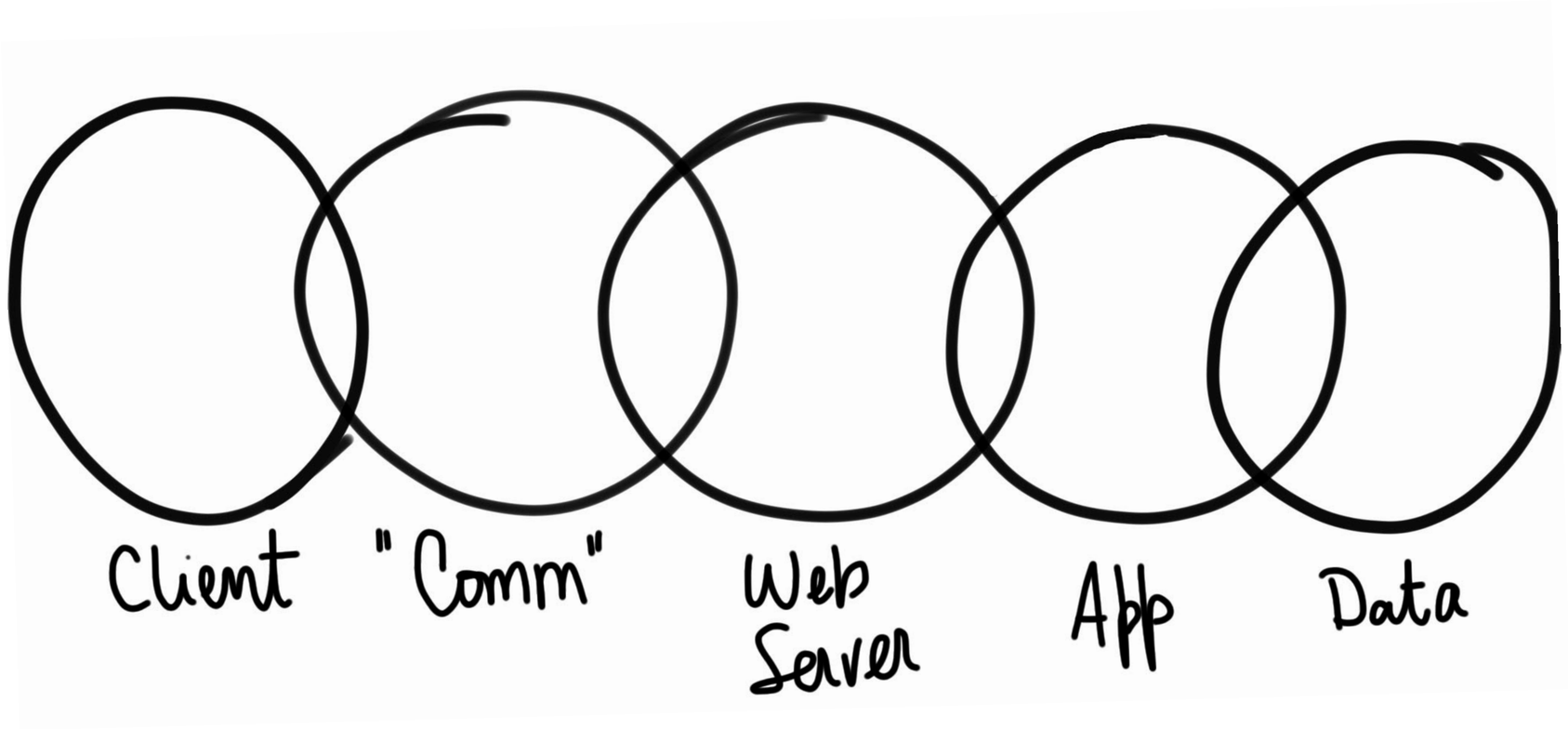
Web architectures, patterns and styles

- Most common web applications follow the 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)



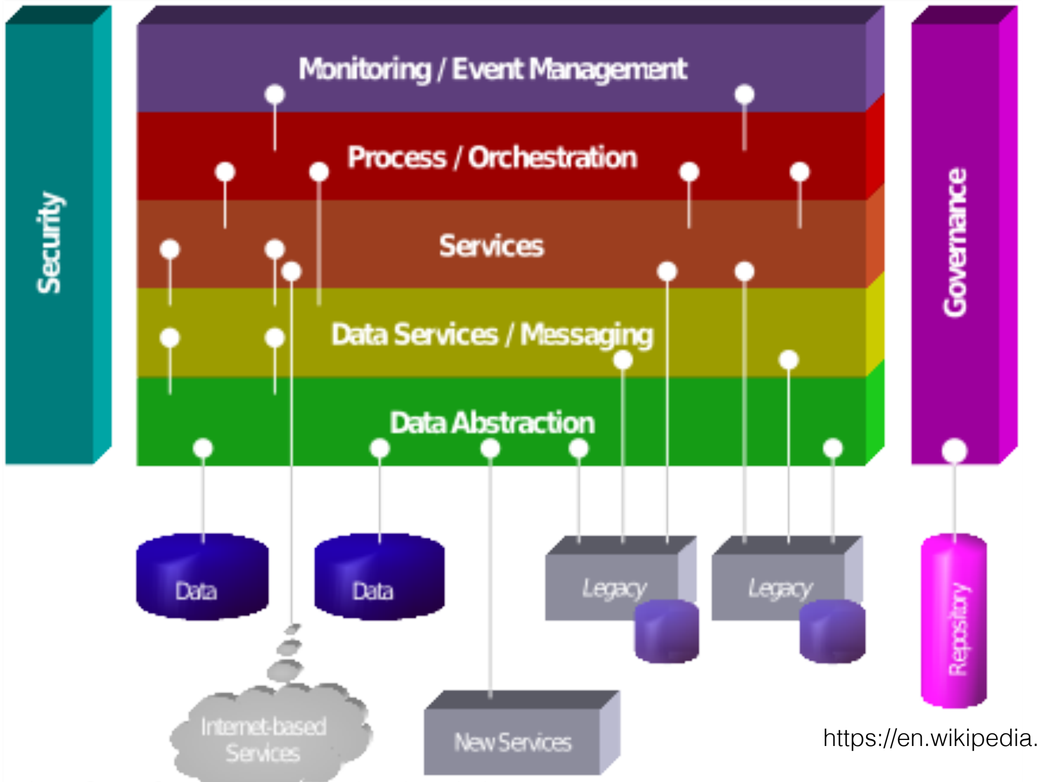
Summary - Web Frameworks

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



(web & local) Services

- Service oriented architectures are a way of decoupling implementation from use.
- Services are implemented based on a “contract” or interface and provided by a broker.



https://en.wikipedia.org/wiki/Service-oriented_architecture

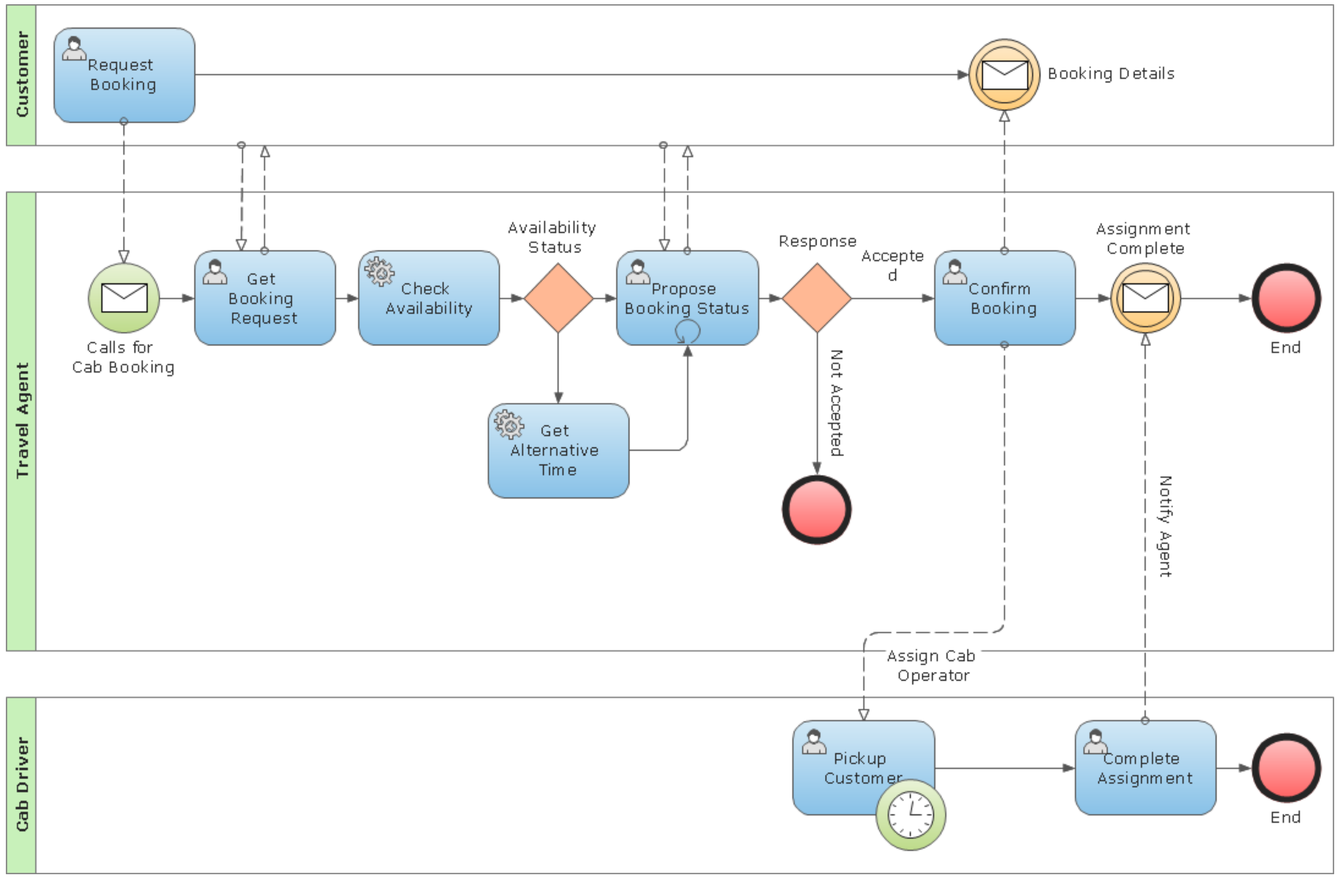
Service Orchestration Languages

- Spring Web Flow

```
<?xml version="1.0" encoding="UTF-8"?>
<flow xmlns="http://www.springframework.org/schema/webflow"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/webflow
                          http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd"


        <start-state="welcome">

        <view-state id="welcome" view="/welcome.jsp">
            <transition on="continue" to="finish"/>
            <transition on="cancel" to="cancelled"/>
```

Process Specification Languages

- From processes to code…



Internet Applications Design and Implementation 2020 - 2021

(Lecture 2, Part 4 - 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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

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/

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}

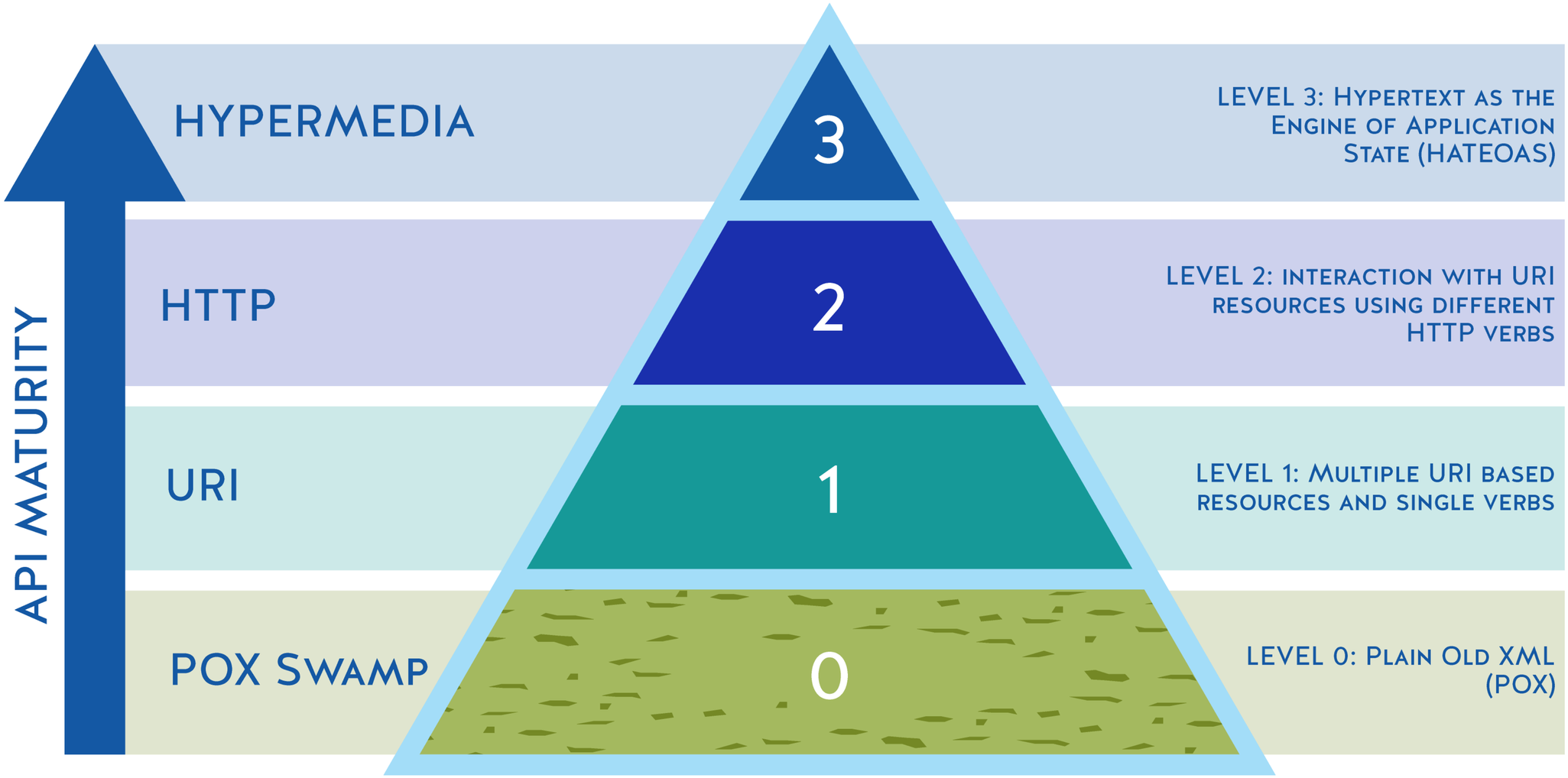
Partner contacts

- `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();
    }
}
```

Internet Applications Design and Implementation 2020 - 2021

(Lecture 2, Part 5 - 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))

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

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

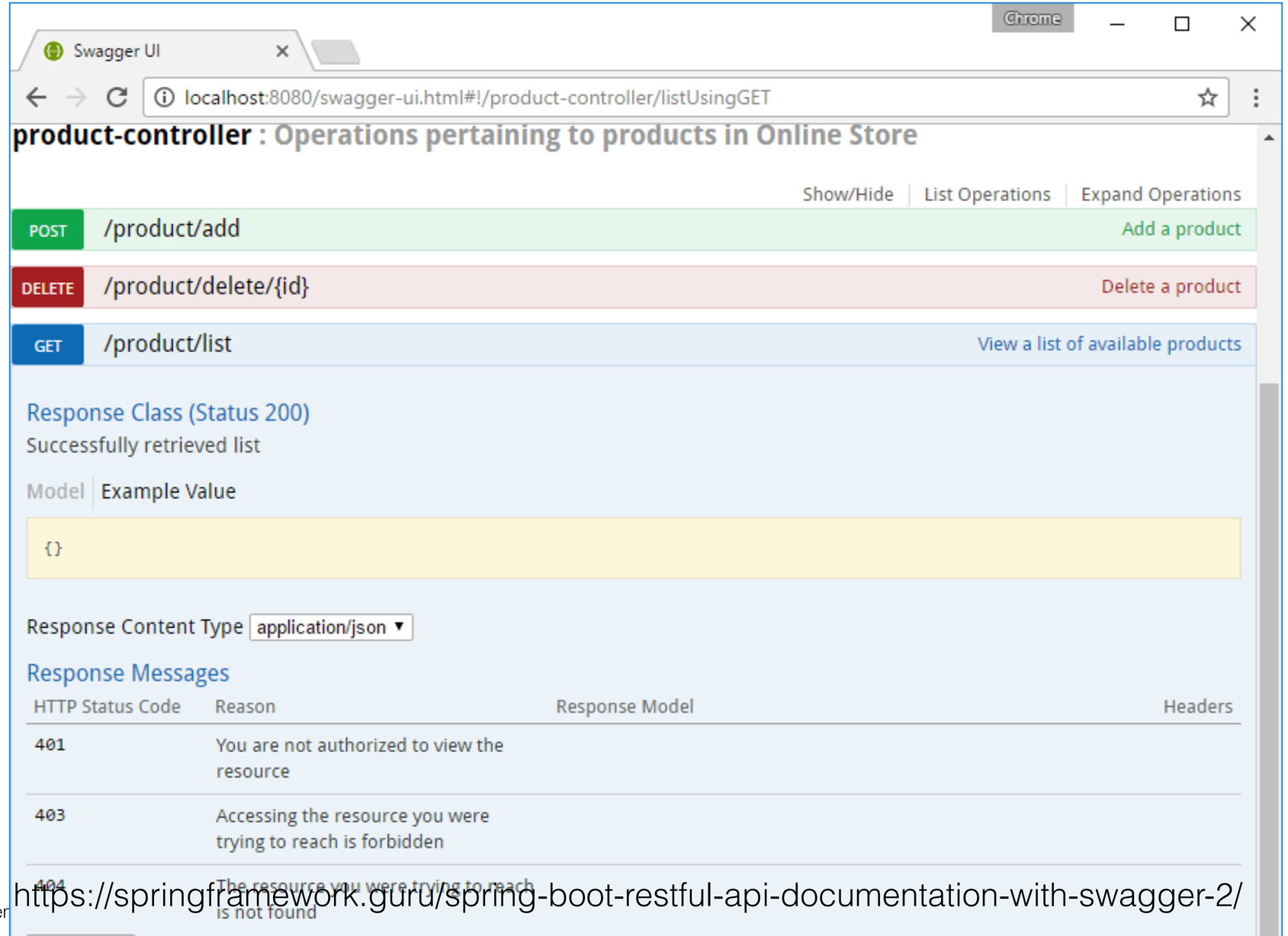
HTTP Status Code	Reason	Response Model	Headers
200	OK		
201	Created		
401	Unauthorized		
403	Forbidden		
404	Not Found		
405	Invalid input		

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



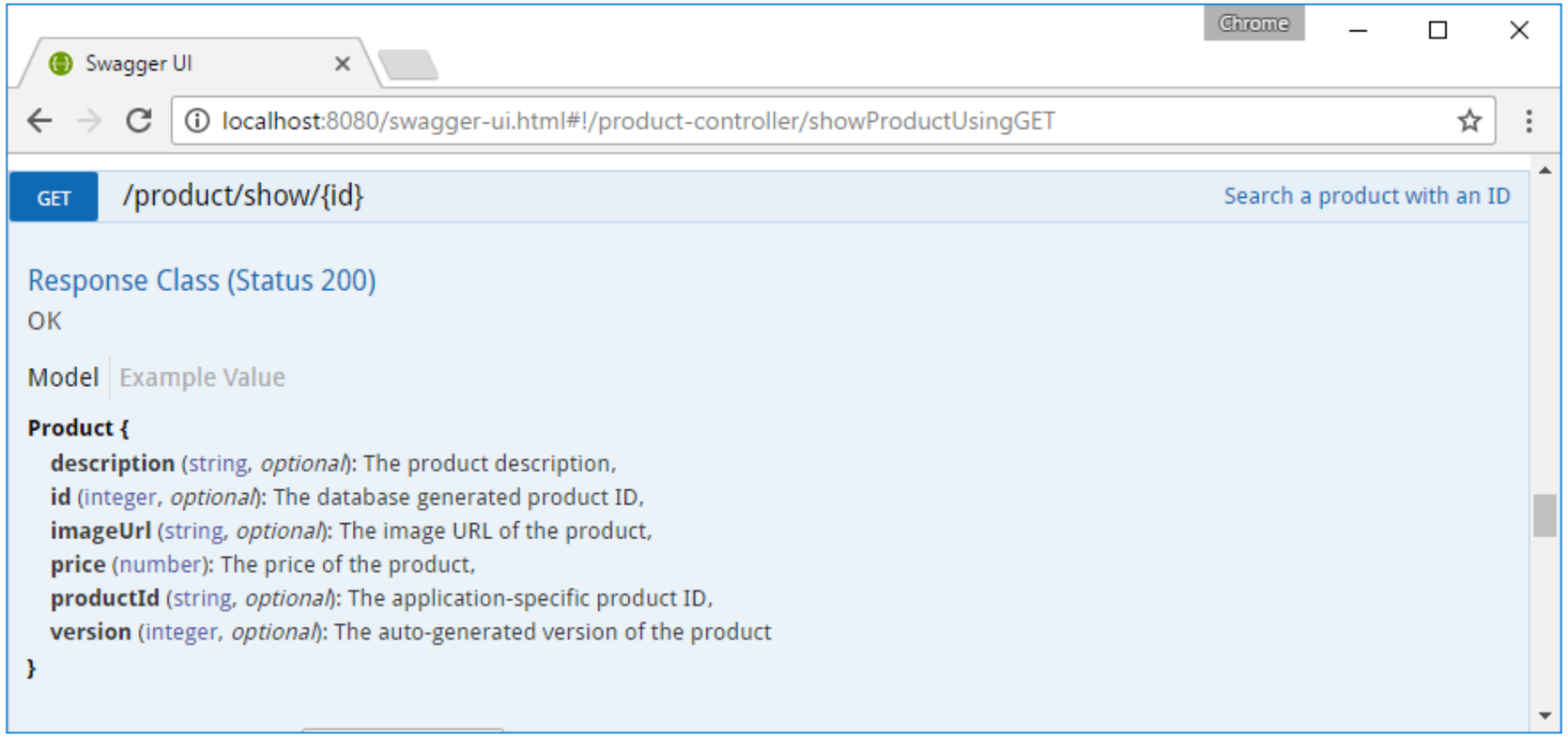
https://springframework.guru/spring-boot-restful-api-documentation-with-swagger-2/

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

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)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Lab Class 2&3

Swagger/Rest in Spring and Kotlin

Lab class 1 - Introduction to Spring and Swagger

- In this lab you will learn about using spring boot initializer, to build a controller for a REST API and use swagger editor to specify a functional API implementation. (editor.swagger.io)
- Requirements: Maven, IDE (Eclipse or IntelliJ)
- Lab Protocol:

1. (done?) Read the project assignment, and design the corresponding ER diagram
2. Write down the resources identified, and its subresources from a REST design perspective
3. Create an empty spring application using SpringBoot initializer, from the IDE, or using the URL start.spring.io.

3.1. Choose appropriate names for the different meta data fields, and choose kotlin as the base language

3.2. Let's keep the stable version 2.1.8 of SpringBoot

3.3. Add the module Spring Web under the Web category

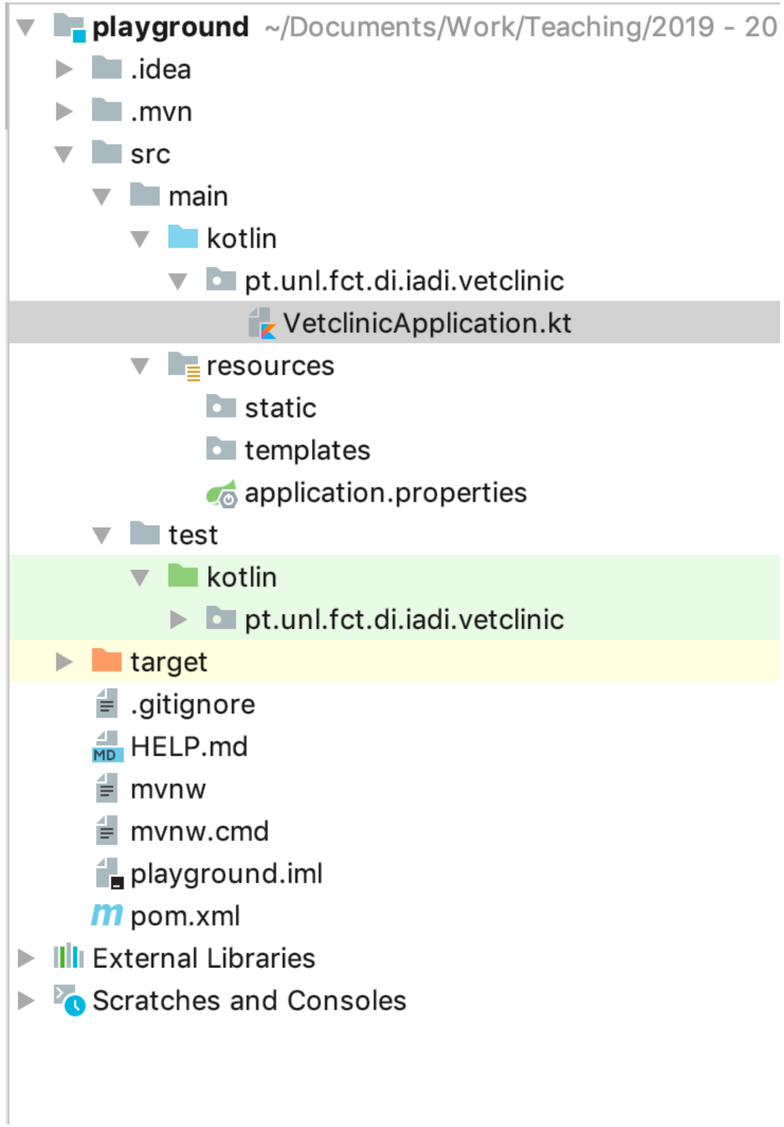
3.4.Be sure to let the IDE and Maven update all dependencies automatically

3.5.Build and Run your application (It starts but does nothing!!)

Lab class 1 - Introduction to Spring and Swagger

- Lab Protocol:
  1. (done?) Read the project assignment, and design the corresponding ER diagram
  2. Write down the resources identified, and its subresources from a REST design perspective
  3. Create an empty spring application using SpringBoot initializer, from the IDE, or using the URL start.spring.io.
  4. Design (on paper) the API for the Grant Application resource (all operations, and subresources)
  5. Create a class controller for the Grant resource (sample code ahead) with fake data. Implement all the designed endpoints. Define the data classes necessary to transfer data (DTOs). Choose the status codes to be returned carefully.
  6. Add Swagger to your project and define the general configuration.
  7. Add documentation annotations to your controller so that swagger automatically generates the online documentation for your API
  8. Use swagger editor and documentation to learn more

Lab class 1 - Spring Project with Kotlin

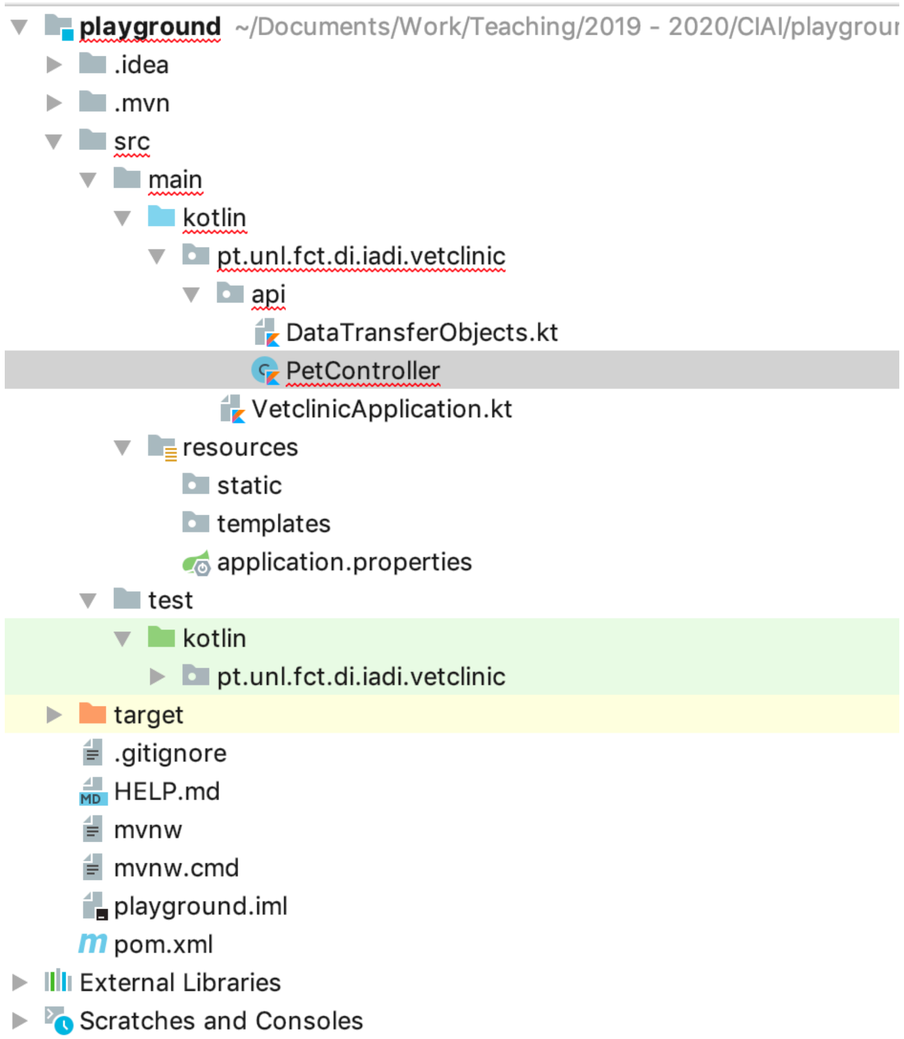


VetclinicApplication.kt

```kotlin
package pt.unl.fct.di.iadi.vetclinic

import org.springframework.boot.autoconfigure.SpringBootApplication
import org.springframework.boot.runApplication

@SpringBootApplication
class VetclinicApplication

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

```

```
Starting VetclinicApplicationKt on 10-170-134-79.docente.di.fct.unl.pt with PID 21595 (start

No active profile set, falling back to default profiles: default
Tomcat initialized with port(s): 8080 (http)
Starting service [Tomcat]
Starting Servlet engine: [Apache Tomcat/9.0.24]
Initializing Spring embedded WebApplicationContext
Root WebApplicationContext: initialization completed in 3454 ms
Initializing ExecutorService 'applicationTaskExecutor'
Tomcat started on port(s): 8080 (http) with context path ''
Started VetclinicApplicationKt in 5.222 seconds (JVM running for 6.834)
```

Lab class 1 - Controller




```kotlin
package pt.unl.fct.di.iadi.vetclinic.api

import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.PathVariable
import org.springframework.web.bind.annotation.RequestMapping
import org.springframework.web.bind.annotation.RestController

@RestController
@RequestMapping( ...value: "/pets")
class PetController {

    @GetMapping( ...value: "")
    fun getAllPets() = emptyList<PetDTO>()

    @GetMapping( ...value: "/{id}")
    fun getOnePet(@PathVariable id:Number) = PetDTO( id: 1,  name: "Pantufas",  species: "Dog")

}
```

```kotlin
package pt.unl.fct.di.iadi.vetclinic.api

data class PetDTO(val id:Number, val name: String, val species: String)
```

Lab class 1 - Calling a service

- use HTTPie, CURL, or Postman

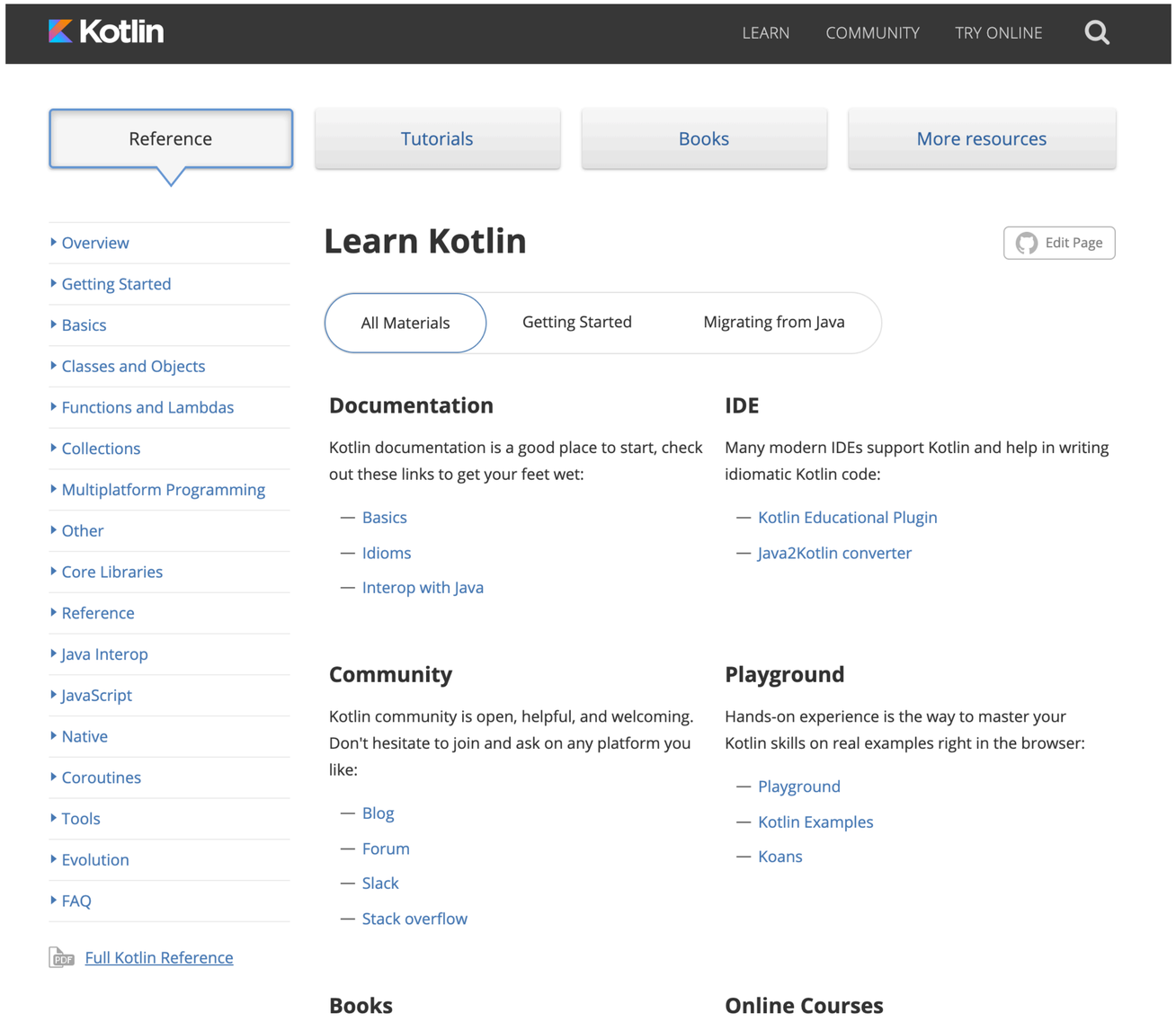
```
iadi-2019-20-private — -bash — 80×24
iadi 2019 $ http :8080/pets
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Tue, 24 Sep 2019 09:55:20 GMT
Transfer-Encoding: chunked

[]

iadi 2019 $ http :8080/pets/1
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Tue, 24 Sep 2019 10:00:08 GMT
Transfer-Encoding: chunked

{
    "id": 1,
    "name": "Pantufas",
    "species": "Dog"
}

iadi 2019 $
```

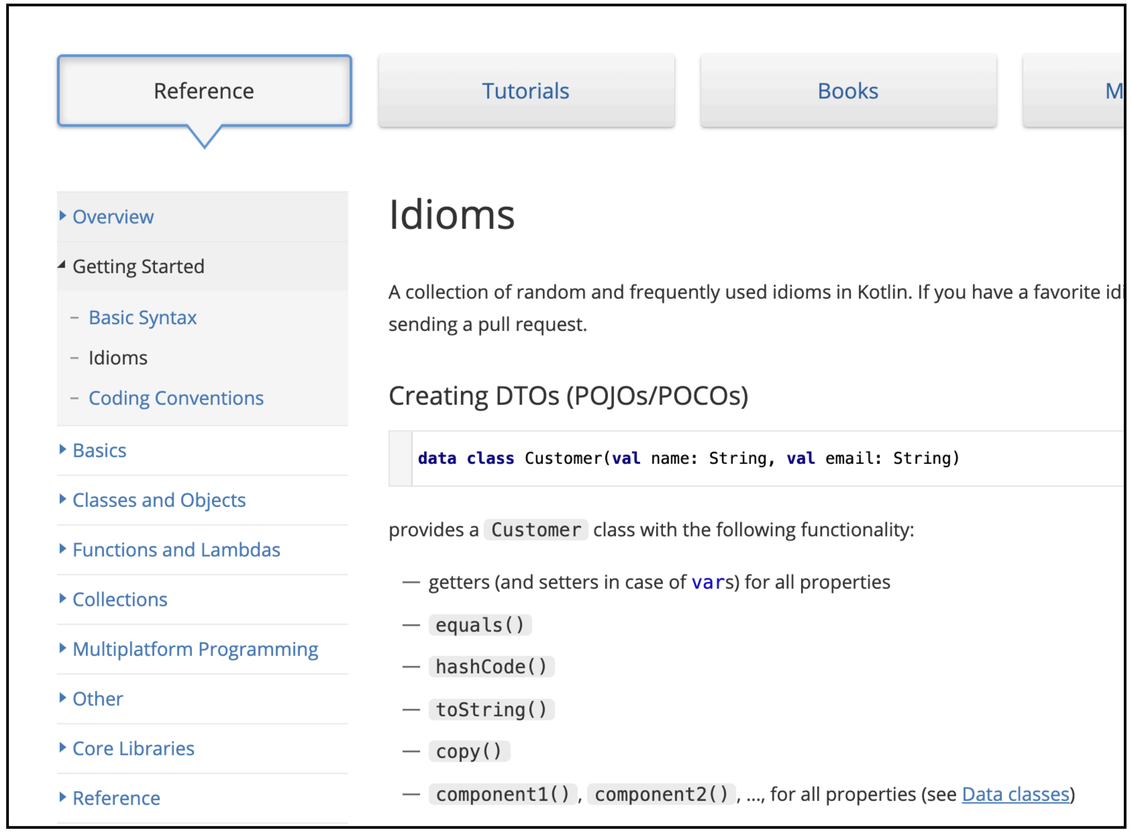
curl://

httpie

POSTMAN

Kotlin References





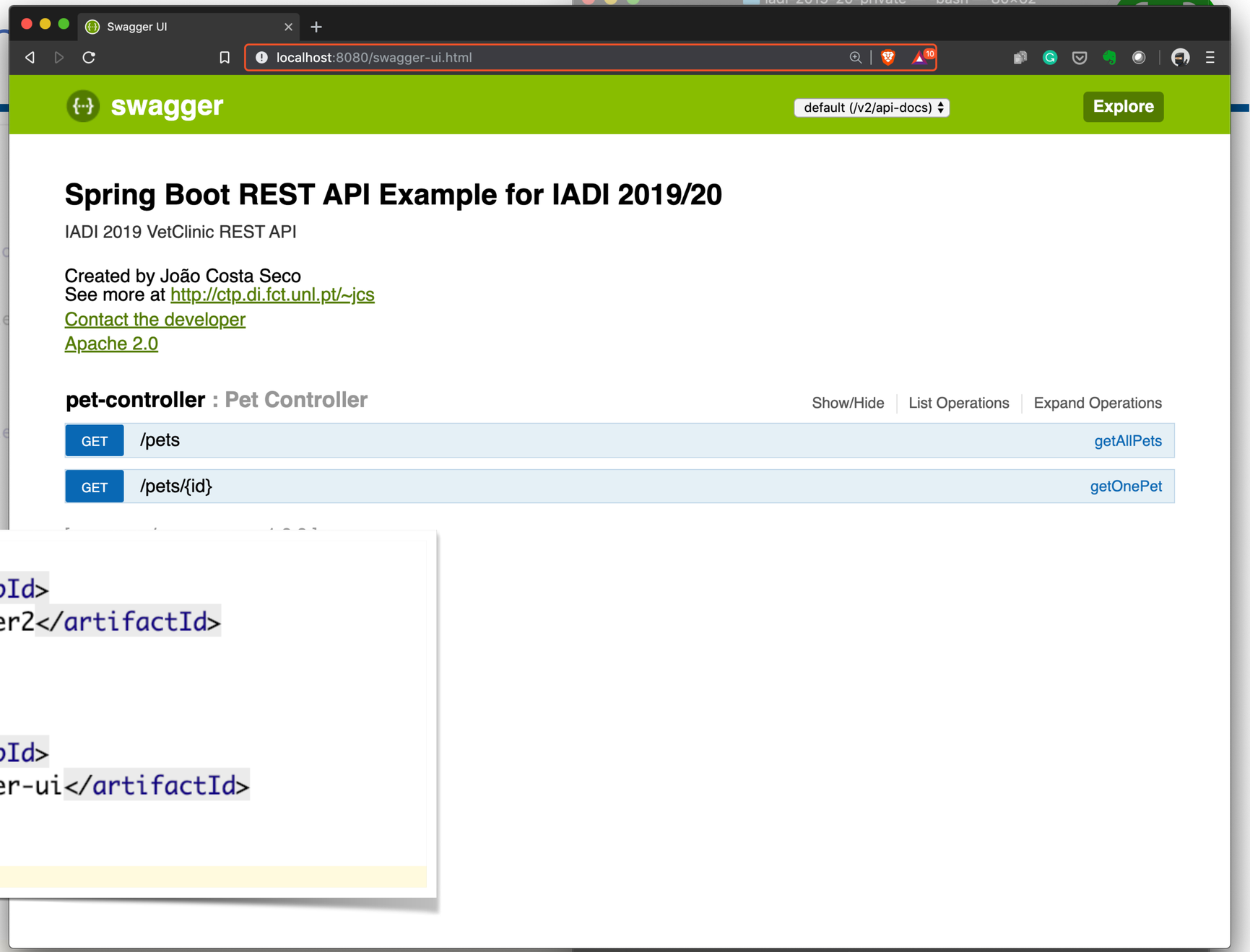
Lab class 1 - Document the API (Sw

```
/** Copyright 2019 João Costa Seco ...*/

package pt.unl.fct.di.iadi.vetclinic.config

import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.Configuration
import springfox.documentation.builders.ApiInfoBuilder
import springfox.documentation.builders.PathSelectors
import springfox.documentation.builders.RequestHandlerSelectors
import springfox.documentation.service.ApiInfo
import springfox.documentation.service.Contact
import springfox.documentation.spi.DocumentationType
import springfox.documentation.spring.web.plugins.Docket
import springfox.documentation.swagger2.annotations.EnableSwagger2

@Configuration
@EnableSwagger2
class SwaggerConfiguration {
    @Bean
    fun api(): Docket =
            Docket(DocumentationType.SWAGGER_2)
                    .select()
                    .apis(RequestHandlerSelectors.basePackage( basePackage: "pt.unl.fct.di.iadi.vetclinic"
                    .paths(PathSelectors.any())
                    .build().apiInfo(apiEndPointsInfo());

    fun apiEndPointsInfo(): ApiInfo =
            ApiInfoBuilder()
                    .title( title: "Spring Boot REST API Example for IADI 2019/20")
                    .description( description: "IADI 2019 VetClinic REST API")
                    .contact(Contact( name: "João Costa Seco", url: "http://ctp.di.fct.unl.pt/~jcs", ema
                    .license( license: "Apache 2.0")
                    .licenseUrl( licenseUrl: "http://www.apache.org/licenses/LICENSE-2.0.html")
                    .version( version: "1.0.0")
                    .build()

}
```

Lab class 1 - Docum

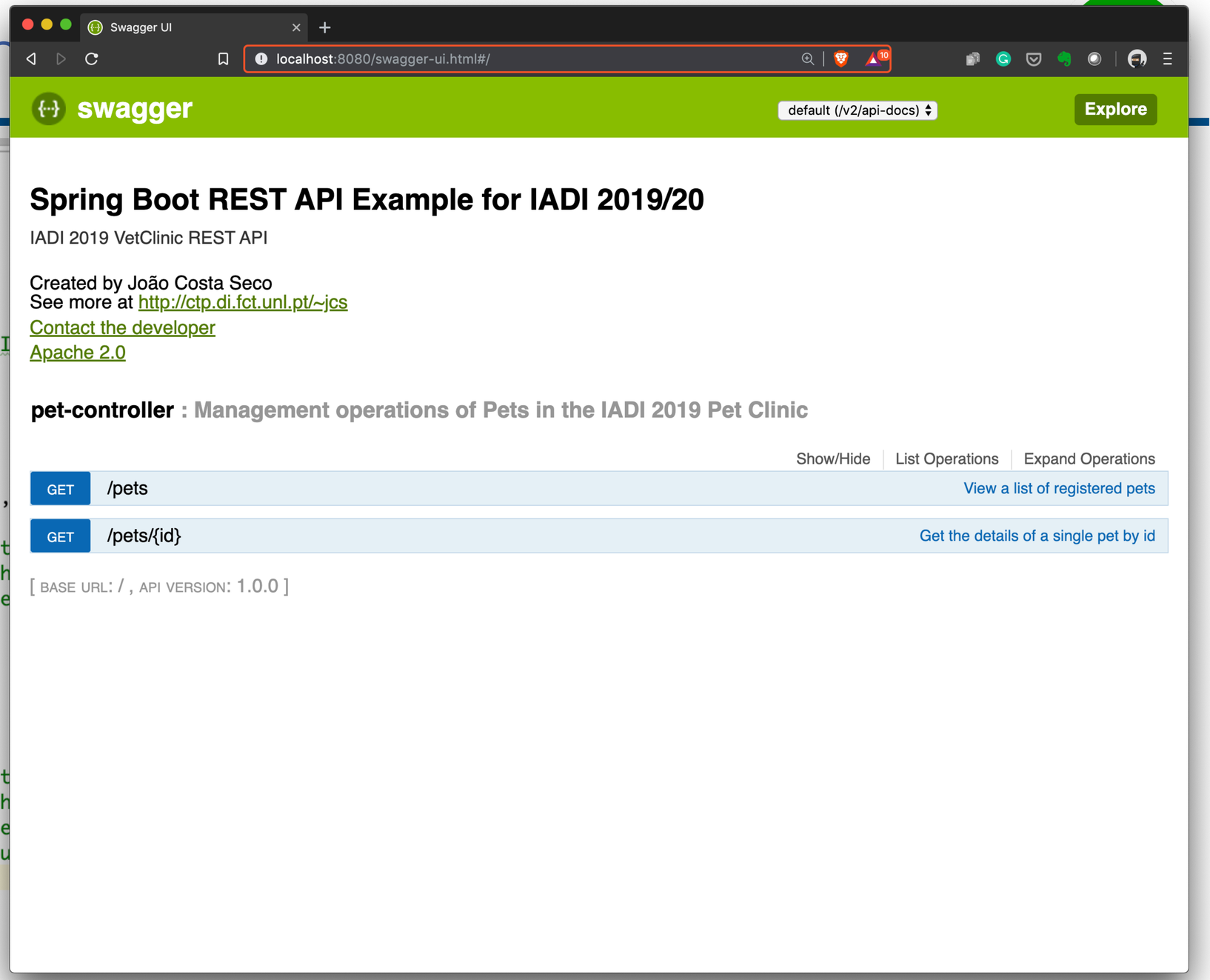
```
/** Copyright 2019 João Costa Seco ...*/

package pt.unl.fct.di.iadi.vetclinic.config

import org.springframework.context.annotation.Bean
import org.springframework.context.annotation.Configuratio
import springfox.documentation.builders.ApiInfoBuilder
import springfox.documentation.builders.PathSelectors
import springfox.documentation.builders.RequestHandlerSele
import springfox.documentation.service.ApiInfo
import springfox.documentation.service.Contact
import springfox.documentation.spi.DocumentationType
import springfox.documentation.spring.web.plugins.Docket
import springfox.documentation.swagger2.annotations.Enable

@Configuration
@EnableSwagger2
class SwaggerConfiguration {
```

```
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger2</artifactId>
            <version>2.7.0</version>
        </dependency>
        <dependency>
            <groupId>io.springfox</groupId>
            <artifactId>springfox-swagger-ui</artifactId>
            <version>2.7.0</version>
        </dependency>
```



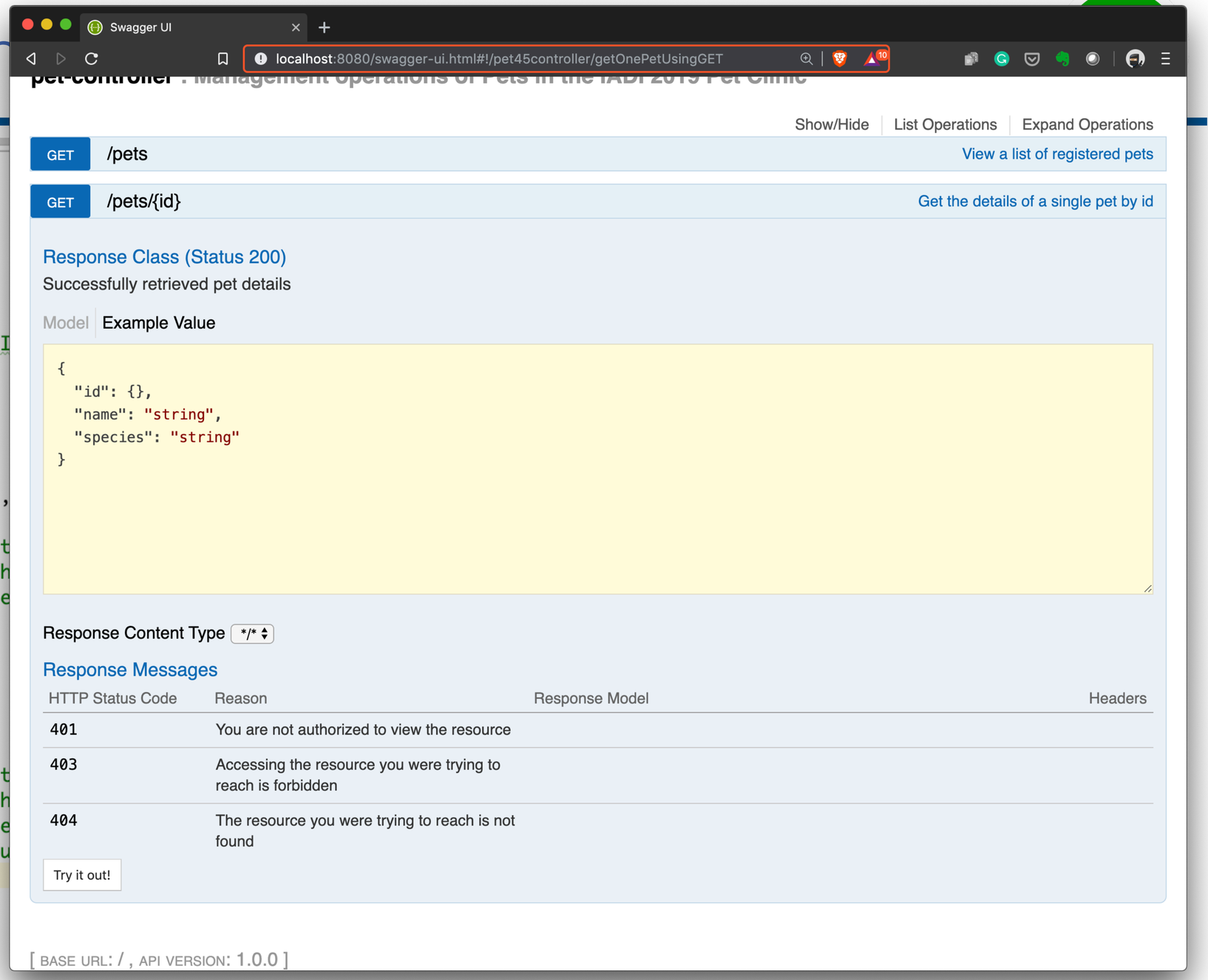
Lab class 1 - Documer

```
/** Copyright 2019 João Costa Seco ...*/

package pt.unl.fct.di.iadi.vetclinic.api

import ...

@Api(value="VetClinic Management System - Pet API",
        description="Management operations of Pets in the I

@RestController
@RequestMapping( ...value: "/pets")
class PetController {

    @ApiOperation(value = "View a list of registered pets",
    @ApiResponses(value = [
        ApiResponse(code = 200, message = "Successfully ret
        ApiResponse(code = 401, message = "You are not auth
        ApiResponse(code = 403, message = "Accessing the re
    ])
    @GetMapping( ...value: "")
    fun getAllPets() = emptyList<PetDTO>()

    @ApiOperation(value = "Get the details of a single pet
    @ApiResponses(value = [
        ApiResponse(code = 200, message = "Successfully ret
        ApiResponse(code = 401, message = "You are not auth
        ApiResponse(code = 403, message = "Accessing the re
        ApiResponse(code = 404, message = "The resource you
    ])
    @GetMapping( ...value: "/{id}")
    fun getOnePet(@PathVariable id:Number) = PetDTO( id: 1,

}
```



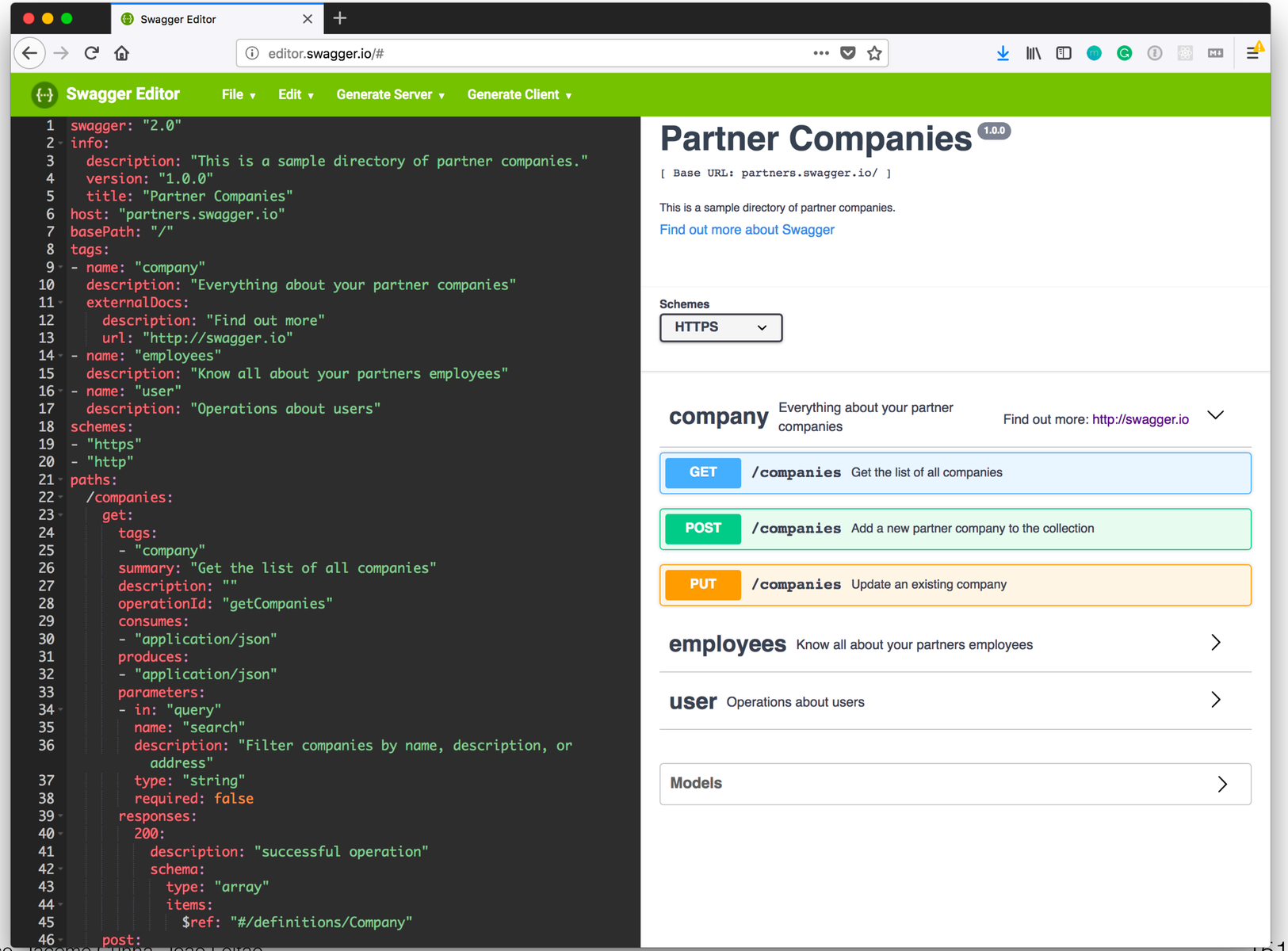
Lab class 1 - Documer

```kotlin
/** Copyright 2019 João Costa Seco ...*/

package pt.unl.fct.di.iadi.vetclinic.api

import ...

@Api(value="VetClinic Management System - Pet API",
        description="Management operations of Pets in the I

@RestController
@RequestMapping( ...value: "/pets")
class PetController {

    @ApiOperation(value = "View a list of registered pets",
    @ApiResponses(value = [
        ApiResponse(code = 200, message = "Successfully ret
        ApiResponse(code = 401, message = "You are not auth
        ApiResponse(code = 403, message = "Accessing the re
    ])
    @GetMapping( ...value: "")
    fun getAllPets() = emptyList<PetDTO>()

    @ApiOperation(value = "Get the details of a single pet
    @ApiResponses(value = [
        ApiResponse(code = 200, message = "Successfully ret
        ApiResponse(code = 401, message = "You are not auth
        ApiResponse(code = 403, message = "Accessing the re
        ApiResponse(code = 404, message = "The resource you
    ])
    @GetMapping( ...value: "/{id}")
    fun getOnePet(@PathVariable id:Number) = PetDTO( id: 1,

}
```



Use Swagger documentation to learn details

- Editor
- Documentation
- Tutorials online



Internet Applications Design and Implementation 2020 - 2021

(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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outline

- Recap REST Architectural Style
- Server Side Patterns
  - Model View Controller
  - Dependency Injection
  - Builder
- Data Sources in MVC
- ORM
- JPA and Spring

Internet Applications Design and Implementation 2020 - 2021

(Lecture 3 - Part 1 - 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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

RESTful design

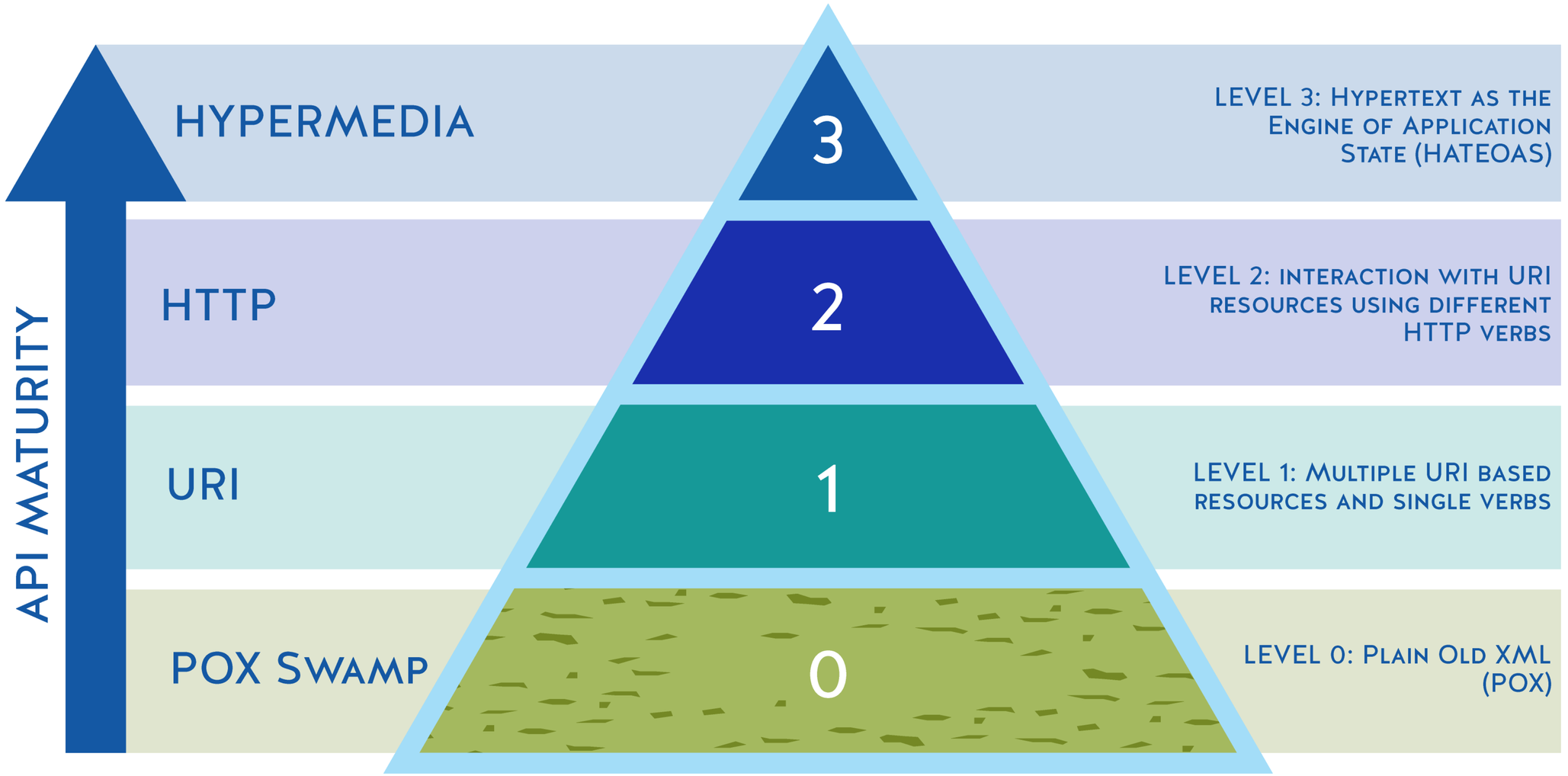
RECAP

- 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

THE RICHARDSON MATURITY MODEL

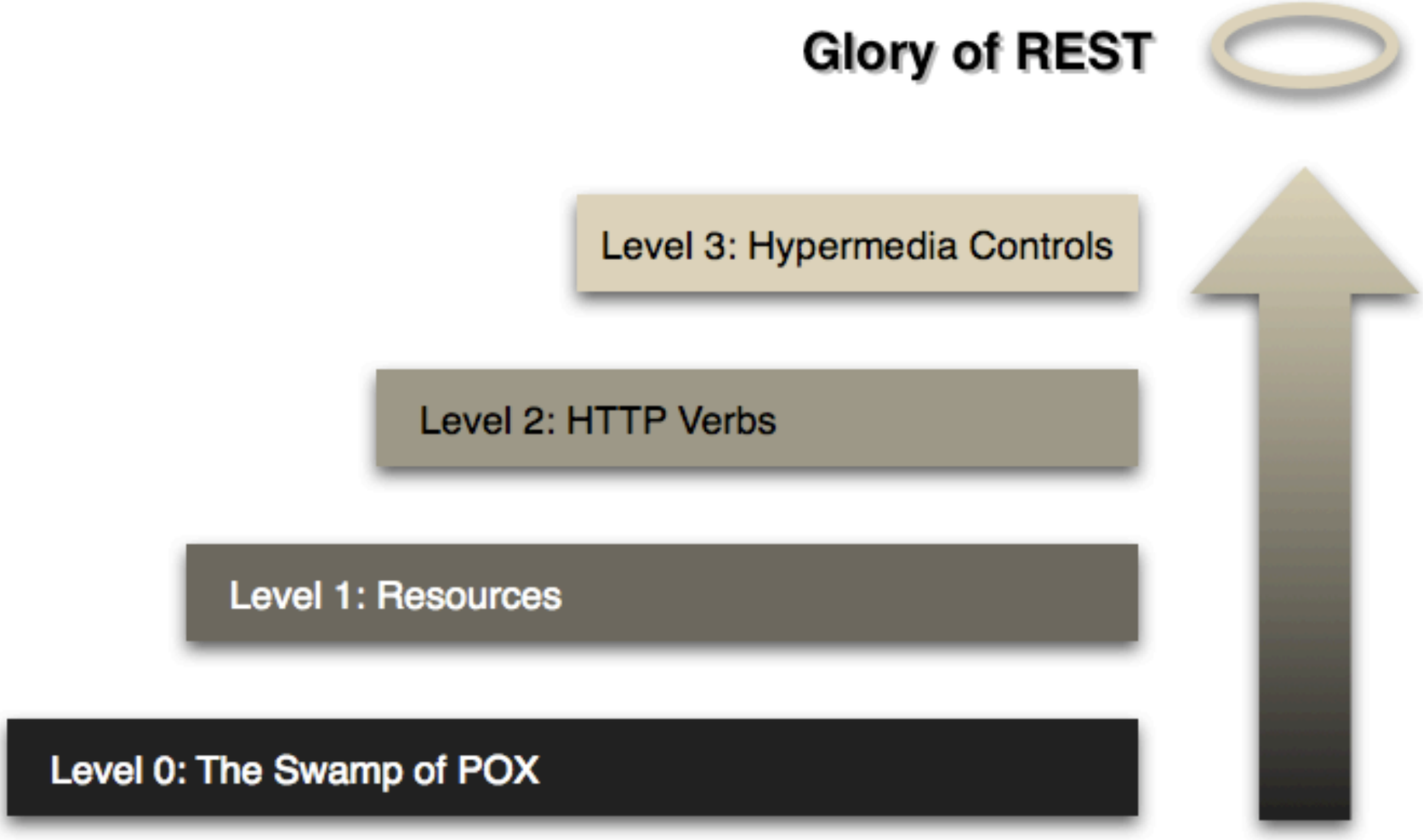
RECAP



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 3

- 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 4

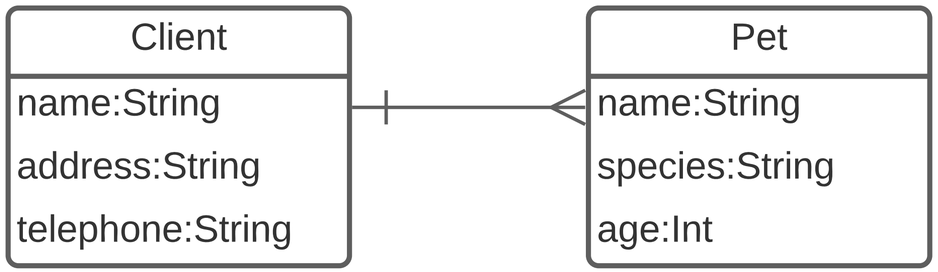
- Hypermedia Controls - HATEOAS
  - Resources are interconnected by links in the response, one entry point

```
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 2 - Server-side MVC Architecture)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

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

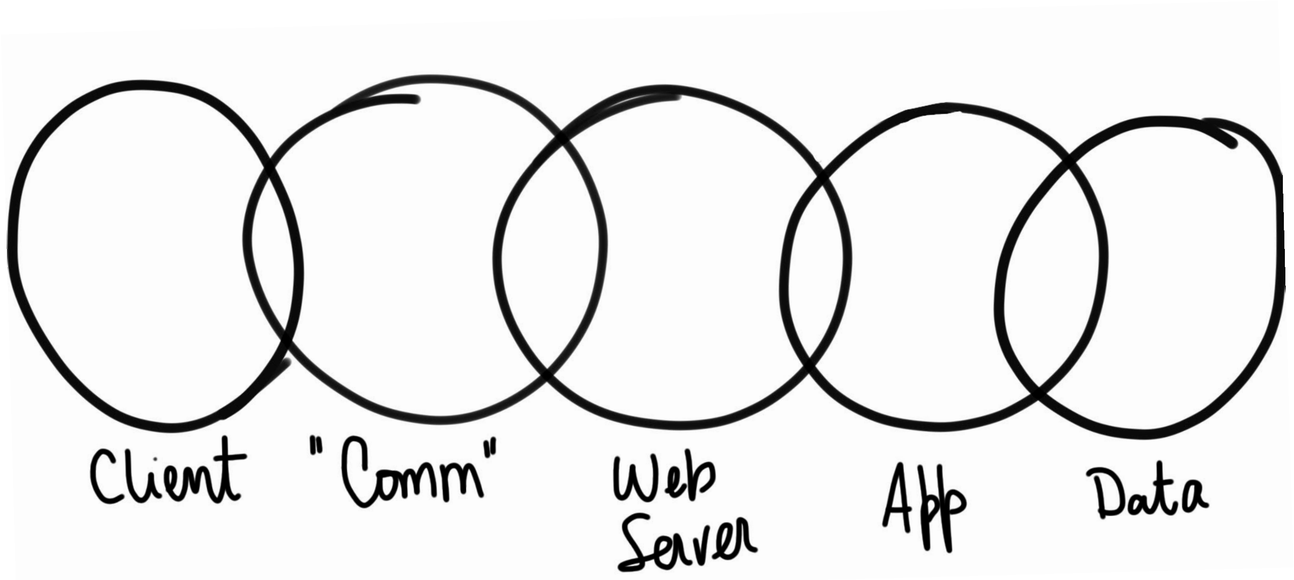
(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Web architectures, patterns and styles

RECAP

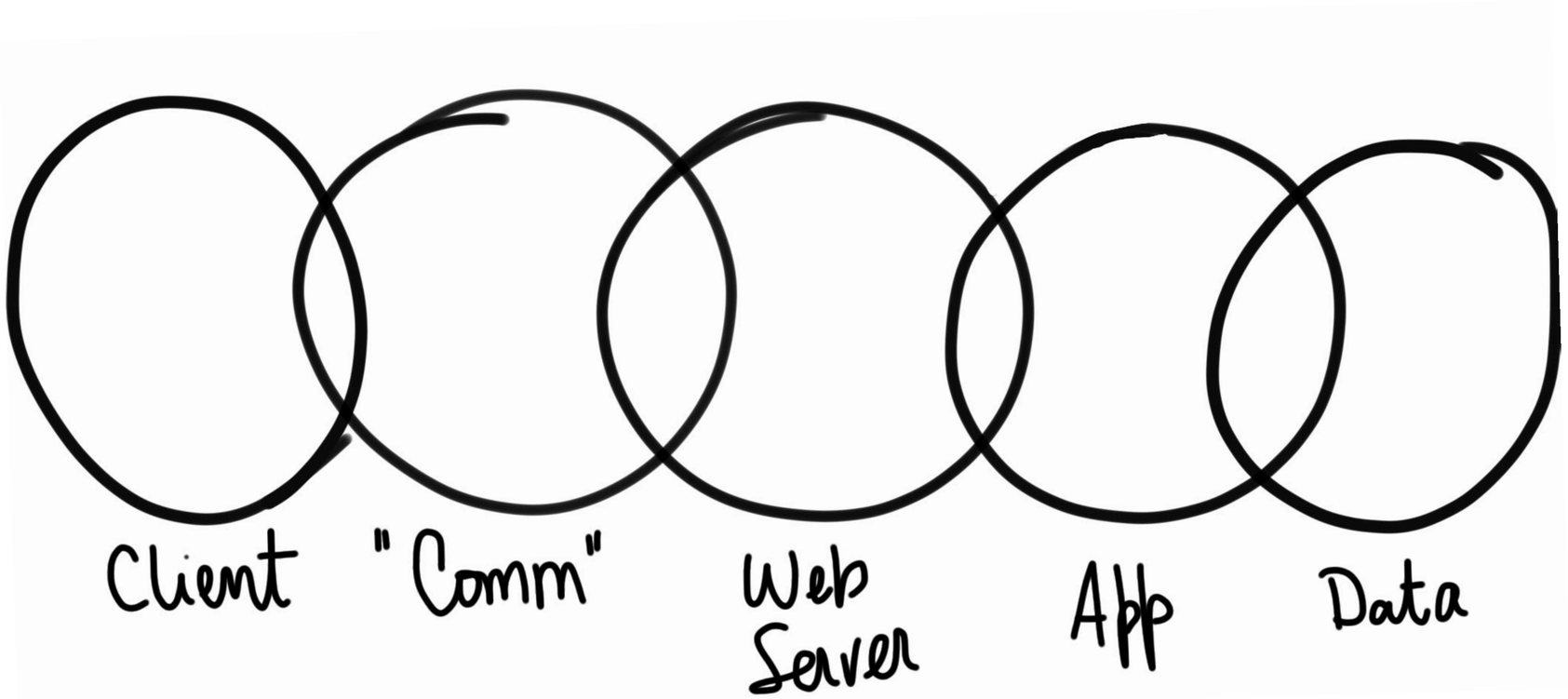
- Web applications usually follow a MVC architectural pattern.
  - Model layer - isolate the representation of persistent data and its operations, validations and conditions
  - Controller - contains the core application logic implementing the application interface (e.g. ad-hoc URL mapping, REST convention)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



Summary - Web Frameworks

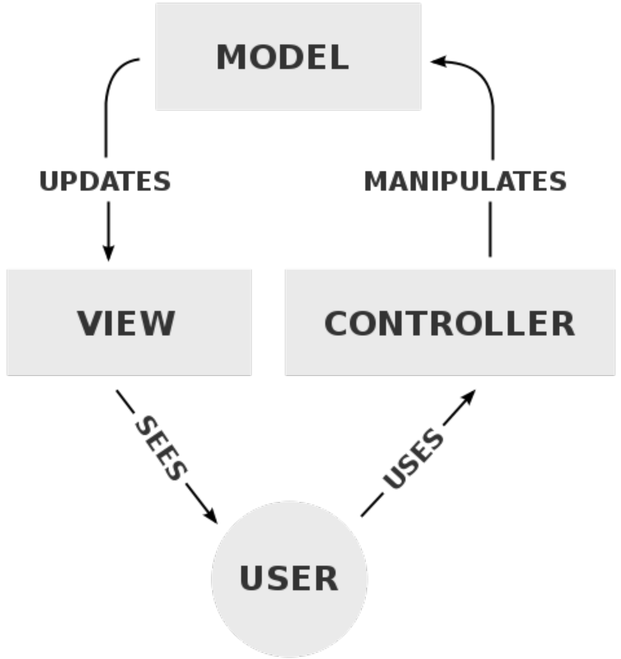
RECAP

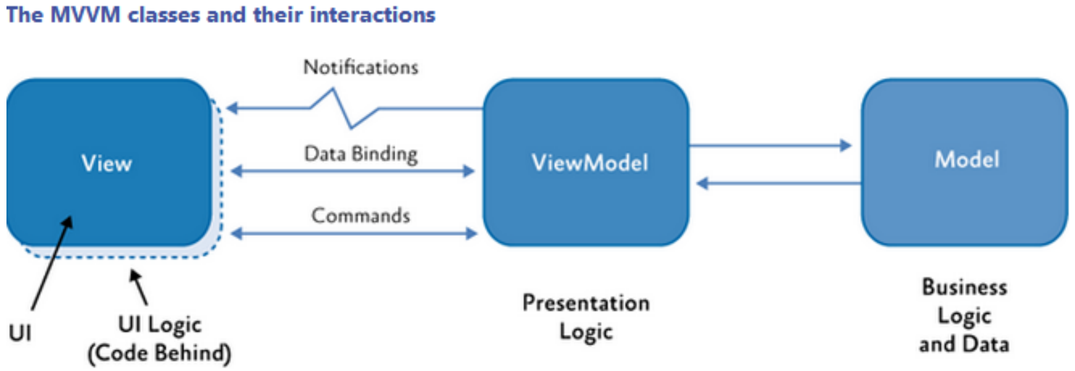
- Web Frameworks are “languages” that carry libraries and abstractions that get compiled to run on the “web virtual machine”.



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)

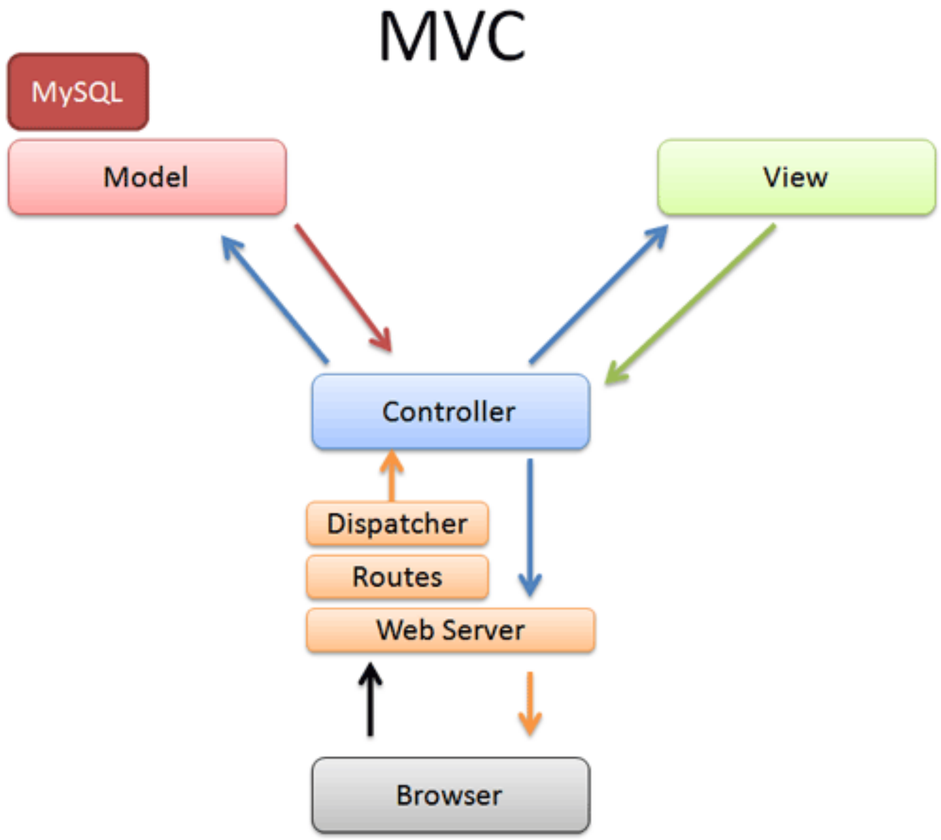




https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/

Frameworks and MVC Architecture

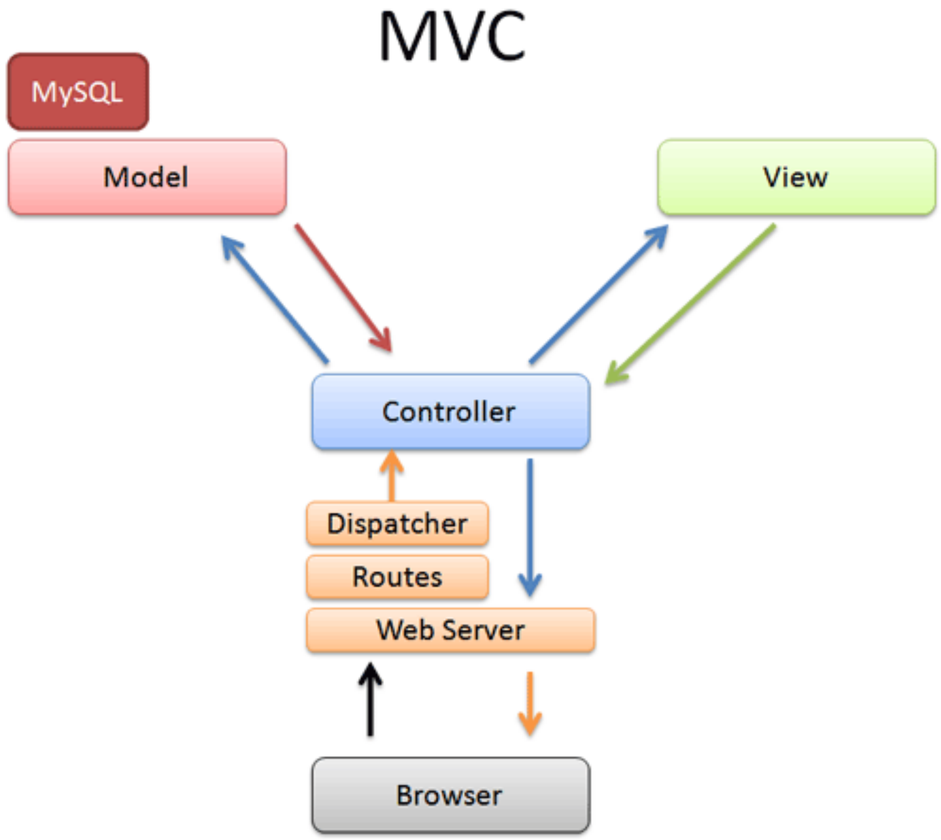
- Frameworks help to implement and maintain architectures.
- Rails (2005):
  - conventions on folder, file, and class names
  - A flexible OO prog language (Ruby) supports data sharing between model, controller, and view objects.



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
  - 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)

```
@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"));
    }

}
```

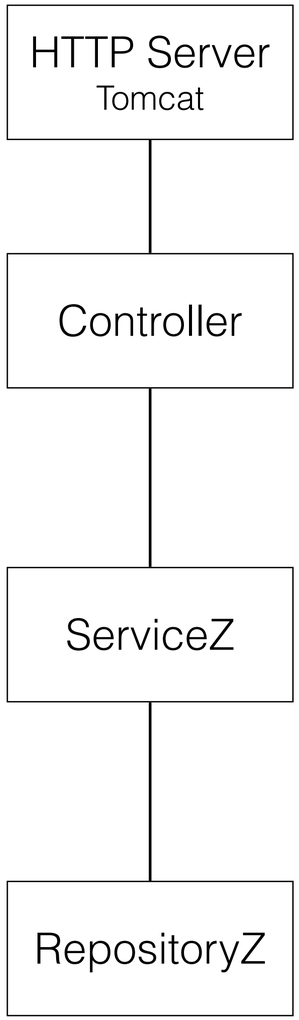
Add-ons to the MVC Framework

- 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

```
@RunWith(SpringRunner::class)
@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

```
@RunWith(SpringRunner::class)
@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)
    }
```

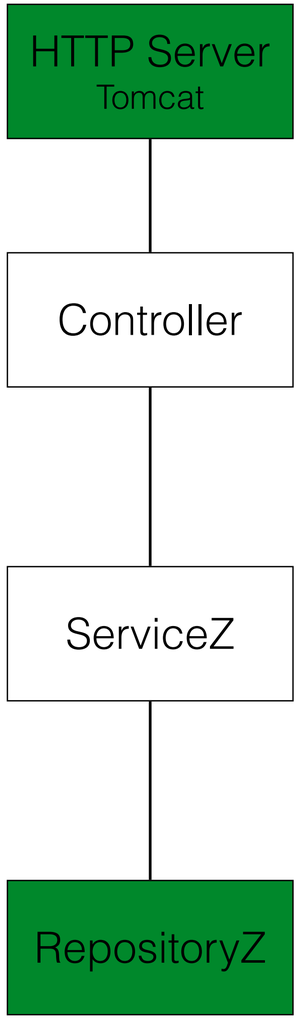
Replaces web server

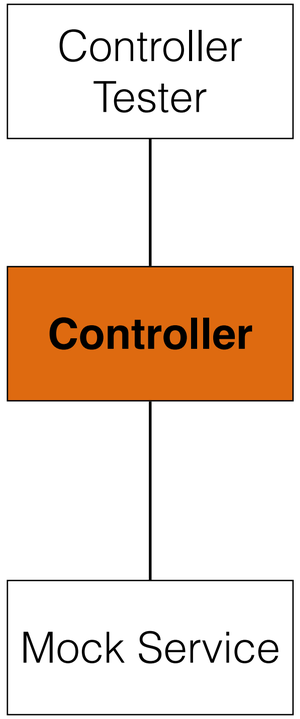
Replaces component with Mock object

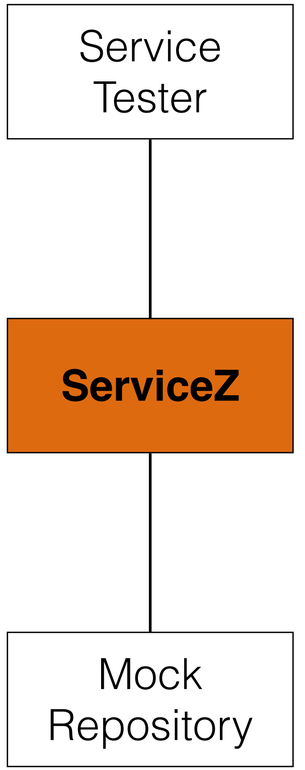
Replaces call results with expected values

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 3 - Data Sources in MVC)

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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Data Abstraction: The M in MVC

- An application layer that abstracts how information is stored, related, and protected.
- Examples of database languages, libraries and frameworks
  - JDBC
  - LINQ (in MVC ASP.NET)
  - ORMs (ActiveRecord in Rails, Hibernate in Java*)
  - NoSQL: MongoDB
  - External Web-services

JDBC

```
Connection conn = DriverManager.getConnection(
     "jdbc:somejdbcvendor:other data needed by some jdbc vendor",
     "myLogin",
     "myPassword" );
try {
     /* you use the connection here */
} finally {
    //It's important to close the connection when you are done with it
    try { conn.close(); } catch (Throwable ignore) { /* Propagate the original exception
instead of this one that you may want just logged */ }
}
```

```
try (Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery( "SELECT * FROM MyTable" )
) {
    while ( rs.next() ) {
        int numColumns = rs.getMetaData().getColumnCount();
        for ( int i = 1 ; i <= numColumns ; i++ ) {
           // Column numbers start at 1.
           // Also there are many methods on the result set to return
           //  the column as a particular type. Refer to the Sun documentation
           //  for the list of valid conversions.
           System.out.println( "COLUMN " + i + " = " + rs.getObject(i) );
        }
    }
}
```

JDBC

- Basic API for Java defining an access to a database
- Does not know the database schema
- Programmer needs to “manually” translate between data formats

```
try (Statement stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery( "SELECT * FROM MyTable" )
) {
     while ( rs.next() ) {
          int numColumns = rs.getMetaData().getColumnCount();
          for ( int i = 1 ; i <= numColumns ; i++ ) {
             // Column numbers start at 1.
             // Also there are many methods on the result set to return
             //  the column as a particular type. Refer to the Sun documentation
             //  for the list of valid conversions.
             System.out.println( "COLUMN " + i + " = " + rs.getObject(i) );
          }
     }
}
```

LINQ .NET

- Language based (integrated)
- Works in memory, xml, databases, etc
  - Based on the notion of **Provider**, it is extensible
- Programmers need to explicitly build objects that matches the database schema

```
var results =  from c in SomeCollection
               where c.SomeProperty < 10
               select new {c.SomeProperty, c.OtherProperty};

foreach (var result in results)
{
        Console.WriteLine(result);
}
```

LINQ .NET

- Language based (integrated)
- Works in memory, xml, databases, etc
  - Based on the notion of **Provider**, it is extensible
- Programmers need to explicitly build objects that matches the database schema

```
[Table(Name="Customers")]
public class Customer
{
      [Column(IsPrimaryKey = true)]
      public int CustID;

      [Column]
      public string CustName;
}
```

LINQ .NET

- Language based (integrated)
- Allows navigation using associations

```
// Query for customers who have placed orders.
var custQuery =
    from cust in Customers
    where cust.Orders.Any()
    select cust;

foreach (var custObj in custQuery)
{
    Console.WriteLine("ID={0}, Qty={1}", custObj.CustomerID,
        custObj.Orders.Count);
}
```

ActiveRecord in Rails

- Follows the active record pattern to implement an ORM. AR objects link to persistent data and define behaviour.
- Well integrated with the Model in the Rails MVC pattern
- Completely abstracts the database management by:
  - Object-mappings
  - Inheritance
  - Associations
  - Validations
  - Migrations (w/ Rails)

```
def create
  @author = Author.new(author_params)

  respond_to do |format|
    if @author.save
      format.html { redirect_to @author,
      format.json { render action: 'show'
    else
      format.html { render action: 'new'
      format.json { render json: @author.
    end
  end
end
```

NoSQL databases

- Not only SQL.
  - Document-based, Chave-Valor, Graph-based
- Desenhadas para escalonamento horizontal (replicação)
- Compromisso na consistência dos dados
- Limited transactional support (real concurrency control)… use of chards
- Very good for querying, harder to get right on “writes”, indexes, etc.
- Queries are written in JavaScript, Java, SPARK, Map reduce algorithms…

Data abstraction in Internet and Web Apps

- Abstraction over the actual data model
  - Hides the actual database engine running beneath
  - Integrates smoothly with the programming model (objects instead of string-based results).
- Independent configuration modes
  - Allows different execution modes with the same code (e.g. in-memory, transactional, replicated, etc.)
- Does not really cover all data models smoothly:
  - SQL model (e.g. Hibernate, ActiveRecord (Rn'R))
  - NoSQL model (e.g. google app engine framework)

Abstraction levels

- Connectivity (e.g. JDBC)
  - abstracts the connectivity and execution of queries.
  - you have to build queries, and parse and translate results
- Data translation (e.g. JPA, ActiveRecord, LINQ)
  - Translates data formats between program and database.
  - Integrates the language values typefully.
- Implementation and execution modes
  - Example: Google Cloud Datastore is a NoSQL document based storage that allows different implementations (cassandra, mongodb)

https://cloud.google.com/appengine/docs/java/datastore/

ORM

Object-relational impedance mismatch

- Most databases are based on relational algebra
- Applications define object oriented representations (object graphs)

346 systems in ranking, October 2018

Rank			DBMS	Database Model	Score		
Oct 2018	Sep 2018	Oct 2017			Oct 2018	Sep 2018	Oct 2017
1.	1.	1.	Oracle	Relational DBMS	1319.27	+10.15	-29.54
2.	2.	2.	MySQL	Relational DBMS	1178.12	-2.36	-120.71
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1058.33	+7.05	-151.99
4.	4.	4.	PostgreSQL	Relational DBMS	419.39	+12.97	+46.12
5.	5.	5.	MongoDB	Document store	363.19	+4.39	+33.79
6.	6.	6.	DB2	Relational DBMS	179.69	-1.38	-14.90
7.	↑ 8.	↑ 9.	Redis	Key-value store	145.29	+4.35	+23.24
8.	↓ 7.	↑ 10.	Elasticsearch	Search engine	142.33	-0.28	+22.09
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	136.80	+3.41	+7.35
10.	10.	↓ 8.	Cassandra	Wide column store	123.39	+3.83	-1.40

https://db-engines.com/en/ranking

Object-relational impedance mismatch

- Most databases are based on relational algebra
- Applications define object oriented representations (object graphs)

355 systems in ranking, October 2019

Rank			DBMS	Database Model	Score		
Oct 2019	Sep 2019	Oct 2018			Oct 2019	Sep 2019	Oct 2018
1.	1.	1.	Oracle	Relational, Multi-model	1355.88	+9.22	+36.61
2.	2.	2.	MySQL	Relational, Multi-model	1283.06	+3.99	+104.94
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1094.72	+9.66	+36.39
4.	4.	4.	PostgreSQL	Relational, Multi-model	483.91	+1.66	+64.52
5.	5.	5.	MongoDB	Document	412.09	+2.03	+48.90
6.	6.	6.	IBM Db2	Relational, Multi-model	170.77	-0.79	-8.91
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	150.17	+0.90	+7.85
8.	8.	↓ 7.	Redis	Key-value, Multi-model	142.91	+1.01	-2.38
9.	9.	9.	Microsoft Access	Relational	131.18	-1.53	-5.62
10.	10.	10.	Cassandra	Wide column	123.22	-0.18	-0.17

https://db-engines.com/en/ranking

Object-Relational Impedance Mismatch

- What's the difference?
  - Encapsulation
  - Interface, class, polymorphism
  - Mapping relational concepts
  - Data type differences
  - Structural and integrity differences
  - Transactional differences

http://blogs.tedneward.com/post/the-vietnam-of-computer-science/

https://dev.to/alagrede/why-i-dont-want-use-jpa-anymore-fl

Object-Relational Impedance Mismatch

- Identity
  - Object: a == b and a.equals(b)
  - Relational: primary key based
- Inheritance
  - Object: natural relation
  - Relational: does not exist, idioms are necessary
- Accessing data
  - Object: through the object interface
  - Relational: select queries (with joins)
- Associations/Navigation
  - Object: unidirectional references through objects' interface
  - Relational: through foreign keys
- Granularity
  - In some cases there may be a difference in the granularity level

Object-Relational Impedance Mismatch

- Identity based on Primary Keys,
- We must define method `equals` (or utils like Lombok).

```
@Entity
public class Person {

    @Id
    @GeneratedValue
    private long id;

    private String name;

    public Person() {}

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Object-Relational Impedance Mismatch

- Inheritance: optional support for inheritance is provided by some frameworks

```
@Entity
@Inheritance(strategy= InheritanceType.JOINED)
@JsonSubTypes({
       @JsonSubTypes.Type(value = Professor.class, name = "PROFESSOR"),
        @JsonSubTypes.Type(value = Staff.class, name = "STAFF"),
        @JsonSubTypes.Type(value = Student.class, name = "STUDENT")
})
public class User {

   @Id
   @GeneratedValue
   private Long id_login;

   private String name;
…
}
```

```
@Entity
public class Student extends User {

    @Column
    private int number;

    public Student() {
        super();
    }

    public Student(String login,
                   String password,
                   String name,
                   String tel,
                   String email,
                   String address,
                   String type,
                   int number) {

        super( login,
               password,
               name,
               tel,
               email,
               address,
               type);
        this.number = number;
    }

    public int getNumber() {
        return number;
    }
}
```

Object-Relational Impedance Mismatch

- Inheritance: optional support for inheritance is provided by some frameworks

```
@Entity
@Inheritance(strategy= InheritanceType.JOINED)
@JsonSubTypes({
       @JsonSubTypes.Type(value = Professor.class, name = "PROFESSOR"),
        @JsonSubTypes.Type(value = Staff.class, name = "STAFF"),
        @JsonSubTypes.Type(value = Student.class, name = "STUDENT")
})
public class User {

   @Id
   @GeneratedValue
   private Long id_login;

   private
…
}
```

```
@Entity
public class Student extends User {

    @Column
    private int number;

    public Student() {
        super();
    }

    public Student(String login,
                   String password,
                   String name,
                   String tel,
                   String email,
                   String address,
                   String type,
                   int number) {

        super( login,
               password,
               name,
               tel,
               email,
               address,
```

- *MappedSuperclass* – the parent classes, can't be entities
- Single Table – the entities from different classes with a common ancestor are placed in a single table
- Joined Table – each class has its table and querying a subclass entity requires joining the tables
- Table-Per-Class – all the properties of a class, are in its table, so no join is required

https://www.baeldung.com/hibernate-inheritance

Object-Relational Impedance Mismatch

- Access to data by object navigation
- All fields are retrieved to memory instead of explicitly selected
- May result in navigation queries

```
Organization o = organizationRepository.findById(id).get();
System.out.println(o.getName() + o.getContactInfo());
```

```
@Entity
public class Organization {

    @Id
    @GeneratedValue
    private Long id_entity;

    @Column
    private String name;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "contact_id")
    private ContactInfo contactInfo;

    public Organization(){ }

   public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public ContactInfo getContactInfo() {
        return contactInfo;
    }

    public void setContactInfo(ContactInfo contactInfo) {
        this.contactInfo = contactInfo;
    }
    …
}
```

ActiveRecord Example

```ruby
class Album < ActiveRecord::Base
    has_many :tracks
end

class Track < ActiveRecord::Base
    belongs_to :album
end

album = Album.create(:title => 'Black and Blue', :performer => 'The Rolling Stones')
album.tracks.create(:track_number => 1, :title => 'Hot Stuff')
album.tracks.create(:track_number => 2, :title => 'Hand Of Fate')
album.tracks.create(:track_number => 3, :title => 'Cherry Oh Baby ')
album.tracks.create(:track_number => 4, :title => 'Memory Motel ')
album.tracks.create(:track_number => 5, :title => 'Hey Negrita')
album.tracks.create(:track_number => 6, :title => 'Fool To Cry')
album.tracks.create(:track_number => 7, :title => 'Crazy Mama')
album.tracks.create(:track_number => 8, :title => 'Melody (Inspiration By Billy
Preston)')

album = Album.create(:title => 'Sticky Fingers',:performer => 'The Rolling Stones')
album.tracks.create(:track_number => 1, :title => 'Brown Sugar')
album.tracks.create(:track_number => 2, :title => 'Sway')
album.tracks.create(:track_number => 3, :title => 'Wild Horses')
album.tracks.create(:track_number => 4,:title => 'Can\'t You Hear Me Knocking')
album.tracks.create(:track_number => 5, :title => 'You Gotta Move')
album.tracks.create(:track_number => 6, :title => 'Bitch')
album.tracks.create(:track_number => 7, :title => 'I Got The Blues')
album.tracks.create(:track_number => 8, :title => 'Sister Morphine')
album.tracks.create(:track_number => 9, :title => 'Dead Flowers')
album.tracks.create(:track_number => 10, :title => 'Moonlight Mile')
```

https://dzone.com/articles/simple-ruby-activerecord

Object-Relational Mapping

- In Java you navigate the object network.
- Not efficient to retrieve data from a RDBMS.
- Minimize the number of SQL queries by using JOINs and selecting the targeted entities from the start (pre-fetching).

```
from Cat as cat
    inner join cat.mate as mate
    left outer join cat.kittens as kitten
```

https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/queryhql.html#queryhql-joins

JPA

Java Persistance API (JPA)

JPA provides a POJO persistence model for ORM

```
@Entity
public class Customer {

    private int id;
    private String name;
    private Collection<Order> orders;

  // no-args constructor necessary
   public Costumer () {}

  // primary key required
    @Id  // property access is used
    public int getId() {
        return id;
    }

  // gets and sets required
    public void setId(int id) {
        this.id = id;
    }
   …
```

```
…
 public String getName() {
     return name;
 }

  public void setName(String name) {
      this.name = name;
  }

// also OneToOne, ManyToOne, and ManyToMany
  @OneToMany(cascade=ALL,
             mappedBy="customer")
  public Collection<Order>
                       getOrders() {
      return orders;
  }

  public void setOrders(
      Collection<Order> newValue) {
      this.orders = newValue;
  }
```

Java Persistance API (JPA)

```
@Entity
@Table(name="ORDER_TABLE")
public class Order {

    private int id;
    private String address;
    private Customer customer;

    @Id
    @Column(name="ORDER_ID")
    public int getId() {
        return id;
    }

    public void setId(int id){
        this.id = id;
    }
  …
```

```
…

    @Column(name="SHIPPING_ADDRESS")
    public String getAddress() {
        return address;
    }
    public void setAddress(
                String address) {
        this.address = address;
    }

    @ManyToOne()
  // foreign key: column used for join
    @JoinColumn(name="CUSTOMER_ID")
    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(
              Customer customer) {
        this.customer = customer;
    }
```

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Persistence : JPA & Hibernate)

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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outline

- Object Relational Mapping Features (JPA & Hibernate)
- Spring and Data Abstraction
- JPA Associations
- Optimization of data fetching

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Part 1 - Object Relational Mapping)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

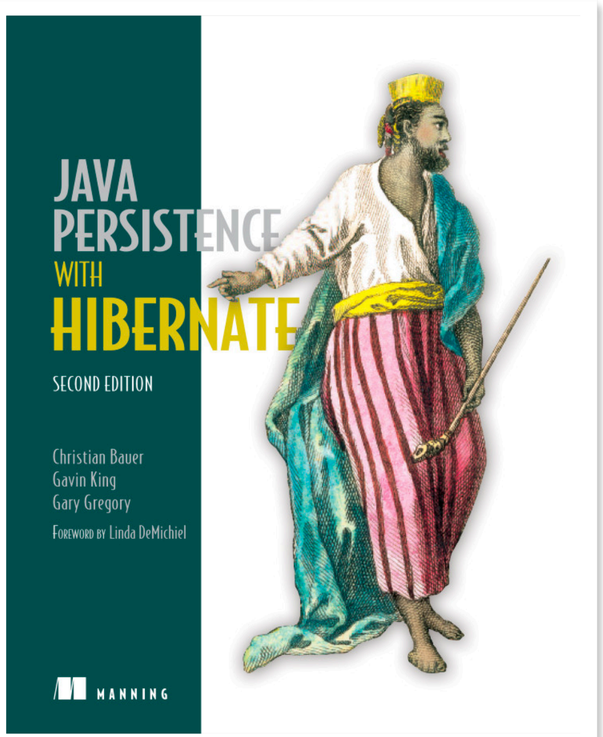
João Costa Seco (joao.seco@fct.unl.pt)

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

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Object-Relational Mapping

- Manages persistence of data in the OO realm
- Abstracts the use of SQL in connection to RDBMSs
- Covers most injection attacks on queries
  (except with APIs that allow creation with string)
- Specified by a common and standard API (JPA)
- Implementations differ by JPA providers




```
public List<AccountDTO> unsafeJpaFindAccountsByCustomerId(String customerId) {
    String jql = "from Account where customerId = '" + customerId + "'";
    TypedQuery<Account> q = em.createQuery(jql, Account.class);
    return q.getResultList()
      .stream()
      .map(this::toAccountDTO)
      .collect(Collectors.toList());
}
```

Object-Relational Impedance Mismatch

RECAP

- Identity
  - Object: a == b and a.equals(b)
  - Relational: primary key based
- Inheritance
  - Object: natural relation
  - Relational: does not exist, idioms are necessary
- Accessing data
  - Object: through the object interface
  - Relational: select queries (with joins)
- Associations/Navigation
  - Object: unidirectional references through objects' interface
  - Relational: through foreign keys
- Granularity
  - In some cases there may be a difference in the granularity level

http://hibernate.org/orm/what-is-an-orm/

JPA - Java Persistence API

- Java Persistence API (Jakarta Persistence since Set 2019)
- Provides a OO interface to a relational database
- Defines JPQL (Java Persistence Query Language)
- The reference implementation is EclipseLink.
- Hibernate is a JPA provider (extended API and query language HQL)

Data Abstraction layers

- Advantages
  - Hides the complexity of a particular query language
  - Allows the portability of database engines (not really models)
  - Prevents attacks such as SQL injection, or XSS
  - Avoids runtime errors in the construction of queries
  - May isolate efficient implementations (previously, with prepared and compiled parameterised SQL queries)
  - Allows scalability via customised runtime configurations (distribution, transactional behaviour, indexing, …)
  - Avoids early optimization pitfalls, develop first, configure later.
- Pitfalls
  - Lack of access to proprietary features of providers
  - May lead to inefficient data transmissions: more queries (N+1 queries), more data than needed.

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Part 2 - Spring & Data Abstraction)

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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

JDBC & Spring

- SpringBoot provides direct support for JDBC
- As in any JDBC setting, it is necessary to define a DataSource (DB, CSV file, etc.)
- SpringBoot offers the JdbcTemplate class to assist programmers
- Spring easily integrates JPA support (later)

JDBC & Spring & In-memory DBs

- JdbcTemplate handles the setup and connection to the DataSource based on the POM dependencies (when possible)

- For instance, for H2 in-memory database (Spring also supports HSQL and Derby):

```
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
    </dependency>
```

- This handles the connection and exceptions

JDBC & Spring

- It also supports “regular” relational databases adding the necessary properties to the **`application.properties`** file, e.g.:

```
spring.datasource.url=jdbc:mysql://localhost/test
spring.datasource.username=dbuser
spring.datasource.password=dbpass
```

JDBC & Spring

```
@Component
public class Hotels {

    private JdbcTemplate jdbc;

    @Autowired
    public Hotels(JdbcTemplate jdbc) {
        this.jdbc = jdbc;
        createTable();
    }

    public Hotels(){}

    public void createTable() {
        jdbc.execute("drop table tablea if exists");
        jdbc.execute("create table tablea(id SERIAL, attributea VARCHAR(16))");
    }

    public List<Map<String,Object>> select() {
        return jdbc.query("select * from tablea", new ColumnMapRowMapper());
    }

    public int save(String att) {
        return jdbc.update("insert into tablea(attributea) values (?)", att);
    }
}
```

JPA & Spring

- Spring-boot-starter-data-jpa provides the necessary dependencies:
  - Hibernate — One of the most popular JPA implementations
  - Spring Data JPA — Makes it easy to implement JPA-based repositories
  - Spring ORMs — Core ORM support from the Spring Framework

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
```

Declare an Entity (in Java)

```java
@Entity
public class Customer {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;

    protected Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    @Override
    public String toString() {
        return String.format(
                "Customer[id=%d, firstName='%s', lastName='%s']",
                id, firstName, lastName);
    }
}
```

Declare a Repository and plug it in… (in Java)

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {

    List<Customer> findByLastName(String lastName);
}
```

```
@Controller
public class CustomerController
{
    @Autowired
    CustomerRepository customers;

   … findByLastName(“Smith”);…
}
```

Declare an Entity and a Repository and plug it in…

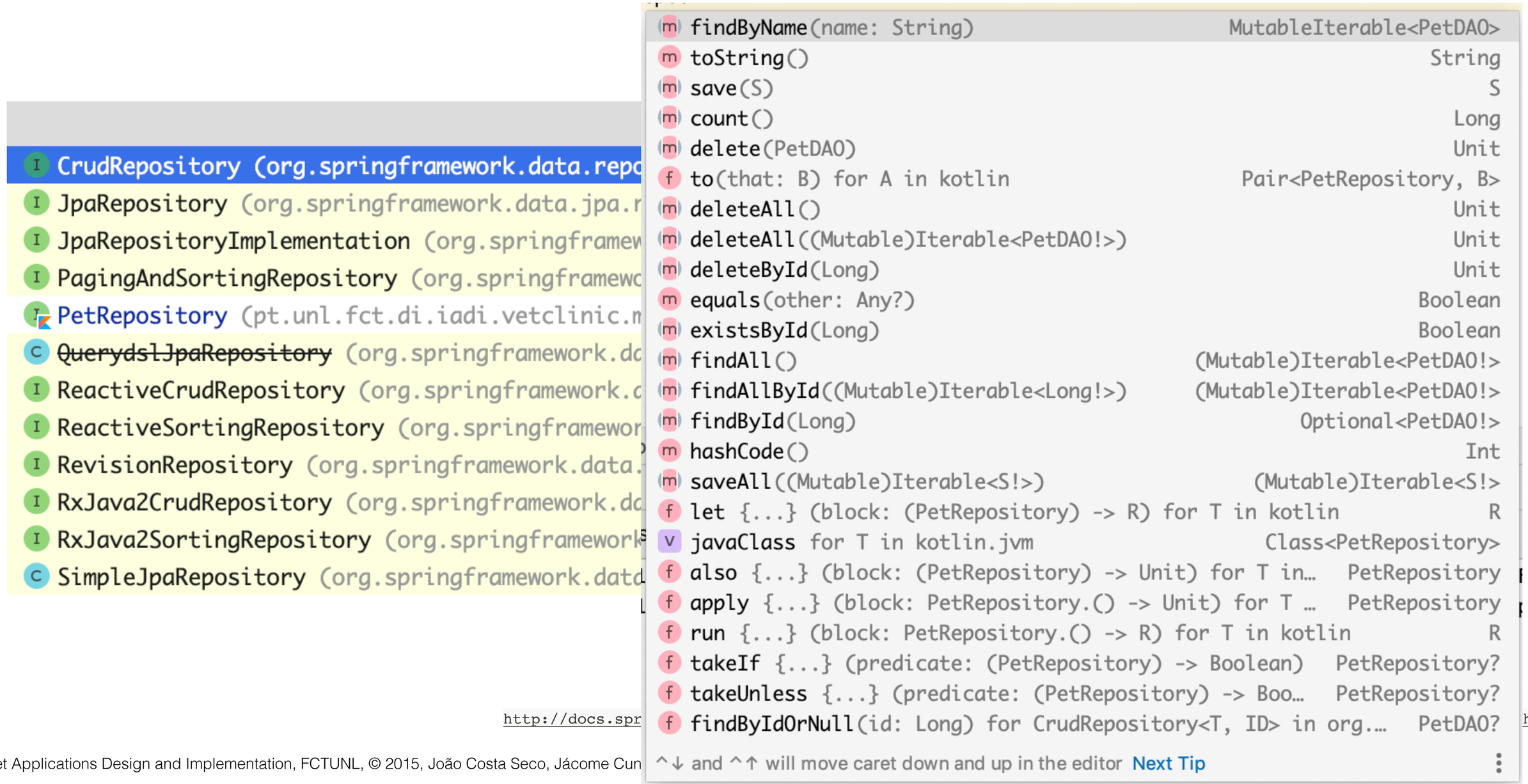
```
@Entity
data class PetDAO(
        @Id @GeneratedValue val id:Long,
        var name: String,
        var species: String
)
```

```
interface PetRepository : CrudRepository<PetDAO, Long> {
    fun findByName(name:String): MutableIterable<PetDAO>
}
```

```
@Autowired
lateinit var repo:PetRepository

fun someFunction() {

…repo.findByName(“pantufas”)…

}
```

```
findByName(name: String)                                  MutableIterable<PetDAO>
toString()                                                                 String
save(S)                                                                         S
count()                                                                     Long
delete(PetDAO)                                                              Unit
to(that: B) for A in kotlin                                 Pair<PetRepository, B>
deleteAll()                                                                 Unit
deleteAll((Mutable)Iterable<PetDAO!>)                                       Unit
                                                                            Unit
                                                                         Boolean
                                                                         Boolean
                                                  (Mutable)Iterable<PetDAO!>
                                (Mutable)Iterable<Long!>)  (Mutable)Iterable<PetDAO!>
                                                          Optional<PetDAO!>
hashCode()                                                                  Int
saveAll((Mutable)Iterable<S!>)                        (Mutable)Iterable<S!>
let {...} (block: (PetRepository) -> R) for T in kotlin                     R
javaClass for T in kotlin.jvm                          Class<PetRepository>
also {...} (block: (PetRepository) -> Unit) for T in…    PetRepository
apply {...} (block: PetRepository.() -> Unit) for T …    PetRepository
run {...} (block: PetRepository.() -> R) for T in kotlin                    R
takeIf {...} (predicate: (PetRepository) -> Boolean)     PetRepository?
takeUnless {...} (predicate: (PetRepository) -> Boo…     PetRepository?
findByIdOrNull(id: Long) for CrudRepository<T, ID> in org.…    PetDAO?
^↓ and ^↑ will move caret down and up in the editor  Next Tip
```

interface CrudRepository<T , ID extends Serializable>



http://docs.spr html

An Example

```
@Entity
@NamedQuery(name = "User.findByTheUsersName",
            query = "from User u where u.username = ?1")
class User(
            @Column(unique = true)
            val username:String,
            val firstname:String,
            val lastname:String
)
```

```
interface SimpleUserRepository : CrudRepository<User, Long> {

    fun findByTheUsersName(username:String):User

    fun findByLastname(lastname:String):List<User>

    @Query("select u from User u where u.firstname = ?")
    fun findByFirstname(firstname:String):List<User>

    @Query("select u from User u where u.firstname = :name or u.lastname = :name")
    fun findByFirstnameOrLastname(@Param("name") name:String):List<User>
}
```

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname =
Is,Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = 1?
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age ⇐ ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
TRUE	findByActiveTrue()	… where x.active = true
FALSE	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstame) = UPPER(?

Spring Data (and other ORM Implementations)

SUMMARY

- Simple declaration of generic queries
- Specific declaration of custom queries (JPQL)
- Richer Behaviour from Repositories (e.g. paged)

```
public interface StudentRepository extends PagingAndSortingRepository<Student, Long> {

    List<Student> findByName(String name);

    @Query("select s from Student s where s.name like CONCAT(?,'%')")
    List<Student> search(String name);

    Page<Student> findByName(String name, Pageable pageable);
}
```

- Dependency Injection and component assembly

```
public class StudentController {

    @Autowired
    StudentRepository students;
```

Spring Data (and other ORM Implementations)

SUMMARY

- Simple declaration of generic queries
- Specific declaration of custom queries (JPQL)
- Richer Behaviour from Repositories (e.g. paged)

```
public interface StudentRepository extends PagingAndSortingRepository<Student, Long> {

    List<Student> findByName(String name);
```

```
@RequestMapping(value= "/page")
public @ResponseBody Page<Student> getStudentsPaged(
     @RequestParam(required=false, defaultValue =  "0") Integer page,
     @RequestParam(required=false, defaultValue = "3") Integer size) {

   return students.findAll(new PageRequest(page, size));
}
```

- Depe

```
    @Autowired
    StudentRepository students;
```

Spring Data (and other ORM Imp

SUMMARY

- Simple declaration of generic queries
- Specific declaration of custom queries (JP
- Richer Behaviour from Repositories (e.g. p

```
public interface StudentRepository extends Pag

    List<Student> findByName(String name);
```

```
@RequestMapping(value= "/page")
public @ResponseBody Page<Student> ge
     @RequestParam(required=false, de
     @RequestParam(required=false, de

   return students.findAll(new PageRe
}
```

- Depe

```
@Autowired
StudentRepository students;
```

```
{
  "content": [
    {
      "id": 1,
      "name": "Ingrid Daubechies",
      "age": 19
    },
    {
      "id": 2,
      "name": "Jacqueline K. Barton",
      "age": 18
    },
    {
      "id": 3,
      "name": "Jane Goodall",
      "age": 20
    }
  ],
  "last": false,
  "totalElements": 6,
  "totalPages": 2,
  "size": 3,
  "number": 0,
  "sort": null,
  "first": true,
  "numberOfElements": 3
}
```

Spring Data - Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

Spring Data

- Spring Data Commons - Core Spring concepts underpinning every Spring Data project.
- Spring Data JPA - Makes it easy to implement JPA-based repositories.
- Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.
- Spring Data Redis - Provides easy configuration and access to Redis from Spring applications.
- Spring Data Solr - Spring Data module for Apache Solr.
- Spring Data Gemfire - Provides easy configuration and access to GemFire from Spring applications.
- Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.

Community Modules

- Spring Data Cassandra - Spring Data module for Apache Cassandra.
- Spring Data Couchbase - Spring Data module for Couchbase.
- Spring Data DynamoDB - Spring Data module for DynamoDB.
- Spring Data Elasticsearch - Spring Data module for Elasticsearch.
- Spring Data Neo4j - Spring based, object-graph support and repositories for Neo4j.

MongoDB Example

```
public class MongoApp {

  private static final Log log = LogFactory.getLog(MongoApp.class);

  public static void main(String[] args) throws Exception {

    MongoOperations mongoOps = new MongoTemplate(new Mongo(), "database");
    mongoOps.insert(new Person("Joe", 34));

    log.info(mongoOps.findOne(new Query(where("name").is("Joe")), Person.class));

    mongoOps.dropCollection("person");
  }
}
```

Other database representations

- Objectify: gives easy and full access to the Google Cloud Datastore
- Dynamic / Reflexion based
- Alternative to JPA interface

```
@Entity
class Car {
    @Id String vin; // Can be Long, long, or String
    String color;
}

ofy().save().entity(new Car("123123", "red")).now();
Car c = ofy().load().type(Car.class).id("123123").now();
ofy().delete().entity(c);
```

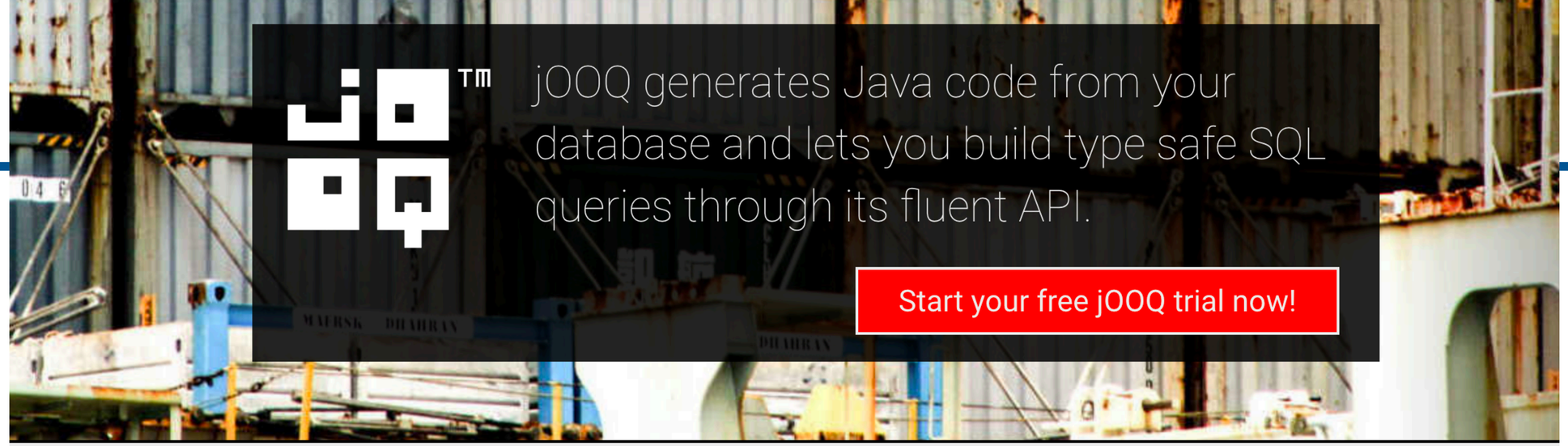
https://github.com/objectify/objectify

Other database representations

- Objectify: gives easy and full access to the Google Cloud Datastore
- Dynamic / Reflexion based
- Alternative to JPA interface

- Objectify surfaces **all native datastore features**, including batch operations, queries, transactions, asynchronous operations, and partial indexes.
- Objectify provides **type-safe key and query classes** using Java generics.
- Objectify provides a **human-friendly query interface**.
- Objectify can automatically **cache your data in memcache** for improved read performance.
- Objectify can store polymorphic entities and perform **true polymorphic queries**.
- Objectify provides a simple, **easy-to-understand transaction model**.
- Objectify provides built-in facilities to **help migrate schema changes** forward.
- Objectify provides **thorough documentation** of concepts as well as use cases.
- Objectify has an **extensive test suite** to prevent regressions.

https://github.com/objectify/objectify



Great Reasons for Using jOOQ

Our customers spend most time on *their* business-logic.
Because jOOQ takes care of all their Java/SQL infrastructure problems.

https://www.jooq.org/

Database First

Tired of ORMs driving your database model?

Whether you design a new application or integrate with your legacy, your database holds your most important asset: your data.

jOOQ is SQL-centric. Your database comes "first".

Typesafe SQL

Fed up with detecting SQL syntax errors in production?

SQL is a highly expressive and type safe language with a rich syntax. jOOQ models SQL as an internal DSL and uses the Java compiler to compile your SQL syntax, metadata and data types.

Code Generation

Bored with renaming table and column names in your Java code?

jOOQ generates Java classes from your database metadata. Your Java compiler will tell you when your code is out of sync with your schema.

Active Records

Annoyed by the amount of SQL you write for CRUD?

jOOQ lets you perform CRUD and POJO mapping directly on Active Records, which are also generated from the code generator.

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Part 3 - JPA Associations)

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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
       @ManyToOne
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @OneToMany
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val fantasy = Category(0, “Fantasy", emptySet<Book>())
categories.save(fantasy)
val lor = Book(0,"Lord of the Rings", fantasy)
books.save(lor)
```

```
call next value for hibernate_sequence
insert into category (name, id) values (?, ?)
binding parameter [1] as [VARCHAR] - [Fantasy]
binding parameter [2] as [BIGINT] - [1]
call next value for hibernate_sequence
insert into book (kind_id, name, id) values (?, ?, ?)
binding parameter [1] as [BIGINT] - [1]
binding parameter [2] as [VARCHAR] - [Lord of the Rings]
binding parameter [3] as [BIGINT] - [2]
```

JPA Associations

```java
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setName(String name) {…}

    @OneToMany(mappedBy = "bookCategory",
               cascade = CascadeType.ALL)
    public Set<Book> getBooks() { … }

    public void setBooks(Set<Book> books) { … }
}
```

```java
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCategory) { … }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }

    public void setName(String name) { … }

    @ManyToOne
    @JoinColumn(name = "book_category_id")
    public BookCategory getBookCategory() { … }

    public void setBookCategory(BookCategory bookCategory) {…}
}
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @ManyToOne
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @OneToMany
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val book = books.findById(lor.id)
logger.info(book.get().kind.toString())
```

```
select book0_.id as id1_0_0_,
       book0_.kind_id as kind_id3_0_0_,
       book0_.name as name2_0_0_,
       category1_.id as id1_1_1_,
       category1_.name as name2_1_1_
from book book0_
left outer join category category1_ on
   book0_.kind_id=category1_.id where book0_.id=?
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @ManyToOne
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @OneToMany
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val kind = categories.findById(fantasy.id)
```

```
select
   category0_.id as id1_1_0_,
   category0_.name as name2_1_0_
from category category0_
where category0_.id=?
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,

        @OneToMany
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @ManyToOne
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val kind = categories.findById(fantasy.id)
logger.info(kind.toString())
```

```
failed to lazily initialize a collection of role:
com.demo.Category.books
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @ManyToOne
        val kind:Category
)

@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        val name:String,
        @OneToMany(fetch = FetchType.EAGER)
        val books:Set<Book>
)
```

```
interface BookRepository : CrudRepository<Book,Long>
interface CategoryRepository: CrudRepository<Category,Long>
…
val kind = categories.findById(fantasy.id)
logger.info(kind.toString())
```

```
Optional[Category(id=1, name=Fantastic, books=[])]
```

```
select
category0_.id as id1_1_0_,
category0_.name as name2_1_0_,
books1_.category_id as category1_2_1_,
book2_.id as books_id2_2_1_,
book2_.id as id1_0_2_,
book2_.kind_id as kind_id3_0_2_,
book2_.name as name2_0_2_,
category3_.id as id1_1_3_,
category3_.name as name2_1_3_
from category category0_
left outer join category_books books1_ on category0_.id=books1_.category_
left outer join book book2_ on books1_.books_id=book2_.id
left outer join category category3_ on book2_.kind_id=category3_.id
where category0_.id=?
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        @Column(nullable = false)
        val name:String,
        @ManyToOne
        val kind:Category
)
@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        @Column(nullable = false)
        val name:String,
        @OneToMany(cascade = arrayOf(CascadeType.ALL), mappedBy = "kind", fetch = FetchType.EAGER)
        var books:List<Book>
)
```

```
val fantasy = Category(0, "Fantasy", emptyList<Book>())
val lor = Book(0,"Lord of the Rings", fantasy)
val silm = Book(0,"Silmarillion", fantasy)
fantasy.books = listOf(lor,silm)
categories.save(fantasy)
```

```
binding parameter [1] as [VARCHAR] - [Fantasy]
binding parameter [2] as [BIGINT] - [1]
insert into book (kind_id, name, id) values (?, ?, ?)
binding parameter [1] as [BIGINT] - [1]
binding parameter [2] as [VARCHAR] - [Lord of the Rings]
binding parameter [3] as [BIGINT] - [2]
insert into book (kind_id, name, id) values (?, ?, ?)
binding parameter [1] as [BIGINT] - [1]
binding parameter [2] as [VARCHAR] - [Silmarillion]
binding parameter [3] as [BIGINT] - [3]
```

```
Category(id=1, name=Fantasy, books=[{ Lord of the Rings, Fantasy }, { Silmarillion, Fantasy }])
```

JPA Associations

```
@Entity
data class Book(
        @Id
        @GeneratedValue
        val id:Long,
        @Column(nullable = false)
        val name:String,
        @ManyToOne
        val kind:Category
)
@Entity
data class Category(
        @Id
        @GeneratedValue
        val id:Long,
        @Column(nullable = false)
        val name:String,
        @OneToMany(cascade = arrayOf(CascadeType.ALL), mappedBy = "kind", fetch = FetchType.EAGER)
        var books:List<Book>
)
```

```
val kind = categories.findById(fantasy.id)
logger.info(kind.toString())
logger.info(kind.get().books.size.toString())
for (b in kind.get().books) {
    logger.info(b.toString())
    logger.info(b.kind.toString())
}
```

```
select
    category0_.id as id1_1_0_,
    category0_.name as name2_1_0_,
    books1_.kind_id as kind_id3_0_1_,
    books1_.id as id1_0_1_,
    books1_.id as id1_0_2_,
    books1_.kind_id as kind_id3_0_2_,
    books1_.name as name2_0_2_
from category category0_
left outer join book books1_ on
    category0_.id=books1_.kind_id where category0_.id=?
```

```
Category(id=1, name=Fantasy, books=[{ Lord of the Rings, Fantasy }, { Silmarillion, Fantasy }])
```

Eager and Lazy

- Evaluation modes - transferring objects to memory

```
@OneToMany(mappedBy = "course", cascade = CascadeType.ALL, fetch = FetchType.EAGER)
private Set<Enrollment> enrollments;
```

- Eager: transfers all objects related to the root

```
@OneToMany(mappedBy = "course", cascade = CascadeType.ALL, fetch = FetchType.Lazy)
private Set<Enrollment> enrollments;
```

- Lazy: transfers object when needed
  - Needs a transactional environment

```
@Component
public class ProfessorService {

    @Autowired
    CourseRepository courseRepository;

    @Autowired
    ProfessorRepository professors;

    @Transactional
    public void addCourses(String name, String... courses) {
        Professor p = professors.findByName(name);

        Set<Course> cs = p.getCourses();
        for(String c: courses)
            cs.add(courseRepository.findByName(c).get(0));
        professors.save(p);
    }
```

Many To Many

```java
@Entity
public class Course {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private long id;

    private String name;

    private int credits;

    @OneToMany(mappedBy = "course",
               cascade = CascadeType.ALL,
               fetch = FetchType.EAGER)
    private Set<Enrollment> enrollments;

    @ManyToMany(mappedBy = "courses")
    private Set<Professor> professors;
}
```

```java
@Entity
public class Professor {

    public Professor(String name) {
        this.setName(name);
    }

    public Professor() {}

    @javax.persistence.Id
     @GeneratedValue(strategy=GenerationType.AUTO)
    private long Id;

    private String name;

    @ManyToMany
    private Set<Course> courses;
}
```

creates *professor_courses* table in a uni-directional relation

https://en.wikibooks.org/wiki/Java_Persistence/ManyToMany

http://www.objectdb.com/java/jpa/getting/started

https://hellokoding.com/jpa-many-to-many-extra-columns-relationship-mapping-example-with-spring-boot-maven-and-mysql/

Internet Applications Design and Implementation 2020 - 2021

(Lecture 4 - Part 4 -Optimization of data fetching)

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))

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

N+1 query problem

```
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setNam

    @OneToMany(mappedBy
               cascade
    public Set<Book> getBooks() { … }

    public void setBoo
}
```

```
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCategory) { … }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }

    @JoinColumn(name = "book_category_id")
    public BookCategory getBookCategory() { … }
```

```
@Repository
public interface BookCategoryRepository extends JpaRepository<BookCategory, Integer>{}
```

```
for( BookCategory c: categoryRepo.findAll() )
  for( Book b: c.books )
    …
```

```
select * from book_category;
select * from book where bookCategory = ?
…
select * from book where bookCategory = ?
```

N+1 query problem

```
@Entity
@Table(name = "book_category")
public class BookCategory {
    private int id;
    private String name;
    private Set<Book> books;

    public BookCategory(){ … }

    public BookCategory(String name) {…}

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() {…}

    public void setName(String name) { }

    public Set<Book> getBooks() { … }

}
```

```
@Repository
public interface BookCategoryRepository extends JpaRepository<BookCategory, Integer>{}
```

```
for( Book b : BookRepo.findAll() )
  System.out.println( b.toString() + b.bookCategory.toString())
```

```
@Entity
public class Book{
    private int id;
    private String name;
    private BookCategory bookCategory;

    public Book() {}

    public Book(String name) { this.name = name;}

    public Book(String name, BookCategory bookCategory) { … }

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() { … }

    public void setId(int id) { … }

    public String getName() { … }

    @JoinColumn(name = "book_category_id")
    public BookCategory getBookCategory() { … }

    setBookCategory(BookCategory bookCategory) {…}
```

```
select * from book;
select * from bookCategory where bookCategory = ?
…
select * from bookCategory where bookCategory = ?
```

Prefetching

- Instead of using global fetching strategies, one can define how objects/ collections related to one particular entity are loaded to memory

```
fun appointmentsOfPet(id: Long): List<AppointmentDAO> {
    val pet = pets.findById(id)
                  .orElseThrow { NotFoundException("There is no Pet with Id $id") }
    return pet.appointments
}
```

Two queries

```
interface PetRepository : JpaRepository<PetDAO, Long> {
    @Query("select p from PetDAO p inner join fetch p.appointments where p.id = :id")
    fun findByIdWithAppointment(id:Long) : Optional<PetDAO>
}
```

```
fun appointmentsOfPet(id: Long): List<AppointmentDAO> {
    val pet = pets.findByIdWithAppointment(id)
                  .orElseThrow { NotFoundException("There is no Pet with Id $id") }
    return pet.appointments
}
```

Appointment's collections is fetched and loaded in one single query

Custom queries for efficient execution

- ORM relations can produce a large number of queries… which can be optimized by means of a single query being dispatched to the DB

```
@Query(" select s from Student s "+
     "   inner fetch join s.courses en "+
     "   inner join en.course c "+
     " where c.name = :name")
List<Student> searchByCourse(@Param("name") String name);
```

- Optimization may work by summarising data

```
@Query("select new ciai.model.StudentSummary(s.name,s.age) from Student s")
List<StudentSummary> summaryStudents();
```

Java Persistence Query Language (JPQL ⊆ HQL)

- Simple custom queries

```
@Query("select s from Student s where s.name like CONCAT(?,'%')")
List<Student> search(String name);
```

- Complex custom queries

```
@Query(" select s from Student s "+
       "   inner join s.courses en "+
       "   inner join en.course c "+
       " where c.name = :name")
List<Student> searchByCourse(@Param("name") String name);
```


http://www.objectdb.com/java/jpa/getting/started

Now it’s time for you to research and experiment…

Internet Applications Design and Implementation 2020 - 2021

(Lab class 4)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

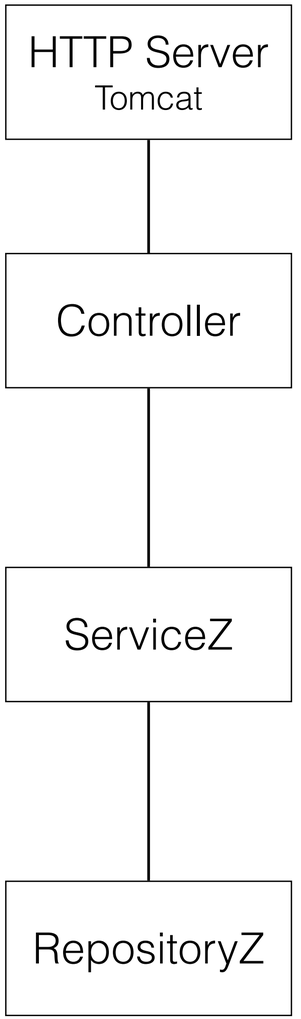
João Costa Seco (joao.seco@fct.unl.pt)

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

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Lab class 4 - Software Architecture using Spring

- Lab Objectives:
  1. Check your group status in the spreadsheet with the link below:
     https://docs.google.com/spreadsheets/d/1CUlXcvlXmU4tH71L2Ox-sGSLuXSDw73AHmorznQNS44/edit?usp=sharing
  2. Create an Architecture with internal services and repositories
  3. Test the architecture for at least one entity using Mock components
  4. Define the data model for resource Application (no relations yet)
  5. Implement the functionality of one resource (Application)

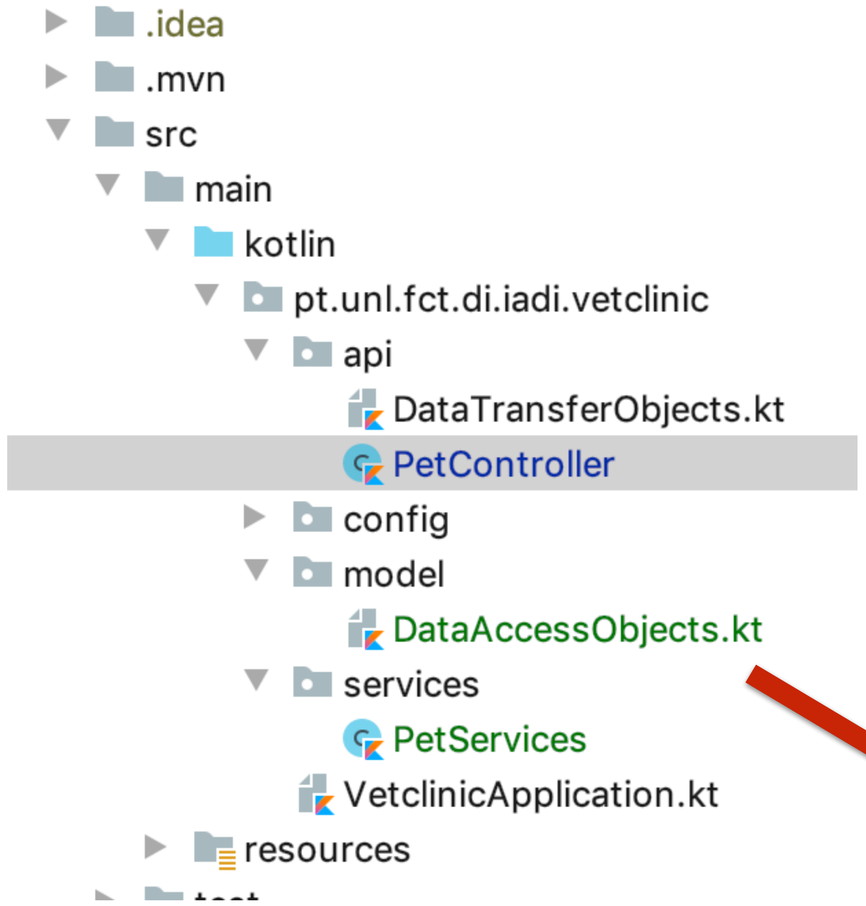


Lab class 4 - Introduction to Spring and Swagger

- Lab Protocol:
  1. Check your group status in the spreadsheet with the link
  2. Make sure that you have the whole API defined and documented with Swagger
  3. Create an internal service class for each top-level resource and wire the calls from controller to service
  4. Create Exceptions for NotFound errors that are thrown by the service
  5. Add JPA Entities for resource Application (ApplicationDAO)
  6. Create tests that separately test the controller and the service for resource Application
  7. Create a repository for Application using database h2 (in memory)
  8. You may want to also create tests for the repository to understand better the data operations
  9. Implement the service using the repository's basic functions.
  10. Create a Repository for Applications and Students
  11. Insert seed data into the database
  12. Produce efficient queries to search Applications by date
  13. Produce a search filter in the "List all applications" endpoint

Lab class 4 - Create internal services

- Create and wire a Service Class, redirect the calls, Make DAO-DTO transformation

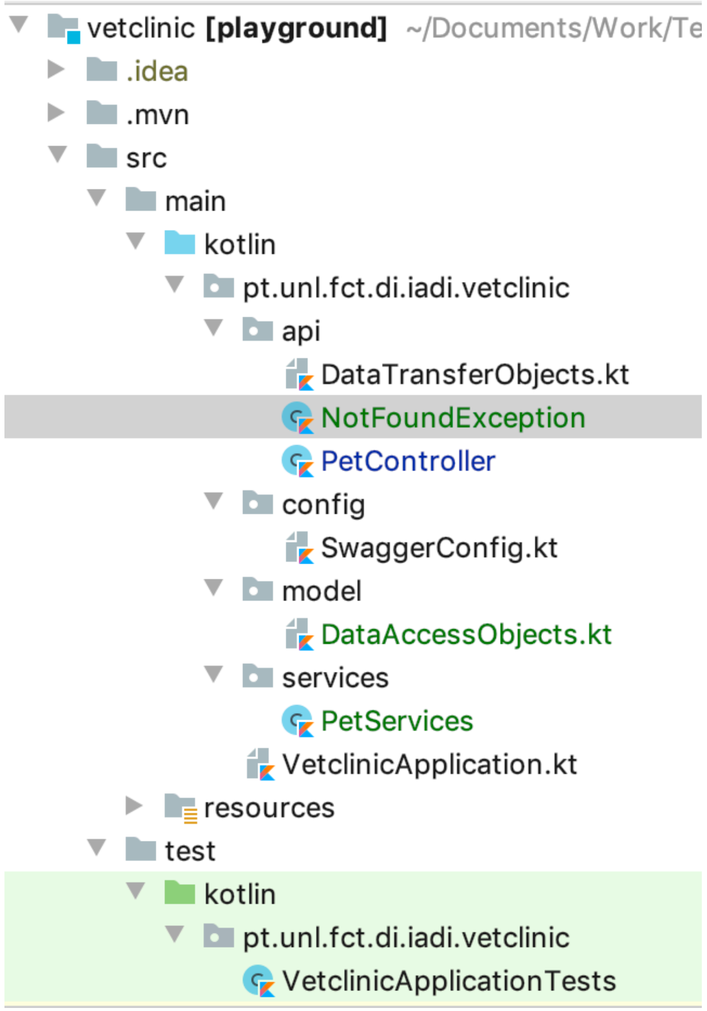



```
@RestController
@RequestMapping( ...value: "/pets")
class PetController(val pets:PetServices) {

    @GetMapping( ...value: "")
    fun getAllPets() =
            pets.getAllPets().map { PetDTO(it.id, it.name, it.species) }

    @PostMapping( ...value: "")
    fun addNewPet(@RequestBody pet:PetDTO) =
            pets.addOnePet(PetDAO(pet.id,pet.name,pet.species))
```

```
data class PetDAO(val id:Long, val name: String, val species: String)
```

Lab class 4 - Exceptions

- Create Exceptions should be used to represent REST Errors

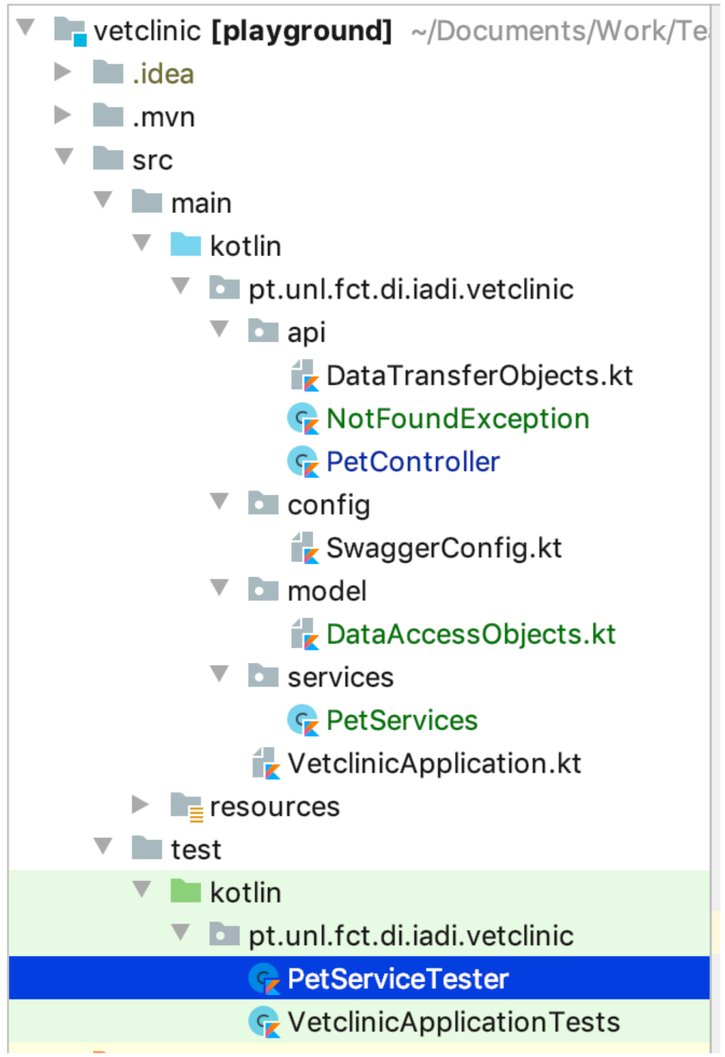



```kotlin
package pt.unl.fct.di.iadi.vetclinic.api

class NotFoundException(msg:String) : RuntimeException(msg)
```

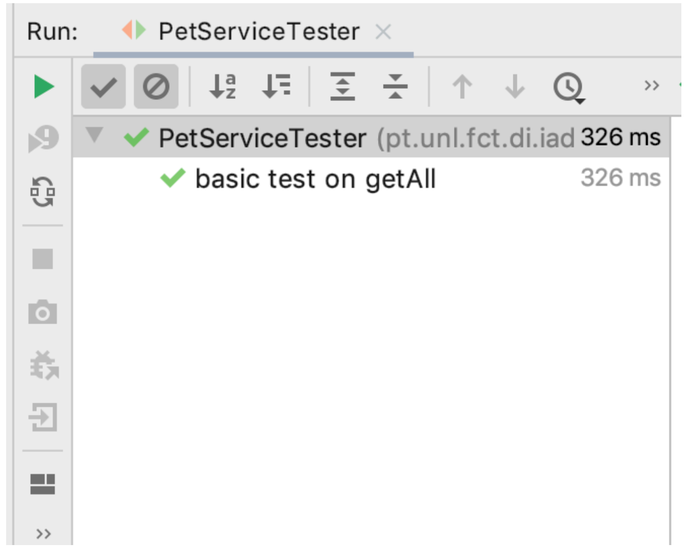
```kotlin
@Service
class PetServices {

    fun getAllPets(): List<PetDAO> = emptyList()

    fun getOnePet(id:Long) =
            if (id == 1L)
                PetDAO(id, name: "Pantufas", species: "Dog")
            else
                throw NotFoundException("Pet $id not found")

    fun addOnePet(pet:PetDAO) {}
}
```

Lab class 4 - Tests with mock objects

- Create a basic test for the service PetServices

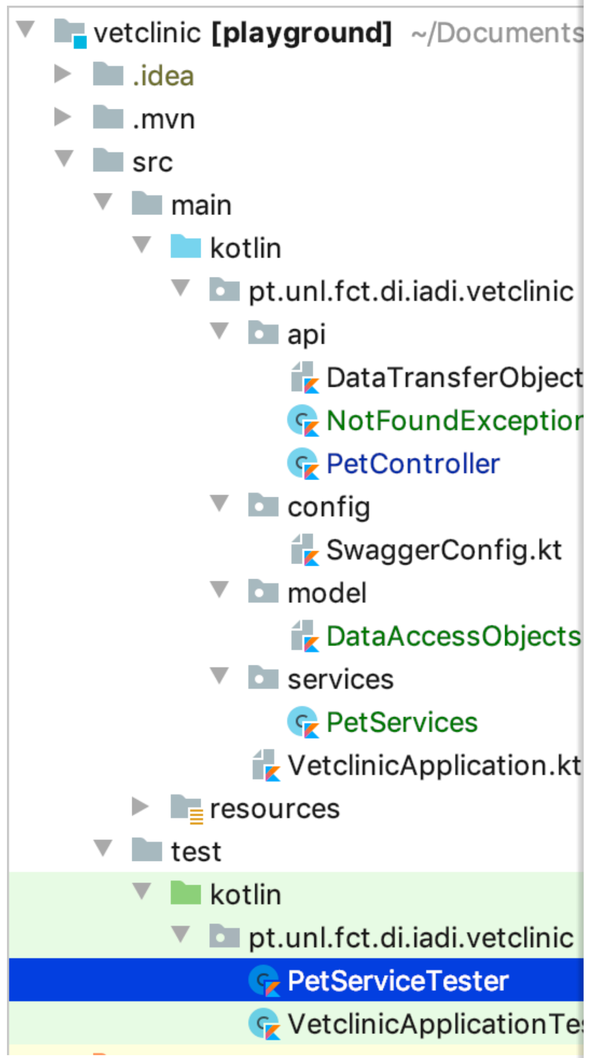



```kotlin
import org.hamcrest.CoreMatchers.equalTo
import org.junit.Assert
import org.junit.Test
import org.junit.runner.RunWith
import org.springframework.beans.factory.annotation.Autowired
import org.springframework.boot.test.context.SpringBootTest
import org.springframework.test.context.junit4.SpringRunner
import pt.unl.fct.di.iadi.vetclinic.services.PetServices

@RunWith(SpringRunner::class)
@SpringBootTest
class PetServiceTester {

    @Autowired
    lateinit var pets:PetServices

    @Test
    fun `basic test on getAll`() {
        Assert.assertThat(pets.getAllPets(), equalTo(emptyList()))
    }
}
```

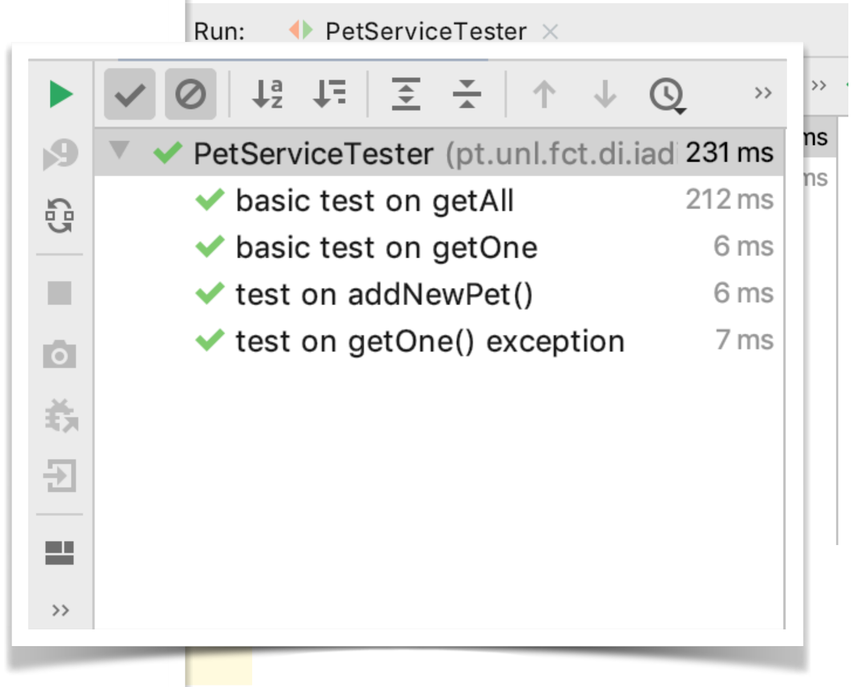


Lab class 4 - Te

- Test all methods

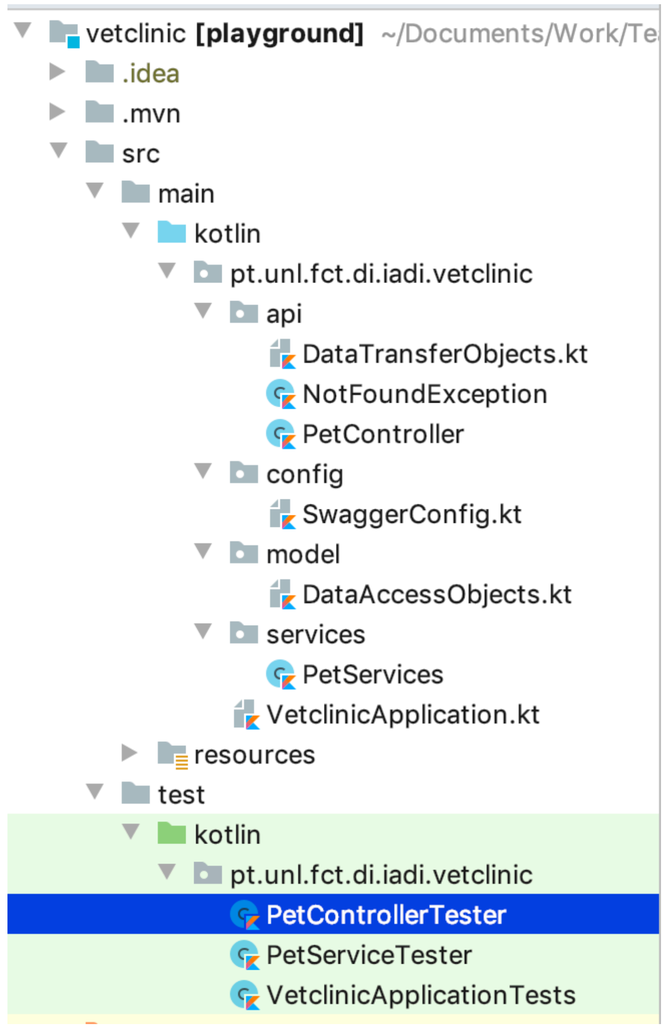



```kotlin
@RunWith(SpringRunner::class)
@SpringBootTest
class PetServiceTester {

    @Autowired
    lateinit var pets:PetServices

    companion object Contants {
        val pantufas = PetDAO( id: 1L,  name: "Pantufas",  species: "Dog")
    }

    @Test fun `basic test on getAll`() {
        assertThat(pets.getAllPets(), equalTo(emptyList()))
    }

    @Test fun `basic test on getOne`() {
        assertThat(pets.getOnePet( id: 1L), equalTo(pantufas))
    }

    @Test(expected = NotFoundException::class)
    fun `test on getOne() exception`() {
        pets.getOnePet( id: 0L)
    }

    @Test
    fun `test on addNewPet()`() {
        pets.addOnePet(pantufas)
        assertThat(pets.getOnePet(pantufas.id), equalTo(pantufas))
    }
}
```

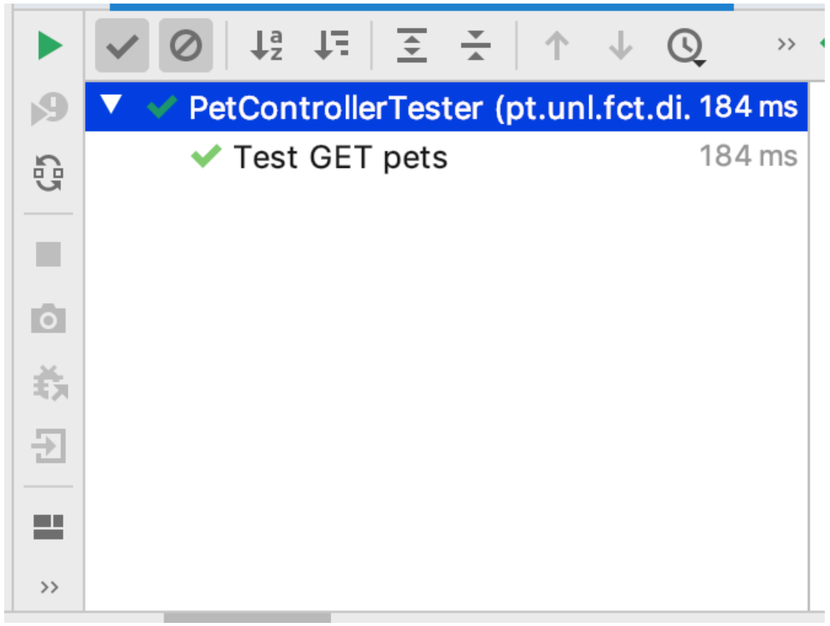


Lab class 4 - Tests with mock objects

- Create a test for controllers mocking the corresponding service

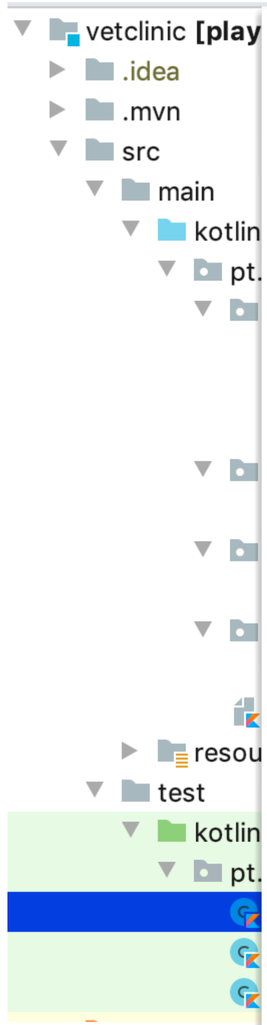



```kotlin
@RunWith(SpringRunner::class)
@SpringBootTest
@AutoConfigureMockMvc
class PetControllerTester {

    @Autowired lateinit var mvc: MockMvc

    @MockBean lateinit var pets:PetServices

    companion object {
        val pantufas = PetDAO( id: 1L,  name: "pantufas",  species: "Dog")
        val bigodes = PetDAO( id: 2L,  name: "bigodes",  species: "Cat")
        val petsDAO = listOf(pantufas, bigodes)

        val petsDTO = petsDAO.map { PetDTO(it.id, it.name, it.species) }

        val petsURL = "/pets"
    }

    @Test fun `Test GET pets`() {
        Mockito.`when`(pets.getAllPets()).thenReturn(petsDAO)

        mvc.perform(get(petsURL))
                .andExpect(status().isOk())
                .andExpect(jsonPath( expression: "$", hasSize<Any>(petsDTO.size)))
    }

}
```



Lab class 4 - Tests with mock objects

- Create a test for controllers mocking the corresponding service




```
@Test fun `Test Get One Pet`() {
    Mockito.`when`(pets.getOnePet( id: 1)).thenReturn(pantufas)

    val result = mvc.perform(get( urlTemplate: "$petsURL/1"))
                    .andExpect(status().isOk)
                    .andReturn()

    val responseString = result.response.contentAsString
    val responseDTO = mapper.readValue<PetDTO>(responseString)
    assertThat(responseDTO, equalTo(petsDTO[0]))
}

@Test fun `Test GET One Pet (Not Found)`() {
    Mockito.`when`(pets.getOnePet( id: 2)).thenThrow(NotFoundException("not found"))

    val result = mvc.perform(get( urlTemplate: "$petsURL/2"))
                    .andExpect(status().is4xxClientError)
}
```

Lab class 4

- Import new Components (Spring-Data & H2)
- Declare DAO @Entity classes
- Declare repositories and implement services

```
import javax.persistence.*

@Entity
data class PetDAO(
        @Id @GeneratedValue val id:Long,
                             val name: String,
                             val species: String
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artif
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
```

```
@Service
class PetServices {

    @Autowired
    lateinit var pets: PetRepository

    fun getAllPets(): MutableIterable<PetDAO> = pets.findAll()

    fun getOnePet(id:Long) : PetDAO {
        val pet = pets.findById(id)
        if (pet.isPresent )
            return pet.get()
        else
            throw NotFoundException("There is no pet with Id ${id}")
    }
```

Lab class 4 - Mock the repository, test the service