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

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 aplɪˈkeɪʃ(ə)n |, noun

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

Mobile application | ˈməʊbʌɪl aplɪˈkeɪʃ(ə)n |, noun

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

Web service | wɛb ˈsəːvɪs |, 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
  - [illegible]
  - [illegible]
  - [illegible]
  - 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.

Introduction To Continuous Delivery

#1 in the Continuous Delivery webinar series

This talk will introduce the principles and practices of Continuous Delivery, an approach pioneered by companies like Facebook, Flickr and ThoughtWorks, that aims to make it possible for an organization to deliver frequently (weekly, daily or even hourly) and confidently. It uses idea -> live (the time from idea being conceived until the feature is live) as a key metric, minimizing that metric across the whole path to production.

By Rolf Russell

  - data and code have different wills, we need to develop software in sync with data.

DEC 11, 2012 @ 11:48 PM 8,338 VIEWS

Gmail Outage Embarrasses Internet Giant -- Cause Was a Software Update

  - 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 ...
}
```

scenari

evolve well)

- work by explicit **configu**

- 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 re configuration)
  - Sometimes by sca
- support **test driv**
- work by explicit **c**

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

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

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

mobile
service
applications
web
techniques
mechanisms
orchestrate
patterns
data
clients
modularity
MVC
Hibernate
abstractions
collaborative
online
practices
persistency
reactive
result
methods
performance
specific
advanced
tools
single
providers
page
pages
architectural
resources
native
software
validation
clients
services
oriented
TDD
Aspects
Web
Beans
Components
compound
development
Internet
Applications
fundamentals
challenges
integration
good
concerns
Model
Spring
REST
Service
Security
Access
Representational
Resourcefulness
Relational
Transactions
Search
Viewstate
Processes
Persistency
Parameters
Multi-language
Sections
Database
NoSQL
WS-BPEL
Ajax
Concurrency
Cookies
User
Control
CMS
Object
Life
Data
JPA
Term
Short
Crawling
Transfer
State
Engines
Analysis
Experience
Cycle
Model
View
Control

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)

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

https://httpstatuses.com/

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

swagger default (/api-docs) Explore

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;
```


© 2015, João Costa Seco, Jácome Cunha, João Leitão

Machine readable specification

Swagger UI

localhost:8080/swagger-ui.html#!/product-controller/listUsingGET

product-controller : Operations pertaining to products in Online Store

Show/Hide | List Operations | Expand Operations

POST /product/add — Add a product

DELETE /product/delete/{id} — Delete a product

GET /product/list — View a list of available products

Response Class (Status 200)

Successfully retrieved list

Model | Example Value

```
{}
```

Response Content Type application/json

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	You are not authorized to view the resource		
403	Accessing the resource you were trying to reach is forbidden		
404	The resource you were trying to reach is not found		

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 $
```

Kotlin References

Kotlin

LEARN COMMUNITY TRY ONLINE

Reference | Tutorials | Books | More resources

- Overview
- Getting Started
- Basics
- Classes and Objects
- Functions and Lambdas
- Collections
- Multiplatform Programming
- Other
- Core Libraries
- Reference
- Java Interop
- JavaScript
- Native
- Coroutines
- Tools
- Evolution
- FAQ

Full Kotlin Reference

Learn Kotlin

Edit Page

All Materials | Getting Started | Migrating from Java

Documentation

Kotlin documentation is a good place to start, check out these links to get your feet wet:

- Basics
- Idioms
- Interop with Java

IDE

Many modern IDEs support Kotlin and help in writing idiomatic Kotlin code:

- Kotlin Educational Plugin
- Java2Kotlin converter

Community

Kotlin community is open, helpful, and welcoming. Don't hesitate to join and ask on any platform you like:

- Blog
- Forum
- Slack
- Stack overflow

Playground

Hands-on experience is the way to master your Kotlin skills on real examples right in the browser:

- Playground
- Kotlin Examples
- Koans

Books

Online Courses

Reference | Tutorials | Books | M

- Overview
- Getting Started
  - Basic Syntax
  - Idioms
  - Coding Conventions
- Basics
- Classes and Objects
- Functions and Lambdas
- Collections
- Multiplatform Programming
- Other
- Core Libraries
- Reference

Idioms

A collection of random and frequently used idioms in Kotlin. If you have a favorite id
sending a pull request.

Creating DTOs (POJOs/POCOs)

```
data class Customer(val name: String, val email: String)
```

provides a `Customer` class with the following functionality:

- getters (and setters in case of `var`s) for all properties
- `equals()`
- `hashCode()`
- `toString()`
- `copy()`
- `component1()`, `component2()`, ..., for all properties (see Data classes)

Lab class 1 - Document the API (Sw

```fortran
/** 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()

}
```

iadi-2019-20-private — -bash — 80×62

```
iadi 2019 $ http :8080/v2/api-docs
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Tue, 24 Sep 2019 10:19:39 GMT
Transfer-Encoding: chunked

{
    "basePath": "/",
    "definitions": {
        "Number": {
            "type": "object"
        },
        "PetDTO": {
            "properties": {
                "id": {
                    "$ref": "#/definitions/Number"
                },
                "name": {
                    "type": "string"
                },
                "species": {
                    "type": "string"
                }
            },
            "required": [
                "id",
                "name",
                "species"
            ],
            "type": "object"
        }
    },
    "host": "localhost:8080",
    "info": {
        "contact": {
            "email": "joao.seco@fct.unl.pt",
            "name": "João Costa Seco",
            "url": "http://ctp.di.fct.unl.pt/~jcs"
        },
        "description": "IADI 2019 VetClinic REST API",
        "license": {
            "name": "Apache 2.0",
            "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
        },
        "title": "Spring Boot REST API Example for IADI 2019/20",
        "version": "1.0.0"
    },
    "paths": {
        "/pets": {
            "get": {
                "consumes": [
                    "application/json"
                ],
                "operationId": "getAllPetsUsingGET",
                "produces": [
                    "*/*"
                ],
                "responses": {
                    "200": {
                        "description": "OK",
                        "schema": {
                            "items": {
```

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 - Documen

```
/** 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,

}
```

Swagger UI

localhost:8080/swagger-ui.html#/

swagger

default (/v2/api-docs)

Explore

Spring Boot REST API Example for IADI 2019/20

IADI 2019 VetClinic REST API

Created by João Costa Seco
See more at http://ctp.di.fct.unl.pt/~jcs
Contact the developer
Apache 2.0

pet-controller : Management operations of Pets in the IADI 2019 Pet Clinic

Show/Hide | List Operations | Expand Operations

GET	/pets	View a list of registered pets
GET	/pets/{id}	Get the details of a single pet by id

[BASE URL: / , API VERSION: 1.0.0]

Lab class 1 - Documen

```
/** 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,

}
```

Swagger UI

localhost:8080/swagger-ui.html#!/pet45controller/getOnePetUsingGET

pet-controller : Management operations of Pets in the IADI 2019 Pet Clinic

Show/Hide | List Operations | Expand Operations

GET /pets — View a list of registered pets

GET /pets/{id} — Get the details of a single pet by id

Response Class (Status 200)

Successfully retrieved pet details

Model | Example Value

```
{
  "id": {},
  "name": "string",
  "species": "string"
}
```

Response Content Type */*

Response Messages

HTTP Status Code	Reason	Response Model	Headers
401	You are not authorized to view the resource		
403	Accessing the resource you were trying to reach is forbidden		
404	The resource you were trying to reach is not found		

Try it out!

[BASE URL: / , API VERSION: 1.0.0]

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/

NORDICAPIS.COM

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

```java
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 ->>

```java
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).

```java
@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

```
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

HIBERNATE

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…

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

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

```kotlin
@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
ong)                                                                        Unit
: Any?)                                                                  Boolean
ong)                                                                     Boolean
                                                  (Mutable)Iterable<PetDAO!>
(Mutable)Iterable<Long!>)                         (Mutable)Iterable<PetDAO!>
ng)                                                     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>

```
CrudRepository (org.springframework.data.repo
JpaRepository (org.springframework.data.jpa.r
JpaRepositoryImplementation (org.springframew
PagingAndSortingRepository (org.springframewo
PetRepository (pt.unl.fct.di.iadi.vetclinic.m
QuerydslJpaRepository (org.springframework.da
ReactiveCrudRepository (org.springframework.d
ReactiveSortingRepository (org.springframewor
RevisionRepository (org.springframework.data.
RxJava2CrudRepository (org.springframework.da
RxJava2SortingRepository (org.springframework
SimpleJpaRepository (org.springframework.data
```

```
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
deleteById(Long)                                                            Unit
equals(other: Any?)                                                      Boolean
existsById(Long)                                                         Boolean
findAll()                                        (Mutable)Iterable<PetDAO!>
findAllById((Mutable)Iterable<Long!>)            (Mutable)Iterable<PetDAO!>
findById(Long)                                          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
```

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> findBy name(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

```kotlin
@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>
)
```

```kotlin
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 bidirectional 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 setName

    @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() { … }

    kCateg
```

```
@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)
Eduardo Geraldo (e.geraldo@campus.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/1CUIXcvIXmU4tH71L2Ox-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

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

Internet Applications Design and Implementation 2020 - 2021

(Lecture 5 - on Security)

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),
contributions from Beatriz Moreira, APDC-INV 2017/2018)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Security of Applications (Overview)

- Security is one of the most hot topics in CS in present days!
- Cryptography can only protect data outside of systems, when properly applied and if there are no “internal” information leaks.
- Security breaches are often caused by crashes, erroneous behaviour due to unexpected inputs, or errors in the code.

Security of Applications (Overview)

- Good engineering practices and correct usage of methods and tools ensures that the specified data confidentiality rules are enforced.
- There are two elective courses in MIEI only about this topic.
  - Software Security: fundamental concepts and technologies
  - Network and Computer Systems Security: system-level security
- This lecture is about models and frameworks that implement software security.

Outline

- Security Concepts Review
- Foundations of Computer Security
- Security Models
- Kotlin and Spring Security
- Model-Based Access Control in Spring

Internet Applications Design and Implementation 2020 - 2021

(Lecture 5 - Part 1 - Security Concepts)

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

Security of Internet Applications

- Layers of internet application security
  - Network Level (network security and system identification)
  - System Level (firewalls, VPNs, SSL, DMZ)
  - Application Level (our focus)
    - Authentication
    - Access control
    - Information flow

Big Picture - Organisation Industry Standards (ISO 27002)

- Definition of General Security policies
- Organization of information security
- Asset management
- Human resources security
- Physical and environmental security
- Communications and operations management
- Access control (data, operations, and other resources)
- Acquisition, development, and maintenance of Software Systems
- Incident management
- Compliance with regulations

About software

Security Policies in Software Systems

- within an software system:
  - “The set of restrictions and properties that specify how a computing system prevents information and computing resources from being used to violate an organizational security policy” in Computer Security, Dieter Gollmann 2011
- define who has permission to access or operate on, a given resource.
  - Access control lists,
  - firewall settings,
  - services that may be run on user devices,
  - security protocols for protecting network traffic,
  - …

Attacks and Attackers

- Security can be defined and challenged based on the definition of attacks and attackers. What does an attacker knows and what operations can they perform on a system.
- The correct functioning of a system is based on assumptions on the environment.
- Attacks are designed to challenge system assumptions with illegal inputs, causing illegal states, erroneous behaviour or crashes.

Firefox Bug Bounty Rewards

On behalf of the Mozilla and the millions of people who visit our sites, use Firefox and our other products we would like to thank them for their hard work in helping to make us more secure.

As of this date, we have paid out almost **$1,000,000** across all of our bounties. Congratulations to everybody who has participicated!

2018

1st Quarter 2018

- Tadj Youssouf
- James Grant
- xisigr
- Root Object
- Jun Kokatsu
- mozillabugs
- Abdulrahman Alqabandi
- Jordi Chancel
- Konark Modi
- Francisco Alonso of NowSecure Research Team
- Zhang Hanming from 360 Vulcan team
- Nils
- Omair
- Rob Wu [:robwu]
- David Dworken

Mozilla Security
Advisories
Known Vulnerabilities
Mozilla Security Blog
Security Bug Bounty

Client Bug Bounty
Frequently Asked Questions
Hall of Fame

Web Bug Bounty
Eligible Websites
Frequently Asked Questions
Hall of Fame

Security flaws in software

Still Up-to-Date

OWASP Top 10 - 2017
A1:2017-Injection
A2:2017-Broken Authentication
A3:2017-Sensitive Data Exposure
A4:2017-XML External Entities (XXE) [NEW]
A5:2017-Broken Access Control [Merged]
A6:2017-Security Misconfiguration
A7:2017-Cross-Site Scripting (XSS)
A8:2017-Insecure Deserialization [NEW, Community]
A9:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging&Monitoring [NEW,Comm.]

https://www.owasp.org/

Classic Example: SQL injection

Classic Example: SQL injection

User Name:

" or ""="

Password:

" or ""="

```
uName = getRequestString("username");
uPass = getRequestString("userpassword");

sql = 'SELECT * FROM Users WHERE Name ="'
+ uName + '" AND Pass ="' + uPass + '"'
```

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

https://www.w3schools.com/sql/sql_injection.asp

In a nutshell

Most common attacks are at the application level and can be avoided by properly using well tested and tested frameworks and well established methods.

Internet Applications Design and Implementation 2020 - 2021

(Lecture 5 - Part 2 - Foundations)

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

Foundations of Computer Security

- Fundamental Properties:
  - **confidentiality** - prevention of unauthorised disclosure of information,
  - **integrity** - prevention of unauthorised modification of information,
  - **availability** - prevention of unauthorised withholding of information or resources
  - … more… accountability; non-repudiation; reliability/dependability; …

"Computer security deals with the prevention and detection of unauthorized actions by users of a computer system" - Dieter Gollmann,

Foundations of Software Security

- Main concepts used in software security
  - **Principal** (the active subject)
  - **Resourse** (the passive object)
  - **Authentication** - the principal provides proof that they are who they say they are
  - **Authorisation** - the principal is trusted to access or perform an operation on a resource
  - **Delegation** - can you operate on behalf of other principal?

- Declassification operations in confidentiality analysis
- Endorsement operations in integrity analysis

Principal/Authentication

- An authenticated entity (or group) that uses the system
- The active subject on system operations
- **Authentication**:
  - The certification process of determining the identity of an external entity that uses the system, invokes actions, produces and consumes data.

Resource/Authorisation

- The process of checking what resources a principal is allowed to access and manipulate, and what operations is it allowed to execute at a given time and system state (data, parameters, context, etc.).
- **Authorisation** (to observe or change)
  - read, execute, write, append, delete, change access, change ownership, …
  - Classic access control models
    - Access Control Matrix
    - Capabilities
    - Access Control Lists

Authorisation and access control models

- Base models
  - Access Control Matrix; Capabilities; Access Control Lists
- Intermediate control structures
  - Groups and negative permissions

Authorisation and access control models

- Base models
  - Access Control Matrix; Capabilities; Access Control Lists
- Intermediate control structures
  - Role-based access control
    - Principal
    - Roles
    - Permissions
    - Operations
    - Resources

Internet Applications Design and Implementation 2020 - 2021

(Lecture 5 - Part 3 - Security Models)

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

Security Models

- Access Control List
- Capability-based access control
- Role-based access control
- Bell–LaPadula model for confidentiality
- Biba Integrity model for data integrity
- Model-based access control

Access Control Lists

- ACLs are lists of permissions (identity, operation) attached to objects stored within the system.
- It must explicitly assign individual identities to operations on resources.
- Spring Security ACL implements ACL on top of explicitly managed database tables

```
@PostFilter("hasPermission(filterObject, 'READ')")
List<NoticeMessage> findAll();

@PostAuthorize("hasPermission(returnObject, 'READ')")
NoticeMessage findById(Integer id);

@PreAuthorize("hasPermission(#noticeMessage, 'WRITE')")
NoticeMessage save(@Param("noticeMessage")NoticeMessage noticeMessage);
```

https://www.baeldung.com/spring-security-acl

Capability-based access control

- A decentralised method that provides a scalable approach to access control
  - No need for centralised access control lists and mechanisms;
- A capability refers to an object and a set of rights, the user that owns it can perform the operation described in the capability
  - Example: a pair (“/etc/pass”, O_RDWR) gives readwrite access
- Common in micro-service based architectures and IoT scenarios

Capability-based access control

- A capability is transmitted through unforgeable (and ephemeral) tokens
- Capabilities can be explicitly and dynamically managed
  - Good support for authorisation and delegation
- To perform an operation on a particular object, the principal has first to acquire the associated capability via some dynamic authorisation method.

Role-Base Access Control

- Standard approach to authorise users to perform operations in software systems.
  - Roles are specified for system related tasks
  - Permissions are assigned to specific roles
  - Users are dynamically assigned roles according to their profile/function

https://upload.wikimedia.org/wikipedia/en/c/c3/RBAC.jpg

Role-Base Access Control

- Standard approach to authorise users to perform operations in software systems.
  - Roles are specified for system related tasks
  - Permissions are assigned to specific roles
  - Users are dynam
    according to thei

- **Roles can also be combined in hierarchies or lattices**
- **Delegation can be integrated in the general model**
- **RBAC provides a framework for a verification based on interceptors and filters**
- **Roles can also be parameterised and depend on actual data.**

https://upload.wikimedia.org/wikipedia/en/c/c3/RBAC.jpg

Bell–LaPadula model - Confidentiality

- Formalises the U.S. Department of Defense multilevel security policies.
- The security level of a subject is compared to the classification of the object (and the security compartment where it is stored)
- The access control rules state that:
  - A subject cannot read up
  - A subject cannot write down
  - + an access control matrix

TOP SECRET

SECRET

CONFIDENTIAL

PUBLIC

No Write Down

No Read Up

Information Flow Control - Confidentiality

- The process of ensuring that data is only seen by trusted principals. This involves assigning a level of trust to data and principals.
- General technique to check it is called **information flow control**
- **Declassification**: The process of lowering the level of trust needed to access a given data item.

```
var high = 1
var low = … 10 + high …

var low = 2
if ( high > 0 ) low else 3
```

Operating Systems

R.S. Gaines Editor

A Lattice Model of Secure Information Flow

Dorothy E. Denning
Purdue University

Biba Integrity model - Data Integrity

- Data integrity has three goals:
  - Prevent data modification by unauthorised principals
  - Prevent unauthorised data modification by authorised principals
  - Maintain internal and external data consistency
- The access control rules state that:
  - A subject cannot read up
  - A subject cannot write down
  - + an access control matrix

No Read Down

No Write Up

Information Flow Control - Data Integrity

- The process of ensuring that data is only provided by trusted principals.
- Also an information flow control property (dual of confidentiality)
- **Endorsement**: the process of increasing the level of trust o a given data item.

```
var untrusted = 1
var trusted = … 10 + untrusted …

var trusted = 2
if ( untrusted > 0 ) trusted else 3
```

Operating Systems

R.S. Gaines
Editor

A Lattice Model of Secure Information Flow

Dorothy E. Denning
Purdue University

Model-Based Access Control

- Security conditions are many times hidden in query filters and program conditions. These policies are very difficult to get right, maintain, and modify.
- Developer-defined roles usually depend on the current state of the application. (e.g. ModeratorOf(…)). Role-based models usually ignore this and require the application logic to validate it.
- Example: authorisations usually depend on the state of the target entity (status == “submitted”)
- Capabilities defined by the developer can extend the standard read/write/delete (not hardwired to the basic programming elements)

Type-based Access Control in Data-Centric Systems

Luís Caires[1], Jorge A. Pérez[1], João C. Seco[1], Hugo T. Vieira[1], and Lúcio Ferrão[2]

[1] CITI and Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa
[2] OutSystems SA

Data-centric multi-user systems, such as web applications, require flexible yet fine-grained data security mechanisms. Such mechanisms are usually enforced by a specially crafted security layer, which adds extra complexity and often leads to error prone coding, easily causing severe security breaches. In this paper, we introduce a programming language approach for enforcing access control policies to data in data-centric programs by static typing. Our development is based on the general concept of refinement type, but extended so as to address realistic and challenging scenarios of permission-based data security, in which policies dynamically depend on the database state, and flexible

Model-Based Access Control

(19) **United States**

(12) **Patent Application Publication**

Ferrão et al.

(10) **Pub. No.: US 2013/0246995 A1**

(43) **Pub. Date: Sep. 19, 2013**

(54) **SYSTEMS, METHODS, AND APPARATUS FOR MODEL-BASED SECURITY CONTROL**

(75) Inventors: **Lúcio Emanuel Represas Ferrão**, Lisboa (PT); **João Ricardo Viegas da Costa Seco**, Cascais (PT); **Luis Manuel Marques da Costa Caires**, Lisboa (PT); **Gonçalo Filipe Xavier Caleira Borrêga**, Alcabideche (PT); **António Augusto Vieira Melo**, Lisboa (PT)

(73) Assignee: **OutSystems - Software em Rede S.A.**, Linda-A-Velha (PT)

(21) Appl. No.: **13/418,922**

(22) Filed: **Mar. 13, 2012**

Publication Classification

(51) **Int. Cl.**
G06F 9/44 (2006.01)
G06F 9/45 (2006.01)

(52) **U.S. Cl.**
USPC **717/104**

(57) **ABSTRACT**

An integrated model-driven application development and execution environment enables declaration of a data-role in an application model. The data-role is based on a property of a data entity in the application model. The data-role provides for the enforcement of domain-specific security policies with respect to data elements corresponding to the data entity.

Further Reading

Computer Security. Dieter Gollmann. 3rd edition. Wiley, 2011.
OWASP Testing Guide V4. Owasp foundation, 2016. https://www.owasp.org/images/1/19/OTGv4.pdf

– CIA triad: confidentiality, integrity, and availability (A1, A6)
– Authentication, authorization, non-repudiation (A1, A2, A6)
– Risk, threats, vulnerabilities, and attack vectors (A3, A6)
– Concept of trust and trustworthiness (A4)
– Threat and attacker modelling (A3, A5, B1)

– Identification and authentication (A2, B3)
– Authorization and access control models (B2, B4, B5)
– Defensive programming (C2, D1)
– Software security testing (A5, C2, D2)

– Secure design basic principles (B2, B4, C1)
– Security best practices and standards (B3, B4, B5, B6, C2)
– Techniques for preserving security across modules and trust maintenance (A4, B1, B6, D1, D2, D3)

– Web security model (B6)
– Session management, authentication (B3, C3)
– Web application vulnerabilities and defenses (A5, B1, C3, C4)
– Client-side security (C4, D3, D4)
– Server-side security tools (C4, D3, D5)

Internet Applications Design and Implementation 2020 - 2021

(Lecture 5 - Part 4 - Using Kotlin & Spring)

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

Using frameworks for security

- Frameworks such as Spring, with Spring Security project promote the reuse of (correct) code, good practices, and great number of base features.

Spring Security 5.4.1

OVERVIEW LEARN

Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements

https://spring.io/projects/spring-security

Using frameworks for security

- Frameworks such as Spring Security promote the reuse of (correct) code, good practices, and great number of base features.

  - HTTP BASIC authentication headers (an IETF RFC-based standard)
  - HTTP Digest authentication headers (an IETF RFC-based standard)
  - HTTP X.509 client certificate exchange (an IETF RFC-based standard)
  - LDAP (a very common approach to cross-platform authentication needs, especially in large environments)
  - Form-based authentication (for simple user interface needs)
  - OpenID authentication
  - Authentication based on pre-established request headers (such as Computer Associates Siteminder)
  - JA-SIG Central Authentication Service (otherwise known as CAS, which is a popular open source single sign-on system)
  - Transparent authentication context propagation for Remote Method Invocation (RMI) and HttpInvoker (a Spring remoting protocol)
  - Automatic “remember me" authentication (so you can tick a box to avoid re-authentication for a predetermined period of time)
  - Anonymous authentication (allowing every unauthenticated call to automatically assume a particular security identity)
  - Run-as authentication (which is useful if one call should proceed with a different security identity)
  - Java Authentication and Authorization Service (JAAS)
  - JEE container autentication (so you can still use Container Managed Authentication if desired)
  - Kerberos

Including Spring Security

- By just including the dependency in the application, security is automatically enabled.

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

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

- Configuration is then possible by declaring basic access properties and basic user information.

https://spring.io/guides/gs/securing-web/

Including Spring Security

- By just including the dependenc
  enabled.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifa
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

- Configuration is then possible by declaring basic access properties and basic user information.

https://spring.io/guides/gs/securing-web/
https://spring.io/guides/topicals/spring-security-architecture

Including Spring Security

- By just including the dependency i
  enabled.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactI
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

- Configuration is then possible by declaring basic access properties and basic user information.

```
@Configuration
@EnableWebSecurity
class WebSecurityConfig : WebSecurityConfigurerAdapter() {
    @Throws(Exception::class)
    override fun configure(http: HttpSecurity) {
        http
        .authorizeRequests()
            .antMatchers("/" ).permitAll()
            .anyRequest().authenticated()
            .and()
        .formLogin()
            .permitAll()
            .and()
        .logout()
            .permitAll()
    }

    @Bean
    public override fun userDetailsService(): UserDetailsService {
        val user: UserDetails = User.withDefaultPasswordEncoder()
                .username("user")
                .password("password")
                .roles("USER")
                .build()
        return InMemoryUserDetailsManager(user)
    }
}
```

Including Spring Security

- By just including the dependency i
  enabled.

```
<dependency>
  <groupId>org.springframework.b
  <artifactId>spring-boot-starte
</dependency>

<dependency>
  <groupId>org.springframework.s
  <artifactId>spring-security-te
  <scope>test</scope>
</dependency>
```

- Configuration is ther
  by declaring basic a
  properties and basic user
  information.

```
@Configuration
@EnableWebSecurity
class WebSecurityConfig : WebSecurityConfigurerAdapter() {

                                    pSecurity) {

                                    tAll()
                                    ted()

                                    ervice(): UserDetailsService {
                                    .withDefaultPasswordEncoder()

                                    )

                                    nager(user)
    }
}
```

Including Spring Security

- By just including the dependen
  enabled.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</arti
</dependency>

<dependency>
  <groupId>org.springframework.security</groupIc
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

- Configuration is then possible by declaring basic access properties and basic user information.

```
@Configuration
@EnableWebSecurity
class WebSecurityConfig : WebSecurityConfigurerAdapter() {
    override fun configure(http: HttpSecurity) {
        http
        .csrf().disable()
        .authorizeRequests()
            .antMatchers("/" ).permitAll()
            .anyRequest().authenticated()
        .and().httpBasic()
    }

    override fun configure(auth: AuthenticationManagerBuilder) {
        auth.inMemoryAuthentication()
                .withUser("user")
                .password(BCryptPasswordEncoder().encode("password"))
                .authorities(emptyList())
            .and()
                .passwordEncoder(BCryptPasswordEncoder())
    }
}
```

https://spring.io/guides/gs/securing-web/

Including Spring Security

- By just includin
  enabled.

```
<dependency>
  <groupId>org.springframe
  <artifactId>spring-boot-
</dependency>

<dependency>
  <groupId>org.springframe
  <artifactId>spring-secu
  <scope>test</scope>
</dependency>
```

- Configuration is
  by declaring ba
  properties and
  information.

jcs — -zsh — 80×24

```
jcs@Joaos-iMac ~ % http :8080/applications
HTTP/1.1 403
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Connection: keep-alive
Content-Type: application/json
Date: Mon, 19 Oct 2020 14:30:34 GMT
Expires: 0
Keep-Alive: timeout=60
Pragma: no-cache
Set-Cookie: JSESSIONID=2454FD5814FB3EEBF80E07CF1737C10D; Path=/; HttpOnly
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

{
    "error": "Forbidden",
    "message": "",
    "path": "/applications",
    "status": 403,
    "timestamp": "2020-10-19T14:30:34.051+00:00"
}

jcs@Joaos-iMac ~ %
```

```
pter() {

agerBuilder) {

encode("password"))

oder())
```

https://spring.io/guides/gs/securing-web/

Including Spring Security

- By just including
  enabled.

```
<dependency>
  <groupId>org.springframe
  <artifactId>spring-boot-
</dependency>

<dependency>
  <groupId>org.springframe
  <artifactId>spring-secur
  <scope>test</scope>
</dependency>
```

- Configuration is
  by declaring bas
  properties and b
  information.

jcs — -zsh — 80×24

```
jcs@Joaos-iMac ~ % http :8080/applications --auth user:password
HTTP/1.1 200
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Connection: keep-alive
Content-Type: application/json
Date: Mon, 19 Oct 2020 14:57:40 GMT
Expires: 0
Keep-Alive: timeout=60
Pragma: no-cache
Set-Cookie: JSESSIONID=2361CEB5951A39C2967863030D2EBA48; Path=/; HttpOnly
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

[]

jcs@Joaos-iMac ~ %
```

```
ter() {

gerBuilder) {

ncode("password"))

der())
```

https://spring.io/guides/gs/securing-web/

Authentication - Simple starter code

- The default security setting provides that all requests are protected, a login form is created, RESTful interface for login/logout…

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

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

Look into the documentation for the up-to-date API!!

Fundamental Concepts mapped to Spring Context

- Principal - Who is the entity behind a particular request

```
@RequestMapping("/messages/inbox")
public ModelAndView findMessagesForUser(@AuthenticationPrincipal CustomUser user) {

    // .. find messages for this user and return them ...
}
```

- Authentication: Certify that a given set of credentials identify one principal

```
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
    auth
        .jdbcAuthentication()
            .dataSource(dataSource)
            .withDefaultSchema()
            .withUser("user").password("password").roles("USER").and()
            .withUser("admin").password("password").roles("USER", "ADMIN");
}
```

Look into the documentation for the up-to-date API!!

Linking to the application model

- Dynamic verification of user credentials is part of the application model

```
override fun configure(auth: AuthenticationManagerBuilder) {
    auth.inMemoryAuthentication()
            .withUser( username: "user")
            .password(BCryptPasswordEncoder().encode( rawPassword: "password"))
            .authorities(emptyList())
            .and()
            .passwordEncoder(BCryptPasswordEncoder())
            .and()
            .userDetailsService(customUserDetails)
}
```

Look into the documentation for the up-to-date API!!

Linking to the application model

- Dynamic verification of user credentials is part of the application model

```
override fun configure(auth: AuthenticationManagerBuilder) {
    auth.inMemoryAuthentication()
            .withUser( username: "
            .password(BCryptPassw
            .authorities(emptyLi
            .and()
            .passwordEncoder(BCry
            .and()
            .userDetailsService(
}
```

```
@Service
class CustomUserDetailsService(
        val users: UsersService
) : UserDetailsService {

    override fun loadUserByUsername(username: String?): UserDetails {

        username?.let { it: String
            val userDAO = users.findUser(it)
            if( userDAO.isPresent ) {
                return CustomUserDetails(userDAO.get().username, userDAO.get().password, mutableListOf())
            } else
                throw UsernameNotFoundException(username)
        }
        throw UsernameNotFoundException(username)
    }
}
```

Look into the documentation for the up-to-date API!!

Security of Internet Applications

- Authorisation - Check if a given principal has rights to access a given piece of information

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .antMatchers("/resources/**", "/signup", "/about").permitAll()
            .antMatchers("/admin/**").hasRole("ADMIN")
            .antMatchers("/db/**").access("hasRole('ROLE_ADMIN') and hasRole('ROLE_DBA')")
            .anyRequest().authenticated()
            .and()
        // ...
        .formLogin();
}
```

- Delegation - to be able to temporarily assign one principals capabilities' to another principal (e.g. sudo)

org.springframework.security.web.authentication.switchuser

Class SwitchUserFilter

java.lang.Object
    org.springframework.web.filter.GenericFilterBean
        org.springframework.security.web.authentication.switchuser.SwitchUserFilte

All Implemented Interfaces:

javax.servlet.Filter, Aware, BeanNameAware, DisposableBean, InitializingBean, Ap

http://docs.spring.io/autorepo/docs/spring-security/3.2.1.RELEASE/apidocs/org/springframework/security/web/authentication/switchuser/SwitchUserFilter.html

Basic Spring Security Guaranties

There really isn't much to this configuration, but it does a lot. You can find a summary of the features below:

- Require authentication to every URL in your application
- Generate a login form for you
- Allow the user with the **Username** *user* and the **Password** *password* to authenticate with form based authentication
- Allow the user to logout
- CSRF attack prevention
- Session Fixation protection
- Security Header integration
  - HTTP Strict Transport Security for secure requests
  - X-Content-Type-Options integration
  - Cache Control (can be overridden later by your application to allow caching of your static resources)

From Spring Security docs

Basic Spring Security Guaranties

- Security Header integration
  - HTTP Strict Transport Security for secure requests
  - X-Content-Type-Options integration
  - Cache Control (can be overridden later by your application to allow caching of your static resources)
  - X-XSS-Protection integration
  - X-Frame-Options integration to help prevent Clickjacking
- Integrate with the following Servlet API methods
  - HttpServletRequest#getRemoteUser()
  - HttpServletRequest.html#getUserPrincipal()
  - HttpServletRequest.html#isUserInRole(java.lang.String)
  - HttpServletRequest.html#login(java.lang.String, java.lang.String)
  - HttpServletRequest.html#logout()

From Spring Security docs

CSRF - token issued by the server / recognised by the same server

6. Cross Site Request Forgery (CSRF)

This section discusses Spring Security's Cross Site Request Forgery (CSRF) support.

6.1. CSRF Attacks

Before we discuss how Spring Security can protect applications from CSRF attacks, we will explain what a CSRF attack is. Let's take a look at a concrete example to get a better understanding.

Assume that your bank's website provides a form that allows transferring money from the currently logged in user to another bank account. For example, the HTTP request might look like:

From Spring Security docs

```
POST /transfer HTTP/1.1
Host: bank.example.com
Cookie: JSESSIONID=randomid; Domain=bank.example.com; Secure; HttpOnly
Content-Type: application/x-www-form-urlencoded

amount=100.00&routingNumber=1234&account=9876
```

CSRF - token issued by the server / recognised by the same server

- Template example…

```
<div th:if="${#httpServletRequest.remoteUser}!=null">
  <label>User: </label><span th:text="${#httpServletRequest.remoteUser}"></span>
  <form style="display:inline-block" th:action="@{/logout}" method="post">
      <input type="submit" value="Sign Out"/>
  </form>
</div>

<div th:if="${#httpServletRequest.remoteUser} == null" >
  <form th:action="@{/}" method="post">
      <label> User : <input type="text" name="username"/> </label>
      <label> Password: <input type="password" name="password"/> </label>
      <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
      <input type="submit" value="Sign In"/>
  </form>

</div>
```

CSRF - token issued by the server / recognised by the same server

- Actual page example

```
▼ <div class="container">
  ▼ <div>
    ▼ <form method="post" action="/">
      ▼ <label>
          " User : "
        ▶ <input type="text" name="username">
        </label>
      ▼ <label>
          " Password: "
        ▶ <input type="password" name="password">
        </label>
      ▶ <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}">
      ▶ <input type="submit" value="Sign In">
      ▶ <input type="hidden" name="_csrf" value="f067a25f-c6f5-4de0-b6ad-ea7416986b22">
      </form>
    </div>
  </div>
</div>
```

Internet Applications Design and Implementation 2020 - 2021

(Lecture 5 - Part 5 - Model-based Authorisation in Spring)

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

Model Based Access Control

- Security conditions are many times hidden in query filters and program conditions. They are very difficult to maintain/modify/get right.
- Developer defined roles usually depend on the state of the application. (e.g. ModeratorOf(…)). Role-based models ignore this.
- Authorisations usually depend on the state of the target entity (status == submitted)
- Capabilities defined by the developer extend read/write/delete (not hardwired to the basic programming elements)

Model Based Access Control

Model Based Access Control - Spring?

```
protected void configure(HttpSecurity http) throws Exception {
    http
        .authorizeRequests()
            .anyRequest().authenticated()
            .and()
            .antMatchers("/admin/hotels/{id}")
            .access(id => managerOfHotel(id))
            .access(id => hotels.findOne(id).isApproved())
//…
}
```

Not really Spring… how could we do this?

Security and Frameworks (Spring)

- Web Development Frameworks provide basic mechanisms like:
  - Authentication
    - username, password, authorities (roles)
  - Authorization
    - hasRole, hasPermission

Role Based Access Control

- Application level pattern implemented on top of basic authentication mechanisms
  - Principal (The authenticated user)
  - Role (Job functions a user may have)
  - Capabilities (Systems operations a role may be associated with)
- Allows the dynamic assignment of capabilities to users
- Allows sophisticated assignment and delegation mechanisms (e.g. roles assigned in name of another principal, or by a limited time).

Model based security

- Security conditions are part of the application level
- Authentication and authorisation are system level

```
// POST /hotels/{id}        - update the hotel with identifier {id}
@RequestMapping(value="{id}", method=RequestMethod.POST)
@PreAuthorize("hasRole('{ROLE_ADMIN,ROLE_MANAGER}')")
public String editSave(@PathVariable("id") long id, Hotel hotel,
                       Model model, Principal principal) {
    if( principal.getName().equals(hotel.getManager().getName())) {
        hotel.setTimestamp(new Date());
        hotels.save(hotel);
        return "redirect:/hotels/"+id;
    } else
        throw new AccessDeniedException("Cannot edit hotel "+hotel.getName());
}
```

- Security conditions are many times encoded into application logic (if-else-throw statements) and where conditions in queries.

Spring Security implementation

- Basic spring security is based on SPEL expressions using principal information and parameters…

```
// POST /hotels/{id}          - update the hotel with identifier {id}
@RequestMapping(value="{id}", method=RequestMethod.POST)
@PreAuthorize("hasRole({'ROLE_ADMIN','ROLE_MANAGER'}) "
            + "and principal.name == #hotel.manager.name")
public String editSave(@PathVariable("id") long id, Hotel hotel,
                       Model model, Principal principal) {
    hotel.setTimestamp(new Date());
    hotels.save(hotel);
    return "redirect:/hotels/"+id;
}
```

- Rule: “a hotel can be edited by its manager or an administrator”
  - Problem: `#hotel.manager.name` is not loaded !!

Pros & Cons

- + May use model information
- + Provides row-level authorisation on resources
- - Has limited access to the model (database)
- - Bad reuse of condition expressions
- - Runtime parsing and execution of security expressions

Policies or security rules

- Isolate the implementation of security rules in services with full access to the model.

```
@Component("mySecurityService")
public class MySecurityService {

    @Autowired
    HotelRepository hotels;

    public boolean canEditHotel(User user, long hotelId) {

        Hotel hotel = hotels.findOne(hotelId);

        return hotel != null && hotel.getManager() != null &&
                user.getUsername().equals(hotel.getManager().getName());
    }
}
```

- Model access, Good reuse, auditing of rules.

Policies or security rules in “services”

- Isolate the implementation of security rules in services with full access to the model.

```
// POST /hotels/{id}        - update the hotel with identifier {id}
@RequestMapping(value="{id}", method=RequestMethod.POST)
@PreAuthorize("hasRole({'ROLE_ADMIN'}) or "
            + "@mySecurityService.canEditHotel(principal,#id)")
public String editSave(@PathVariable("id") long id, Hotel hotel,
                       Model model, Principal principal) {
      hotel.setTimestamp(new Date());
      hotels.save(hotel);
      return "redirect:/hotels/"+id;
}
```

- Model access, Good reuse, auditing of rules.

Capabilities encoded in “annotations”

- Statically identify capabilities and associated rules in annotations, associated to controllers…

```
// POST /hotels/{id}          - update the hotel with identifier {id}
@RequestMapping(value="{id}", method=RequestMethod.POST)
@AllowedForEditHotel
public String editSave(@PathVariable("id") long id, Hotel hotel, Model model) {
    hotel.setTimestamp(new Date());
    hotels.save(hotel);
    return "redirect:/hotels/"+id;
}
```

Capabilities encoded in “annotations”

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
@PreAuthorize(AllowedForHotelCreation.Condition)
public @interface AllowedForHotelCreation {
    String Condition = "hasRole('ROLE_ADMIN')";
}
```

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
@PreAuthorize(AllowedForEditHotel.Condition)
public @interface AllowedForEditHotel {
    String Condition = "@mySecurityService.canEditHotel(principal,#id) or "
                       + AllowedForHotelCreation.Condition;
}
```

http://blog.novoj.net/2012/03/27/combining-custom-annotations-for-securing-methods-with-spring-security/

View authorisations

- Authorization expressions can be also used in views (with an akward syntax, yes…)

```
<div class="container">
    <h1>Hotel <span th:text="${hotel.name}"></span></h1>
    <p th:text="${hotel.timestamp}"></p>

    <h2>Rooms</h2>
    <ul th:each="room : ${rooms}">
    <li th:text="${room.name}+ ' ' + ${room.type.description}"></li>
    </ul>

    <div sec:authorize="${#authorization.expression(T(booking.security.AllowedForEditHotel).Condition2)}">
        <p><a th:href="@{|/hotels/${hotel.id}/edit|}">Edit</a></p>
    </div>
    <div sec:authorize="${#authorization.expression(T(booking.security.AllowedForHotelCreation).Condition)}">
        <p><a href="/hotels/new">Add a new hotel</a></p>
    </div>
    <p><a href="/hotels">List all hotels</a></p>
</div>
```

- Binding is sometimes different and has to be adapted :(

```
String Condition = "@mySecurityService.canEditHotel(principal,#id) or "
                   + AllowedForHotelCreation.Condition;
String Condition2 = "@mySecurityService.canEditHotel(principal,#vars.hotel.id) or "
                   + AllowedForHotelCreation.Condition;
```

Pros & Cons

- + May use model information
- + Provides row-level authorisation on resources
- + Has full access to the model (database)
- + Good reuse of condition expressions (in services)
- - Runtime parsing and execution of security expressions
  (only better using static verification of code)

Further Reading

Computer Security. Dieter Gollmann. 3rd edition. Wiley, 2011.

OWASP Testing Guide V4. Owasp foundation, 2016.
https://www.owasp.org/images/1/19/OTGv4.pdf

Blogs:

https://heimdalsecurity.com/blog/best-internet-security-blogs/

https://www.schneier.com

OWASP
Open Web Application Security Project

Internet Applications Design and Implementation 2020 - 2021

(Lecture 6 - Session and Token based Security)

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

Overview: HTTP Authentication (Basic and Digest)

- Credentials (username/password) are repeated on each request
  - All requests are vulnerable to attacks (instead of only the login request)
  - Basic: username/password are passed in clear text and can be captured
  - Digest: digests can also be captured and guessed by brute force attacks

- Kind of ok under HTTPS, but…
  - Must have a centralised authority to control and manage principal capabilities
  - Does not support “logout” mechanisms (credentials are always valid)

Overview: Sessions to implement security

- Stateless APIs are good
- but not so good for:
  - ephemeral or distributed authentication,
  - capability based authorisation models
  - protocol management,
  - user preferences (in webapps)
  - …
- Hence, let's implement session management

Outline

- Sessions and cookies
- Token based authentication
- JSON Web Token (JWT)
- OAuth2

Internet Applications Design and Implementation 2020 - 2021

(Lecture 6 - Part 1 - Sessions and Cookies)

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

Basic support for sessions, HTTP cookies

- Basic support to represent stateful information **on the client side**
- Designed to allow websites to remember stateful information about a session
- Shopping carts, authentication info, browser activity, search criteria, etc.
- Source of many security vulnerabilities, attacks and tracking of user activity
- Managed by client and server alike
  - Basically a string managed as a key/value store, can contain cyphered values

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session
https://developer.mozilla.org/en-US/docs/Web/API/Document/cookie

Cookies can be read and written in Spring

- In Spring the annotation @CookieValue is used to retrieve a value from the HTTP cookie and map the value to a parameter.

```
@GetMapping("/applications/")
fun getAll(
    @CookieValue (value="filter", defaultValue = "") filter:String
): List<ApplicationDTO> =
    applications.getAll(filter).map { ApplicationDTO(it) }
```

- Without a declared default value, an exception will be thrown (`java.lang.IllegalStateException`) if the cookie in the request does not contain the key "`filter`".

https://dzone.com/articles/how-to-use-cookies-in-spring-boot

Cookies can be read and written in Spring

- To set the value of a cookie in SpringBoot, object **`HttpServletResponse`** must be added a new cookie value.

```
@PostMapping("/sudents/{id}/applications")
@ResponseStatus(HttpStatus.CREATED)
fun create2(@PathVariable student_id:String,
            @RequestBody @Valid anApplication:ApplicationDTO,
            response: HttpServletResponse) {
    val student = students.get(student_id)
    val id = applications.create(anApplication.toDAO(student))
    val cookie = Cookie("current_application", "id");
    response.addCookie(cookie);
}
```

- Disclaimer: this is just a sample on how to use cookies in Spring not a recommendation that you should do so…

https://dzone.com/articles/how-to-use-cookies-in-spring-boot

Sessions

- To have necessary stateful information in a stateless world
- Basic support to represent stateful information **on the server side**
- A session is a sequence of network HTTP requests and responses associated to the same principal. A session creates the opportunity to create a common context to the set of interactions between parties.
- From the first interaction, a session ID (or token) is established, even for anonymous users.
- This session token and/or identifier is used on the server side for a number of purposes.

https://www.ietf.org/rfc/rfc2616.txt

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

https://developer.mozilla.org/en-US/docs/Web/HTTP/Session

Session management in Spring

- Spring controls how a session is created and how Spring Security will handle it.
  - **always:** a session will always be created if one doesn't already exist
  - **ifRequired:** a session will be created only if required (default)
  - **never:** the framework will never create a session itself but it will use one if it already exists
  - **stateless:** no session will be created or used by Spring Security

```
override fun configure(http: HttpSecurity) {
    http.csrf().disable()
        .authorizeRequests()
        .anyRequest().authenticated()
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
}
```

https://www.baeldung.com/spring-security-session

https://docs.spring.io/spring-security/site/docs/5.4.1-SNAPSHOT/reference/html5/

Session management in Spring

- Spring controls how it.
  - **always:** a session will
  - **ifRequired:** a session
  - **never:** the framework
  - **stateless:** no session

```
override fun configure(h
    http.csrf().disable(
        .authorizeReques
        .anyRequest().au
        .and().sessionMa                                IRED)
}
```

jrcs — -zsh — 80×24

```
jrcs@JoaoCosSecosMac ~ % http :8080/applications --auth admin:pass
HTTP/1.1 200
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Connection: keep-alive
Content-Type: application/json
Date: Sun, 25 Oct 2020 13:46:23 GMT
Expires: 0
Keep-Alive: timeout=60
Pragma: no-cache
Set-Cookie: JSESSIONID=CBC98BBDE7E04C24FB1B0311C6C34254; Path=/; HttpOnly
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

[]

jrcs@JoaoCosSecosMac ~ %
```

https://www.baeldung.com/spring-security-session

https://docs.spring.io/spring-security/site/docs/5.4.1-SNAPSHOT/reference/html5/

Session management in Spring

- Spring controls how a session is created and how Spring Security will handle it.
  - **always:** a session will always be created if one doesn't already exist
  - **ifRequired:** a session will be created only if required (default)
  - **never:** the framework will never create a session itself but it will use one if it already exists
  - **stateless:** no session will be created or used by Spring Security

```
override fun configure(http: HttpSecurity) {
    http.csrf().disable()
        .authorizeRequests()
        .anyRequest().authenticated()
        .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
}
```

https://www.baeldung.com/spring-security-session

https://docs.spring.io/spring-security/site/docs/5.4.1-SNAPSHOT/reference/html5/

Session management in Spring

- Spring controls how it.
  - **always:** a session will
  - **ifRequired:** a session
  - **never:** the framework
  - **stateless:** no session

```
override fun configure(ht
    http.csrf().disable()
        .authorizeRequest
        .anyRequest().aut
        .and().sessionMan                                    S)
}
```

jrcs — -zsh — 80×24

```
jrcs@JoaoCosSecosMac ~ % http :8080/applications --auth admin:pass
HTTP/1.1 200
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Connection: keep-alive
Content-Type: application/json
Date: Sun, 25 Oct 2020 13:49:38 GMT
Expires: 0
Keep-Alive: timeout=60
Pragma: no-cache
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

[]

jrcs@JoaoCosSecosMac ~ %
```

https://www.baeldung.com/spring-security-session

https://docs.spring.io/spring-security/site/docs/5.4.1-SNAPSHOT/reference/html5/

Session management

- Spring Security installs a filter that handles sessions in the Security Context (**`SecurityContextPersistenceFilter`**).
- The session can be managed using the bean (**`HttpSessionSecurityContextRepository)`** that uses HTTP Session as storage.
- For the STATELESS attribute (**`NullSecurityContextRepository`**) is used
- Sessions can be made persistent automatically by means of jdbc, or redis

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

Spring Security Filters

WebAsyncManagerIntegrationFilter

SecurityContextPersistenceFilter

HeaderWriterFilter

LogoutFilter

RequestCacheAwareFilter

SecurityContextHolderAwareRequestFilter

AnonymousAuthenticationFilter

SessionManagementFilter

ExceptionTranslationFilter

FilterSecurityInterceptor

https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html

Spring Security Filters

WebAsyncManagerIntegrationFilter

SecurityContextPersistenceFilter

He

Reques

SecurityContex

Anonymo

Sessio

ExceptionTranslationFilter

FilterSecurityInterceptor

```
public void doFilter(
   ServletRequest req,
   ServletResponse res,
   FilterChain chain) throws IOException, ServletException {

   ...
      chain.doFilter(req,res)
   ...
}
```

Working With the Session

- Spring declares and handles session automatically through beans. A bean with scope "session" is created when the session is first created and linked to the lifecycle of the HTTPSession object.

```
@Component
@Scope("session", proxyMode = ScopedProxyMode.TARGET_CLASS)
class SessionInfo(var numberOfGets:Int = 0, var numberOfPosts:Int = 0)
```

- This bean can then be used in other beans, for instance, a controller.

```
@RestController
class ApplicationController(val applications: ApplicationService): ApplicationAPI {

    @Autowired lateinit var info: SessionInfo;

    override fun getAll(): List<ApplicationDTO> {
       print(info.numberOfGets++)
       return applications.getAll().map { ApplicationDTO(it) }
    }
…
```

https://www.baeldung.com/spring-security-session

Working With the Session

- Spring declares and handles session automatically through beans. A bean with scope "session" is created when the session is first created and linked to the lifecycle of the HTTPSession object.

```
@Component
@Scope("session", proxyMode = ScopedProxyMode.TARGET_CLASS)
class SessionInfo(var numberOfGets:Int = 0, var numberOfPosts:Int = 0)
```

- This bean can then be used in other beans, for instance, a controller.

```
@RestController
class ApplicationController(val applications: ApplicationService): ApplicationAPI {

    @Autowired lateinit var info: SessionInfo;

    override fun getAll(session:HttpSession): List<ApplicationDTO> {
        var info = session.getAttribute(SessionInfo) as SessionInfo;
        session.setAttribute(SessionInfo, SessionInfo(info.numberOfGets+1, info.numberOfPosts))
        return applications.getAll().map { ApplicationDTO(it) }
    }
```

https://www.baeldung.com/spring-security-session

Internet Applications Design and Implementation 2020 - 2021

(Lecture 6 - Part 2 - Token-based Authentication - JWT)

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

HTTP Authentication modes

- Basic Authentication
  - username/password in the header of requests using Base64 encoding
- Digest Authentication
  - has of username/password in the header of requests (MD5 hashing with nonce)
- OAuth Token-based authentication and JWT
  - signed bearer token that allows interactions between independent authorisation and resource servers

https://spring.io/guides/tutorials/spring-boot-oauth2/

Internet Engineering Task Force (IETF) — M. Jones, Microsoft
Request for Comments: 7519 — J. Bradley, Ping Identity
Category: Standards Track — N. Sakimura, NRI
ISSN: 2070-1721 — May 2015

JSON Web Token (JWT)

Abstract

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7519.

Token-based authentication

- Adds a level of indirection and avoids repeated username/password validation (avoids password discovery attacks on basic authentication mode)
- Allows users to access and manipulate a given resource without using username/password
- More benefits:
  - robust authentication solution for repeated requests
  - allows custom limited session duration (limited trust)
  - quickly transfer (user) information between systems (micro services)
  - allows the customisation of roles assigned to a given user at a given time
  - single sign-on in federated systems
  - external authorization servers (google, facebook, github, etc)
  - can be stored in local storage/cookies, can be invalidated or customized

Token-based authentication

- (Good list of benefits, unknown author, link)
- **Cross-domain / CORS:** cookies + CORS don't play well across different domains. A token-based approach allows you to make AJAX calls to any server, on any domain because you use an HTTP header to transmit the user information.
- **Stateless (a.k.a. Server side scalability):** there is no need to keep a session store, the token is a self-contained entity that conveys all the user information. The rest of the state lives in cookies or local storage on the client side.
- **CDN**: you can serve all the assets of your app from a CDN (e.g. javascript, HTML, images, etc.), and your server side is just the API.
- **Decoupling**: you are not tied to any particular authentication scheme. The token might be generated anywhere, hence your API can be called from anywhere with a single way of authenticating those calls.
- **Mobile ready**: when you start working on a native platform (iOS, Android, Windows 8, etc.) cookies are not ideal when consuming a token-based approach simplifies this a lot.
- **CSRF**: since you are not relying on cookies, you don't need to protect against cross site requests (e.g. it would not be possible to sib your site, generate a POST request and re-use the existing authentication cookie because there will be none).
- **Performance**: we are not presenting any hard perf benchmarks here, but a network roundtrip (e.g. finding a session on database) is likely to take more time than calculating an HMACSHA256 to validate a token and parsing its contents.

Token Based Authentication

- Tokens can be managed manually in Spring using cookies (client-side) and sessions (server-side).
- The integration in Spring Security is performed by adding a filter in the Security Filter Chain that intercepts and overrides the authentication attempts.
- A standardised way of providing authentication is Bearer Authentication, where a token is inserted in the “Authorization Header”.

https://tools.ietf.org/html/rfc6750

Spring Security Filters

WebAsyncManagerIntegrationFilter

SecurityContextPersistenceFilter

HeaderWriterFilter

LogoutFilter

UserPasswordAuthenticationFilterToJWT

RequestCacheAwareFilter

SecurityContextHolderAwareRequestFilter

AnonymousAuthenticationFilter

SessionManagementFilter

ExceptionTranslationFilter

FilterSecurityInterceptor

Token Creation - Login

```
class UserPasswordAuthenticationFilterToJWT (
        defaultFilterProcessesUrl: String?,
        private val anAuthenticationManager: AuthenticationManager
) : AbstractAuthenticationProcessingFilter(defaultFilterProcessesUrl) {

    override fun attemptAuthentication(request: HttpServletRequest?,
                                       response: HttpServletResponse?): Authentication? {
        //getting user from request body
        val user = ObjectMapper().readValue(request!!.inputStream, UserDAO::class.java)

        // perform the "normal" authentication
        val auth = anAuthenticationManager.authenticate(UsernamePasswordAuthenticationToken(user.username, user.password))

        return if (auth.isAuthenticated) {
            // Proceed with an authenticated user
            SecurityContextHolder.getContext().authentication = auth
            auth
        } else
            null
    }

    override fun successfulAuthentication(request: HttpServletRequest,
                                          response: HttpServletResponse,
                                          filterChain: FilterChain?,
                                          auth: Authentication) {

        // When returning from the Filter loop, add the token to the response
        addResponseToken(auth, response)
    }
```

https://auth0.com/blog/implementing-jwt-authentication-on-spring-boot/

Token Creation - Login

```kotlin
object JWTSecret {
    private const val passphrase = "este é um grande segredo que tem que ser mantido escondido"
    val KEY: String = Base64.getEncoder().encodeToString(passphrase.toByteArray())
    const val SUBJECT = "JSON Web Token for CIAI 2019/20"
    const val VALIDITY = 1000 * 60 * 60 * 10 // 10 minutes in microseconds
}

private fun addResponseToken(authentication: Authentication, response: HttpServletResponse) {

    val claims = HashMap<String, Any?>()
    claims["username"] = authentication.name

    val token = Jwts
            .builder()
            .setClaims(claims)
            .setSubject(JWTSecret.SUBJECT)
            .setIssuedAt(Date(System.currentTimeMillis()))
            .setExpiration(Date( date: System.currentTimeMillis() + JWTSecret.VALIDITY))
            .signWith(SignatureAlgorithm.HS256, JWTSecret.KEY)
            .compact()

    response.addHeader("Authorization", "Bearer $token")
}
```

```
class UserPasswordAuthenticationFilterToJWT (
        defaultFilterProcessesUrl: String?,
        private val anAuthenticationManager: AuthenticationManager
```

iadi-2019-20-private — -bash — 70×23

```
$ http POST :8080/login username=user password=password
HTTP/1.1 200
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKU09OIFdlYiBUb2tlbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0MjI0LCJpYXQiOjE1NzE3NDgyMjQsInVzZXJuYW1lIjoidXNlciJ9.MqIv5EUab1HjD1vST5LfkUObvHsY0MyEHFt7-KDVoZ4
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Content-Length: 0
Date: Tue, 22 Oct 2019 12:43:44 GMT
Expires: 0
Pragma: no-cache
Set-Cookie: JSESSIONID=B7AED89D85B4BBB68257666D32E51E26; Path=/; HttpOnly
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

$
```

```
r.password))
}
```

JSON Web Token (JWT)

- Base64, signed token that asserts claims about a session/user
- Customisable claims (can carry user information, roles, dates)
- Can include ciphered information also, e.g. user capabilities
- Bearer
  eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKU09OIFdlYiBUb2tlbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0MjI0LCJpYXQiOjE1NzE3NDgyMjQsInVzZXJuYW1lIjoidXNlciJ9.MqIv5EUab1HjD1vST5LfkUObvHsY0MyEHFt7-KDVoZ4

  headerB64.payloadB64.SigHS256

ALGORITHM HS256

Encoded PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKU09OIFdlYiBUb2tlbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0MjI0LCJpYXQiOjE1NzE3NDgyMjQsInVzZXJuYW1lIjoidXNlciJ9.MqIv5EUab1HjD1vST5LfkUObvHsY0MyEHFt7-KDVoZ4

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "sub": "JSON Web Token for CIAI 2019/20",
  "exp": 1571784224,
  "iat": 1571748224,
  "username": "user"
}
```

Tue Oct 22 2019 13:43:44 GMT+0100 (Western European Summer Time)

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
```

Token Validation

```
class JWTAuthenticationFilter: GenericFilterBean() {

    // To try it out, go to https://jwt.io to generate custom tokens, in this case we only need a name...

    override fun doFilter(request: ServletRequest?,
                          response: ServletResponse?,
                          chain: FilterChain?) {

        val authHeader = (request as HttpServletRequest).getHeader("Authorization")

        if( authHeader != null && authHeader.startsWith("Bearer ") ) {
            val token = authHeader.substring(7) // Skip 7 characters for "Bearer "
            val claims = Jwts.parser().setSigningKey(JWTSecret.KEY).parseClaimsJws(token).body

            // should check for token validity here (e.g. expiration date, session in db, etc.)
            val exp = (claims["exp"] as Int).toLong()
            if ( exp < System.currentTimeMillis()/1000) // in seconds

                (response as HttpServletResponse).sendError(HttpServletResponse.SC_UNAUTHORIZED) // RFC 6750 3.1

            else {

                val authentication = UserAuthToken(claims["username"] as String)
                // Can go to the database to get the actual user information (e.g. authorities)

                SecurityContextHolder.getContext().authentication = authentication

                // Renew token with extended time here. (before doFilter)
                addResponseToken(authentication, response as HttpServletResponse)

                chain!!.doFilter(request, response)
            }
        } else {
            chain!!.doFilter(request, response)
        }
    }
}
```

https://bitbucket.org/costaseco/iadi-2019-20/src/master/

Token Validation

```
class UserPasswordAuthenticationFilterToJWT (
        defaultFilterProcessesUrl: String?,
) : Ab
    ov
    }
    ov
        // When returning from the Filter loop, add the token to the response
        addResponseToken(auth, response)
    }
```

```
iadi-2019-20-private — -bash — 70×24
$ http :8080/pets Authorization:"Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiO
iJKU09OIFdlYiBUb2tlbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0Nzg1LCJ
pYXQiOjE1NzE3NDg3ODUsInVzZXJuYW1lIjoidXNlciJ9.hBenpmApZMcEOalI4p-UKIy5
9FSe0-19Fw987He7HGg"
HTTP/1.1 200
Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJKU09OIFdlYiBUb2t
lbiBmb3IgQ0lBSSAyMDE5LzIwIiwiZXhwIjoxNTcxNzg0ODM3LCJpYXQiOjE1NzE3NDg4M
zcsInVzZXJuYW1lIjoidXNlciJ9.cPog74fYmoFirCYvyOR_HeJ3DyYPRbUPEqHUiVbfqh
Q
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Content-Type: application/json;charset=UTF-8
Date: Tue, 22 Oct 2019 12:53:57 GMT
Expires: 0
Pragma: no-cache
Set-Cookie: JSESSIONID=D11358DC1A75490FCAF2C314DF5413EA; Path=/; HttpO
nly
Transfer-Encoding: chunked
X-Content-Type-Options: nosniff
X-Frame-Options: DENY
X-XSS-Protection: 1; mode=block

[]

$
```

Internet Applications Design and Implementation 2020 - 2021

(Lecture 6 - Part 3 -OAuth)

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

OAuth 2.0

- Provides a protocol for authorisation for Internet applications, resource owners, through third-parties on behalf of a principal.
- OAuth defines four roles (from the RFC):
  - **resource owner:** An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
  - **resource server:** The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.
  - **client:** An application making protected resource requests on behalf of the resource owner and with its authorisation.
  - **authorization server:** The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorisation.

https://tools.ietf.org/html/rfc6749

Auth 2.0 protocol flow

```
+--------+                                +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

Token-based interactions

Using spring…

- Spring Boot provides an implementation for oAuth 2.0 that is easy to configure by loading a single module:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

- and configuring two properties

```
spring.security.oauth2.client.registration.github.client-id = client-id
spring.security.oauth2.client.registration.github.client-secret = client-secret
```

https://spring.io/guides/tutorials/spring-boot-oauth2/

The network interactions

1. A request is captured by a filter in Spring Security with no token
2. It is redirected to :
   `http://localhost:8080/oauth2/authorize/github?redirect_uri=<TheURIOfYourApp>`
3. The user is redirected to the AuthorizationUrl of GitHub
4. When authorised, it is redirected to:
   `http://localhost:8080/oauth2/callback/github`
5. A JWT token is created and the user is redirected to the TheURIOfYourApp

Internet Applications Design and Implementation

2020 - 2021

(Lecture 7: Client application development, HTML/CSS/JS)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outline

- User-centric development vs Data-centric development
- The underlying technology: HTML5 (the browser as a virtual machine)
- Client-side frameworks
- Client-side programming languages

Internet Applications Design and Implementation

2020 - 2021

(Lecture 7- Part 1: User Centric Development)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

User-centric vs Data-centric development

- The decoupling between client applications and a data-based service architecture promotes that:
  - in user-centric development:
    - the main focus is on user interactions
    - the design method works around user stories, whose combination provides a full-fledged application.
  - in data-centric development:
    - the main focus is on the representation of state of a systems' information.
    - Interfaces and operations are based on data properties and state changes.

…It's much easier mentally to set up a context around "I am building an API for a database" and "I am building a front-end for users"… Jeff Escalante, carrot

Spectrum of client applications

- **website / static HTML pages**
  - no dynamically loaded data, poor behaviour expression, efficient response times (w/ caching), dynamic websites can be expanded for more efficiency.
- **website / dynamic server rendered HTML pages**
  - poor loading times, poor development features (distance between code and result). Poor modularity, high usage of templating languages.
- **website / ajax**
  - better loading times, development closer to the result (no templating needed). Better modularity and testing conditions.
- **web applications (SPA)**
  - app context, service based back-ends, richer behaviour and fluid UI
- **<u>progressive</u> web applications (PWA)**
  - device agnostic, and yet, closer to native client applications (storage, resources, responsive UI, notifications, etc.)
- **mobile applications**
  - more device resources, many times PWA in a shell

THE NETFLIX TECH BLOG

Netflix TechBlog
Learn about Netflix's world class engineering efforts, company culture, product developments and...

Follow

1.3K

Optimize Page Load

The previous version of the home page rendered **all** rows of titles as its highest priority. This included fetching data from the server, creating all DOM nodes, and loading images. Our goal was to enable fast vertical scrolling through 30+ rows of titles.

For the new experience, we needed our page to load faster, minimize memory overhead, and allow for smooth playback. We knew these performance optimizations would come with a tradeoff against existing member behavior. To understand these tradeoffs, we began by measuring the existing home page load.

When the home page loads, we first render the billboard image and the top three rows on the server. Once this page has been delivered to the client, we make a call for the rest of the homepage, render the rest of the rows, and load all the images.

mic websites can be

, high usage of templating

testing conditions.

fications, etc.)

THE NETFLIX TECH BLOG

Netflix TechBlog
Learn about Netflix's world class engineering efforts, company culture, product developments and...

Follow

1.3K

Optimize Page Load

The previous version of the hom
highest priority. This included fe
DOM nodes, and loading image
scrolling through 30+ rows of ti

For the new experience, we nee
memory overhead, and allow fo
performance optimizations wou
member behavior. To understan
the existing home page load.

When the home page loads, we
three rows on the server. Once t
make a call for the rest of the ho
load all the images.

THE NETFLIX TECH BLOG

Netflix TechBlog
Learn about Netflix's world class engineering efforts, company culture, product developments and...

Follow

1.3K

Page load from the server vs. page load after auto-loading all rows

Here's what the page load looks like from a data perspective.

	Initial Page State	Full Page Load
DOM Nodes	650	11,744
Memory	8MB	45MB
# Images	46	298
Images (Total Size)	1.2MB	4.4MB

Client (web / mobile) Applications

- Running on a Browser
  - Cross-platform compatibility (HTML, CSS, Javascript)
  - Slow interpreted code
  - Limited capabilities (no camera, no gyro, no resources, sandboxed)

- Native code
  - All device capabilities available
  - Fast and efficient applications
  - Proprietary languages and APIs (Java/Android ADT, Swift, C#/C++ UWP)

- Running on a shell (PhoneGap, ReactNative, Electron, etc.)
  - Cross-platform compatibility (HTML, CSS, Javascript)
  - All device capabilities available
  - Precompiled and packaged code (fast)
  - Look and feel is almost right on latest approaches

Internet Applications Design and Implementation

2020 - 2021

(Lecture 7 - Part 2: HTML5)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Web client architecture - Virtual Machine

- 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)

Browser logic architecture

- Source files with static content: HTML / mime-types
- DOM abstract representation (dynamic)
  - Elements
  - Style annotations
  - Event handlers
- User-interface (visible elements)
- JavaScript behaviour

What's HTML5

- Adds meaningful semantic markup
- Separates design from content
- Promotes accessibility and design responsiveness
- Reduces overlap between HTML, CSS, and JavaScript
- Supports rich media contents without the need for plugins (Flash, Java)

https://html.com/html5/

HTML5 comes as a package

- HTML5 is divided into:
  - HTML code to define structure and basic content to web pages
    - new tags like `<video>` `<audio>` `<svg>` `<header>`
  - CSS (Cascading Style Sheets) to define the appearance, layout, and some behaviour
  - JavaScript defines behaviour associated to the webpage elements (including dynamic construction of pages)
- The supporting structure for all these elements is the DOM

What's HTML5 - in more detail

- Deprecated elements like `center`, `font`, and `strike` have been dropped
- Improved parsing rules allow for more flexible parsing and compatibility
- New elements including `video`, `time`, `nav`, `section`, `progress`, `meter`, `aside` and `canvas`
- New input attributes including email, URL, dates and times
- New attributes including charset, async and ping
- New APIs that offer offline caching, drag-and-drop support and more
- Support for vector graphics (SVG) without the aid of programs like Silverlight, Flash, or Java
- Support for MathML to allow better display of mathematical notations
- JavaScript can now run in the background thanks to the JS Web worker API
- Global attributes such as tabindex, repeat and id can be can be applied for all elements

https://www.keycdn.com/blog/html-vs-html5

DOM - Document Object Model

- Standard (W3C) specification of the API of the data structure representing a structured document (HTML, XML, etc.).
- Defines objects and properties for all elements of a page and methods to access and modify them.
- Allows the dynamic retrieval, constructions, and modification of HTML elements in a web page.

http://www.w3schools.com/htmldom

DOM - Document Object Model

- DOM is a convention to represent HTML (XML) documents in a tree of objects.
- It's the **dynamic** supporting structure of a web page.
- Each node of the DOM has a particular structure (attributes) and associated behaviour (methods).
- DOM objects can be created, accessed, and modified using:
  - HTML: the static structure of DOM objects
  - CSS: using (rich) object selectors
  - JavaScript: by traversing the DOM tree using the DOM API.

HTML5 is an assembly-like technology

- The underlying technology is HTML, CSS, JS
- Developing tools should ensure:
  - A higher level of abstraction
  - Tools to develop faster, safer, and more secure

- A simple and robust base to build an UI
- Electron and ReactNative are building a (portable) bridge to native platforms.

HTML

HTML

- Each HTML element has a tag
  - **<body> <ul> <li> ** …
- Content
  - **This is a paragraph**
- and a set of attributes:
  - Generic: set on any HTML element. (ex: **id**, **class**, **hidden**, …)
  - Particular: img/**src**, option/**value**, etc.
  - Events: dependent on the kind of element
    body / **onload**, button / **onclick**, etc.

HTML essentials in one slide

- <!DOCTYPE html>
- <html></html> root element
- <head></head> section to declare meta-information, stylesheets and scripts
- <body></body> section to define content
- <p> <h1>… <ul> block elements
-   <i> inline elements
- < > é escaped characters

HTML **5** semantic elements

HTML **5** semantic elements (part of)

Tag	Description
<article>	Defines an article in the document
<figcaption>	Defines a caption for a <figure> element
<figure>	Defines self-contained content, like illustrations, diagrams, photos, code listings, etc.
<footer>	Defines a footer for the document or a section
<header>	Defines a header for the document or a section
<main>	Defines the main content of a document
<menuitem>	Defines a command/menu item that the user can invoke from a popup menu
<nav>	Defines navigation links in the document
<section>	Defines a section in the document

- What is structural?
  - …
- What is style?
  - …
- What is behaviour?
  - …

General structure - <html>

```
<!DOCTYPE html>
<html>
 <head>
    ...
 </head>

 <body>
    ...
 </body>
</html>
```

Invisible info - <head>

```html
<head>
    <!-- head defines invisible information about the page -->
    <!-- Meta information (machine parseable) can also be defined -->
    <meta charset="UTF-8">
    <meta name="description" content="My Home Library Web Site">
    <meta name="keywords" content="HTML,CSS,XML,JavaScript">
    <meta name="author" content="João Costa Seco">
    <meta http-equiv="refresh" content="30">

    <!-- The title of the browser window (mandatory) -->
    <title>My home library</title>


    <!-- To define the base URL for all relative links -->
    <base href="ctp.di.fct.unl.pt/~jcs/daweb14-15">

    <!-- To define the style of the elements of the page -->
    <style>
        * { font-family: sans-serif;}
    </style>
    <!-- Or import an external file with the stylesheet -->
    <link href="screen.css" type="text/css" rel="stylesheet" media="screen"/>

    <!-- Custom behaviour can be defined using Javascript -->
    <!-- Directly in the HTML code -->
    <script>
        function Hello() { alert("Hello World."); }
    </script>
    <!-- Or by importing a script file -->
    <script src="books.js"></script>
</head>
```

Visible info - <body> - **old style**

```
<div id="header">
  <h1>Monday Times</h1>
</div>

<div id="menu">
  <ul>
    <li>News</li>
    <li>Sports</li>
    <li>Weather</li>
  </ul>
</div>

<div id="content">
  <h2>News Section</h2>
  <div class="article">
    <h2>News Article</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in
porta lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.</p>
  </div>
  <div class="article">
    <h2>News Article</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in
porta lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.</p>
  </div>
</div>

<div id="footer">
  <p>&amp;copy; 2016 Monday Times. All rights reserved.</p>
</div>
```

Visible info - <body> - **new style**

```
<header>
<h1>Monday Times</h1>
</header>

<nav>
<ul>
<li>News</li>
<li>Sports</li>
<li>Weather</li>
</ul>
</nav>

<section>
<h2>News Section</h2>
<article>
<h2>News Article</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in porta
lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.</p>
</article>
<article>
<h2>News Article</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque in porta
lorem. Morbi condimentum est nibh, et consectetur tortor feugiat at.</p>
</article>
</section>

<footer>
<p>&copy; 2014 Monday Times. All rights reserved.</p>
</footer>
```

Textual block elements

```
<html>
    <body>
        <h1>This is a heading</h1>
        <p>This is a paragraph.</p>
        <p>This is another paragraph.</p>
    </body>
</html>
```

Textual inline elements

```
<html>

    <body>

        <h1>This is a heading</h1>

        <p>This is a paragraph. <span>This is a span</span> <b>This is a strong tag</b>
        <a>This is an anchor</a> </p>

        <p>This is another paragraph.</p>

    </body>

</html>
```

Structural block elements

```
<html>
  <body>
    <nav>This is a navigation section</nav>
    <article>This is a document section.
      <section>This is another section. </section>
      <section>This is another section. </section>
    </article>
    <article>This is a document section.
      <section>This is another section. </section>
      <section>This is another section. </section>
    </article>
  </body>
```

Block elements

This is a block element.

This is a block element.

This is a block element.

Inline elements

```
<span>This is </span> <span> an inline </span> <span> element.
</span> <span>This is </span> <span> an inline </span> <span>
element. </span> <span>This is </span> <span> an inline </span>
<span> element. </span>
```

Attributes

- HTML elements can be modified through attributes

```
<a href="http://www.w3schools.com">This is a link</a>
<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
<div style="border:1px">This is a section.</div>
```

HTML <button> Tag

« Previous Complete HTML Reference Next »

Example

A clickable button is marked up as follows:

```
<button type="button">Click Me!</button>
```

Try it Yourself »

Attributes

= New in HTML5.

Attribute	Value	Description
autofocus	autofocus	Specifies that a button should automatically get focus when the page loads
disabled	disabled	Specifies that a button should be disabled
form	*form_id*	Specifies one or more forms the button belongs to

HTML - Basic Interaction

- Links
  - Produce a GET request to the given URL and replace the entire content of the DOM
- Forms
  - Pre-defined controls (buttons)
  - Produce a request and expects a response document (HTML)
  - replaces the entire content of the DOM (depends on _target)
  - method attribute defines the request type (GET/POST)
  - called URL format depends on the used method (body/query string)

HTML - FORMS

```html
<form action="/my-handling-form-page" method="post">
  <div>
    <label for="name">Name:</label>
    <input type="text" id="name" />
  </div>
  <div>
    <label for="mail">E-mail:</label>
    <input type="email" id="mail" />
  </div>
  <div>
    <label for="msg">Message:</label>
    <textarea id="msg"></textarea>
  </div>

  <div class="button">
    <button type="submit">Send your message</button>
  </div>
</form>
```

HTML

- Inputs

```
<input type="text" name="input" value="Type here">
```

```
<button name="button">Click me</button>
```

```
<select name="select">
  <option value="value1">Value 1</option>
  <option value="value2" selected>Value 2</option>
  <option value="value3">Value 3</option>
</select>
```

- Other interface elements (output)

```
<p>Heat the oven to <meter min="200" max="500" value="350">350 degrees</meter>.</p>
```

```
<progress value="70" max="100">70 %</progress>
```

CSS

CSS - Cascading Style Sheets

- Describes the presentation of a document defined using a markup language (HTML, XML)
- Decouples presentation from the document structure and behavior of a web application
- Based on declarative object descriptors and object attributes (and values) - Rules
- Based on general application priorities on conflicting cases
- Includes basic presentation behavior (animations)

Decorating the DOM

- A set of rules is applied to the DOM

- rule = selector + style definitions with the form **`property:value`**
- Rules are applied by the following order
  - Browser default
  - External and Internal Style Sheet (following the order in the section **`<head>`**)
  - Inline Style (HTML element)
- Propagate hierarchically through the DOM

CSS Selectors

from http://www.w3schools.com/cssref/css_selectors.asp

Selector	Example	Example description	CSS
.class	.intro	Selects all elements with class="intro"	1
#id	#firstname	Selects the element with id="firstname"	1
*	*	Selects all elements	2
element	p	Selects all <p> elements	1
element,element	div, p	Selects all elements and all <p> elements	1
element element	div p	Selects all <p> elements inside elements	1
element>element	div > p	Selects all <p> elements where the parent is a element	2
element+element	div + p	Selects all <p> elements that are placed immediately after elements	2
element1~element2	p ~ ul	Selects every <ul> element that are preceded by a <p> element	3
[*attribute*]	[target]	Selects all elements with a target attribute	2
[*attribute=value*]	[target=_blank]	Selects all elements with target="_blank"	2
[*attribute~=value*]	[title~=flower]	Selects all elements with a title attribute containing the word "flower"	2
[*attribute|=value*]	[lang|=en]	Selects all elements with a lang attribute value starting with "en"	2
[*attribute^=value*]	a[href^="https"]	Selects every <a> element whose href attribute value begins with "https"	3

CSS object selectors

- all elements:

```
* { font-family: sans-serif; }
```

- by HTML tag

```
h1, h2 { color: #f0f0f0; }
```

- by id

```
#form1 { border: solid 1px; }
```

- by class

```
.listitem { list-style: none; }
```

- composed selector

```
#booklist li { background-color: "rbg(100,80,10)"; }
```

- composed selector

```
ul.menu > li { margin: 10px; padding: 5px;}
```

- pseudo-classes

```
.menuitem:hover { background-color: blue; }
```

CSS Selectors

```
h1 {background-color: red;}
```

CSS Selectors

```
h1, h2, h4 {background-color: red;}
```

CSS Selectors

```
#comment {background-color: red;}
```

<section>

<h1>Heading 1</h1>

<h2>Heading 2</h2>

<h3>Heading 3</h3>

<h4>Heading 4</h4>

Lorem ipsum dolor sit amet, consectetur adipisicing elit,...

Lorem ipsum dolor sit amet, consectetur...

</section>

CSS Selectors

```
.text {background-color: red;}
```

```
<section>
  <h1>Heading 1</h1>
  <h2>Heading 2</h2>
  <h3>Heading 3</h3>
  <h4>Heading 4</h4>
  <p class="text">Lorem ipsum dolor sit amet, consectetur...</p>
  <p class="text">Lorem ipsum dolor sit amet, consectetur...</p>
</section>
```

CSS Selectors

```
div ul {background-color: red;}
```

```
<div>
    <ul>
        • <li>
            <p>Item 1</p>
          </li>
        • <li>
            <p>Item 2</p>
          </li>
        • <li>
            <ul>
                ◦ <li>
```

CSS Selectors

```
div li {border: dashed red 2px;}
```

CSS Selectors

```
div > ul > li {border: dashed red 2px;}
```

```
<div>
    <ul>
      • <li>
          <p>Item 1</p>
        </li>
      • <li>
          <p>Item 2</p>
        </li>
      • <li>
          <ul>
            ○ <li>
```

CSS Selectors

```
section+div {border: dashed red 2px;}
```

```
<p class="text">Lorem ipsum dolor sit amet, consectetur...</p>
<p class="text">Lorem ipsum dolor sit amet, consectetur...</p>
</section>
<div>
  Lorem ipsum ...
</div>
<div>
    <ul>
    • <li>
```

CSS Selectors

```
div~div {border: dashed red 2px;}
```

CSS performance

- CSS selectors are read from right to left

```
div#nav ul.menu > li.menuitem
```

  - Collect all menu items with CSS class .menuitem
  - Narrow down to all menu items that are actually list items (LI)
  - Narrow down to those that are under an element with CSS .menu
  - Narrow down to those which CSS .menu class belongs to a UL element
  - Narrow down to those items that fall under element of ID #nav
  - Finally, narrow the above to those elements with ID nav that are indeed DIV elements
- Better > ID > tag > class > composition > Worst

https://moduscreate.com/blog/efficient-dom-and-css/

CSS Properties

CSS Properties

CSS Property Groups

- Color
- Background and Borders
- Basic Box
- Flexible Box
- Text
- Text Decoration
- Fonts
- Writing Modes
- Table
- Lists and Counters
- Animation
- Transform
- Transition
- Basic User Interface
- Multi-column
- Paged Media
- Generated Content
- Filter Effects
- Image/Replaced Content
- Masking
- Speech
- Marquee

CSS Properties

Color Properties

Property	Description	CSS
color	Sets the color of text	1
opacity	Sets the opacity level for an element	3

Background and Border Properties

Property	Description	CSS
background	Sets all the background properties in one declaration	1
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page	1
background-color	Sets the background color of an element	1
background-image	Sets the background image for an element	1
background-position	Sets the starting position of a background image	1
background-repeat	Sets how a background image will be repeated	1
background-clip	Specifies the painting area of the background	3
background-origin	Specifies the positioning area of the background images	3
background-size	Specifies the size of the background images	3
border	Sets all the border properties in one declaration	1

The Box Model

- The layout of web pages is based on the setting of box dimensions, using 4 different measures
  - Width and height; padding; border; margin

```
#a1 { width:100px; margin:20px; padding: 5px;}

#a2 { margin: 5px; }
```

The Box Model

```
div {
    width: 300px;
    padding: 25px;
    border: 25px solid navy;
    margin: 25px;
}
```

```
div {
    box-sizing: border-box;
    width: 50%;
    float: left;
}
```

http://css-tricks.com/box-sizing/

Display, Visible, Float, and Position

- There are three ways (properties) that control the positioning of an H element in a rendered page.
- **`display`**: inline, block, inline-block, none, etc.
- **`visibility`**: visible, hidden, collapse (tables)
- **`float`**: left, right…
- **`position`**: static, absolute, fixed, relative
  - associated properties: left, right, top, bottom

Display

```
<div id="a1">1</div>
<div id="a2">2</div>
<div id="a3">3</div>
<div id="a4">4</div>
```

- inline

```
div { display: inline ; }
```

- block

```
div { display: block ; }
```

- none

```
#a2 { display:none; }
```

Visibility

```
<div id="a1">1</div>
<div id="a2">2</div>
<div id="a3">3</div>
<div id="a4">4</div>
```

- visible

1
2
3
4

- hidden

1
3
4

```
#a2 { visibility:hidden;    }
```

Float & Clear

- Property `float` lets the elements to be arranged freely and other elements wrap around them (e.g. text)
- Possible values: left, right

- clear resets wrapping of elements
- overflow may make the container extend to the borders of the floating elements.

```
<div class="left">1</div>
<div class="right">2</div>
<div class="right">3</div>
<div class="left clear">4</div>
<div class="left">5</div>
<div class="right">6</div>
```

The Box Model

Basic Box Properties

Property	Description	CSS
bottom	Specifies the bottom position of a positioned element	2
clear	Specifies which sides of an element where other floating elements are not allowed	1
clip	Clips an absolutely positioned element	2
display	Specifies how a certain HTML element should be displayed	1
float	Specifies whether or not a box should float	1
height	Sets the height of an element	1
left	Specifies the left position of a positioned element	2
overflow	Specifies what happens if content overflows an element's box	2
overflow-x	Specifies whether or not to clip the left/right edges of the content, if it overflows the element's content area	3
overflow-y	Specifies whether or not to clip the top/bottom edges of the content, if it overflows the element's content area	3
padding	Sets all the padding properties in one declaration	1
padding-bottom	Sets the bottom padding of an element	1
padding-left	Sets the left padding of an element	1
padding-right	Sets the right padding of an element	1

```html
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
    div { border: solid 1px; width: 50px; text-align: center; }

    .left { float: left;}

    .right { float:right;}

    .clear { clear: left;}

    .container { width: 100%; }

    p { margin: 0px; text-align: left;}
</style>
</head>
<body>

<div class="container">
<div class="left">1</div>
<div class="right">2</div>
<div class="right">3</div>
<div class="right">4</div>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
<div class="left">5</div>
<div class="right">6</div>
</div>
</body>
</html>
```

1	Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.	4	3	2
5				6

```html
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
    div { border: solid 1px; width: 50px; text-align: center; }

    .left { float: left;}

    .right { float:right;}

    .clear { clear: left;}

    .container { width: 100%; overflow: auto;}

    p { margin: 0px; text-align: left;}
</style>
</head>
<body>

<div class="container">
<div class="left">1</div>
<div class="right">2</div>
<div class="right clear">3</div>
<div class="right">4</div>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse
cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non
proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
</p>
<div class="left">5</div>
<div class="right">6</div>
</div>
</body>
</html>
```

1 Lorem ipsum dolor sit amet, consectetur adipisicing 2

elit, sed do eiusmod tempor incididunt ut labore 4 3

et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

5 6

http://www.w3schools.com/css/css_float.asp
https://developer.mozilla.org/en-US/docs/Web/CSS/float

Position

- Position property defines the position of an element (independent of the flow layout).
- static (initial value, in the flow)
- absolute (with relation to the container element)
- fixed (with relation to the window)
- relative (with relation to the position in the flow)
- sticky ("normal" until the constraints are triggered, then changes to fixed position (see http://jsfiddle.net/daker/ecpTw/light/).

Position

fixed block

container

. starting point

.

relative position

absolute position

```
.fixed {
    position:fixed;
    top: 10px;
    right: 0px;
    width: 100px;
}
.relative {
    position: relative;
    left:50px;
    top:50px;
    width:100px;
}
.absolute {
    position: absolute;
    bottom: 100px;
    left: 50px;
    width: 100px;
}
```

Responsive design

- Sets of methods and techniques to optimize reading and navigation of a certain web design to a myriad of devices and displays.

- Usually achieved by gracious resizing, hiding, and rearrangement of page sections.
  - Media queries (CSS @media rule)
  - Flexible (flow and grid) layouts (e.g. bootstrap)
  - JavaScript reactive interfaces

Media queries

- The @media rule is used to define different style rules for different media types/devices.

```
<!-- CSS media query on a link element -->
<link rel="stylesheet" media="(max-width: 800px)" href="example.css" />

<!-- CSS media query within a stylesheet -->
<style>
@media (max-width: 600px) {
  .facet_sidebar {
    display: none;
  }
}
</style>
```

https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries

Media queries

- @media <media type> and <media features>

```
@media (min-width: 700px) { ... }
```

```
@media (min-width: 700px) and (orientation: landscape) { ... }
```

```
@media tv and (min-width: 700px) and (orientation: landscape) { ... }
```

```
@media (min-width: 700px), handheld and (orientation: landscape) { ... }
```

https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries

Media queries

Pseudo-BNF (for those of you that like that kind of thing)

```
media_query_list: <media_query> [, <media_query> ]*
media_query: [[only | not]? <media_type> [ and <expression> ]*]
  | <expression> [ and <expression> ]*
expression: ( <media_feature> [: <value>]? )
media_type: all | aural | braille | handheld | print |
  projection | screen | tty | tv | embossed
media_feature: width | min-width | max-width
  | height | min-height | max-height
  | device-width | min-device-width | max-device-width
  | device-height | min-device-height | max-device-height
  | aspect-ratio | min-aspect-ratio | max-aspect-ratio
  | device-aspect-ratio | min-device-aspect-ratio | max-device-aspect-ratio
  | color | min-color | max-color
  | color-index | min-color-index | max-color-index
  | monochrome | min-monochrome | max-monochrome
  | resolution | min-resolution | max-resolution
  | scan | grid
```

https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Media_queries

Bootstrap

Preprocessors

Bootstrap ships with vanilla CSS, but its source code utilizes the two most popular CSS preprocessors, Less and Sass. Quickly get started with precompiled CSS or build on the source.

One framework, every device.

Bootstrap easily and efficiently scales your websites and applications with a single code base, from phones to tablets to desktops with CSS media queries.

Full of features

With Bootstrap, you get extensive and beautiful documentation for common HTML elements, dozens of custom HTML and CSS components, and awesome jQuery plugins.

Bootstrap Grid System

- Bootstrap is a CSS/Javascript client framework that provides simpler and effective layout tools.
- Bootstrap divides a page into 12 columns and provides classes for rows and columns:
  - `container`, `row`, `col-X-Y`, etc.
  - X: **xs** (phones), **sm** (tablets), **md** (desktops), and **lg** (larger desktops)
  - Y: 1 .. 12
  - Explore bootstrap grids at: http://getbootstrap.com/examples/grid/

Bootstrap Grid System

span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1	span 1
span 4				span 4				span 4			
span 4				span 8							
span 6						span 6					
span 12											

Internet Applications Design and Implementation

2020 - 2021

(Lecture 7 - Part 3: Client-side frameworks)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Client-side Frameworks and Languages

Many cooperating languages

Weak binding mechanisms

Untyped development environment

Weak execution guaranties

Low-level DOM manipulation

Sequential programming language

HTML

A dynamic computational platform

Uniform across browsers (almost)

portable to mobile and desktop

Internet ready low-level interface

Low-level support for concurrency

Equipped with many libraries (node)

Client-side Frameworks and Languages

HTML

+ Abstractions + Patterns + Languages + Tools = Frameworks

The Big 5 JavaScript Frameworks

The five JavaScript frameworks that currently dominate the market in terms of popularity and usage are:

- React
- Vue
- Angular
- Ember
- Backbone.js.

They each have large communities. If you are a front-end developer or are going to start your new project on front-end technologies, these five are your best bets. Here's a look at the npm trends over the last six months.

Sample Features

Often more than a client MVC

Join HTMLCSSJS in one place

Offer abstractions of the DOM

Composition mechanisms

Out of the box reactivity

Transparency across network

Compare Frameworks

TodoMVC

Helping you **select** an MV* framework

Download | View on GitHub | Blog

Introduction

Developers these days are spoiled with choice when it comes to **selecting** an **MV* framework** for structuring and organizing their JavaScript web apps.

Backbone, Ember, AngularJS… the list of new and stable solutions continues to grow, but just how do you decide on which to use in a sea of so many options?

To help solve this problem, we created **TodoMVC** - a project which offers the same Todo application implemented using MV* concepts in most of the popular JavaScript MV* frameworks of today.

Follow | Tweet | G+

Examples

JavaScript	Compile-to-JS		Labs
These are examples written in pure JavaScript.			
Backbone.js R	AngularJS R	Ember.js R	KnockoutJS R
Dojo R	Knockback.js R	CanJS R	Polymer R
React R	Mithril R	Vue.js R	Marionette.js R
These are applications written in programming languages that compile to JavaScript.			
Kotlin + React R	Spine R	Dart R	GWT R
Closure R	Elm R	AngularDart	TypeScript + Backbone.js
TypeScript + AngularJS	TypeScript + React	Reagent R	Scala.js + React R
Scala.js + Binding.scala R	js_of_ocaml R	Humble + GopherJS R	

http://todomvc.com

Compare Framework

Tod

Helping you selec

Download | View on GitHu

Introduction

Developers these days are spoiled
it comes to **selecting** an **MV* fram**
structuring and organizing their Jav
apps.

Backbone, Ember, AngularJS… th
stable solutions continues to grow,
you decide on which to use in a se
options?

To help solve this problem, we crea
a project which offers the same To
implemented using MV* concepts i
popular JavaScript MV* framework

Follow Tweet G+

just how do you decide on which to use in a sea of so many options?

To help solve this problem, we created **TodoMVC** - a project which offers the same Todo application implemented using MV* concepts in most of the popular JavaScript MV* frameworks of today.

Follow Twee G+

Dojo R | Knockback.js R | CanJS R | Polymer R

React R | Mithril R | Vue.js R | Marionette.js R

These are applications written in programming languages that compile to JavaScript.

Kotlin + React R | Spine R | Dart R | GWT R

Closure R | Elm R | AngularDart | TypeScript + Backbone.js

TypeScript + AngularJS | TypeScript + React | Reagent R | Scala.js + React R

Scala.js + Binding.scala R | js_of_ocaml R | Humble + GopherJS R

These are examples written in JavaScript that we are still evaluating.

Backbone.js + RequireJS R | KnockoutJS + RequireJS | AngularJS + RequireJS R | CanJS + RequireJS R

Lavaca + RequireJS R | cujoJS | Sammy.js R | soma.js

DUEL | Kendo UI R | Dijon | Enyo + Backbone.js R

SAPUI5 R | Exoskeleton R | Ractive.js | React + Alt R

React + Backbone.js | Aurelia | Angular 2.0 | Riot R

JSBlocks R

R = App also demonstrates routing

Real-time

SocketStream R | Firebase + AngularJS R

Node.js

Express + gcloud-node

Compare these to a non-framework implementation

Vanilla JS R | Vanilla ES6 R | jQuery R

Angular

ines (75 sloc) | 3.27 KB

```
<!doctype html>
<html lang="en" data-framework="angularjs">
	<head>
		<meta charset="utf-8">
		<title>AngularJS • TodoMVC</title>
		<link rel="stylesheet" href="node_modules/todomvc-common/base.css">
		<link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
		<style>[ng-cloak] { display: none; }</style>
	</head>
	<body ng-app="todomvc">
		<ng-view></ng-view>

		<script type="text/ng-template" id="todomvc-index.html">
			<section class="todoapp">
				<header class="header">
					<h1>todos</h1>
					<form class="todo-form" ng-submit="addTodo()">
						<input class="new-todo" placeholder="What needs to be done?" ng-mod
					</form>
				</header>
				<section class="main" ng-show="todos.length" ng-cloak>
					<input id="toggle-all" class="toggle-all" type="checkbox" ng-model="allChec
					<label for="toggle-all">Mark all as complete</label>
					<ul class="todo-list">
						<li ng-repeat="todo in todos | filter:statusFilter track by $index"
							<div class="view">
								<input class="toggle" type="checkbox" ng-model="tod
								<label ng-dblclick="editTodo(todo)">{{todo.title}}<
								<button class="destroy" ng-click="removeTodo(todo)"
							</div>
							<form ng-submit="saveEdits(todo, 'submit')">
								<input class="edit" ng-trim="false" ng-model="todo.
							</form>
						</li>
					</ul>
				</section>
```

```
angular.module('todomvc', ['ngRoute', 'ngResource'])
	.config(function ($routeProvider) {
		'use strict';

		var routeConfig = {
			controller: 'TodoCtrl',
			templateUrl: 'todomvc-index.html',
			resolve: {
				store: function (todoStorage) {
					// Get the correct module (API or localStorage).
					return todoStorage.then(function (module) {
						module.get(); // Fetch the todo records in the ba
						return module;
					});
				}
			}
		};
```

```
angular.module('todomvc')
	.directive('todoFocus', function todoFocus($timeout) {
		'use strict';

		return function (scope, elem, attrs) {
			scope.$watch(attrs.todoFocus, function (newVal) {
				if (newVal) {
					$timeout(function () {
						elem[0].focus();
					}, 0, false);
				}
			});
		};
	});
```

Vue

57 lines (56 sloc) | 2.43 KB

```html
<!doctype html>
<html data-framework="vue">
	<head>
		<meta charset="utf-8">
		<title>Vue.js • TodoMVC</title>
		<link rel="stylesheet" href="node_modules/todomvc-common/base.css">
		<link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
		<style> [v-cloak] { display: none; } </style>
	</head>
	<body>
		<section class="todoapp" v-cloak>
			<header class="header">
				<h1>todos</h1>
				<input class="new-todo" autofocus autocomplete="off" p
			</header>
			<section class="main" v-show="todos.length">
				<input id="toggle-all" class="toggle-all" type="checkb
				<label for="toggle-all">Mark all as complete</label>
				<ul class="todo-list">
					<li class="todo" v-for="todo in filteredTodos"
						<div class="view">
							<input class="toggle" type="ch
							<label @dblclick="editTodo(tod
							<button class="destroy" @click
						</div>
						<input class="edit" type="text" v-mode
					</li>
				</ul>
			</section>
<footer class="footer" v-show="todos.length">
```

```
	exports.app = new Vue({

		// the root element that will be compiled
		el: '.todoapp',

		// app initial state
		data: {
			todos: todoStorage.fetch(),
			newTodo: '',
			editedTodo: null,
			visibility: 'all'
		},

		// watch todos change for localStorage persistence
		watch: {
			todos: {
				deep: true,
				handler: todoStorage.save
			}
		},

		// computed properties
		// http://vuejs.org/guide/computed.html
		computed: {
			filteredTodos: function () {
				return filters[this.visibility](this.todos);
			},
			remaining: function () {
				return filters.active(this.todos).length;
			},
			allDone: {
				get: function () {
					return this.remaining === 0;
				},
				set: function (value) {
					this.todos.forEach(function (todo) {
						todo.completed = value;
```

React

33 lines (30 sloc) | 1.29 KB

```html
<!doctype html>
<html lang="en" data-framework="react">
	<head>
		<meta charset="utf-8">
		<title>React • TodoMVC</title>
		<link rel="stylesheet" href="node_modules/todomvc-common/base.css">
		<link rel="stylesheet" href="node_modules/todomvc-app-css/index.css">
	</head>
	<body>
		<section class="todoapp"></section>
		<footer class="info">
			<p>Double-click to edit a todo</p>
			<p>Created by <a href="http://github.com/petehunt/">petehunt</a></p>
			<p>Part of <a href="http://todomvc.com">TodoMVC</a></p>
		</footer>

		<script src="node_modules/todomvc-common/base.js"></script>
		<script src="node_modules/react/dist/react-with-addons.js"></script>
		<script src="node_modules/classnames/index.js"></script>
		<script src="node_modules/react/dist/JSXTransformer.js"></script>
		<script src="node_modules/director/build/director.js"></script>

		<script src="js/utils.js"></script>
		<script src="js/todoModel.js"></script>
		<!-- jsx is an optional syntactic sugar that transforms methods in React's
		`render` into an HTML-looking format. Since the two models above are
		unrelated to React, we didn't need those transforms. -->
		<script type="text/jsx" src="js/todoItem.jsx"></script>
		<script type="text/jsx" src="js/footer.jsx"></script>
		<script type="text/jsx" src="js/app.jsx"></script>
	</body>
</html>
```

```
function render() {
	React.render(
		<TodoApp model={model}/>,
		document.getElementsByClassName('todoapp')[0]
	);
}
```

```
return (
	<div>
		<header className="header">
			<h1>todos</h1>
			<input
				className="new-todo"
				placeholder="What needs to be done?"
				value={this.state.newTodo}
				onKeyDown={this.handleNewTodoKeyDown}
				onChange={this.handleChange}
				autoFocus={true}
			/>
		</header>
		{main}
		{footer}
	</div>
);
```

React and Kotlin/JS

```
import kotlinx.html.*
import kotlinx.html.js.*
import org.jetbrains.common.*
import org.jetbrains.demo.thinkter.model.*
import react.*
import react.dom.*
import kotlin.browser.*
import kotlinx.coroutines.experimental.async

class LoginComponent : ReactDOMComponent<UserProps, LoginFormSta
    companion object : ReactComponentSpec<LoginComponent, UserPr

    init {
        state = LoginFormState("", "", false, "")
    }

    override fun ReactDOMBuilder.render() {
        div {
            form(classes = "pure-form pure-form-stacked") {
                legend { +"Login" }

                fieldSet(classes = "pure-group") {
                    input(type = InputType.text, name = "login")
                        value = state.login
                        placeholder = "Login"
                        disabled = state.disabled

                        onChangeFunction = {
                            setState {
                                login = it.inputValue
                            }
                        }
                    }
                    input(type = InputType.password, name = "pas
                        value = state.password
```

```
package org.jetbrains.demo.thinkter

import kotlinx.html.*
import kotlinx.html.js.*
import org.jetbrains.demo.thinkter.model.*
import react.*
import react.dom.*
import kotlin.browser.*
import kotlinx.coroutines.experimental.async

fun main(args: Array<String>) {
    runtime.wrappers.require("pure-blog.css")

    ReactDOM.render(document.getElementById("content")) {
        div {
            Application {}
        }
    }
}

class Application : ReactDOMComponent<ReactComponentNoProps, ApplicationPageState>() {
    companion object : ReactComponentSpec<Application, ReactComponentNoProps, ApplicationPageState>
    val polling = Polling()

    init {
        state = ApplicationPageState(MainView.Home)
        checkUserSession()
    }

    override fun componentWillUnmount() {
        polling.stop()
        super.componentWillUnmount()
    }

    override fun ReactDOMBuilder.render() {
        div("pure-g") {
```

Elm Language (elm-lang.org)

Why a *functional* language?

- No runtime errors in practice. No `null`. No `undefined` is not a function.
- Friendly error messages that help you add features more quickly.
- Well-architected code that *stays* well-architected as your app grows.
- Automatically enforced semantic versioning for all Elm packages.

https://ellie-app.com/new

```
module Main exposing (main)

import Browser
import Html exposing (Html, button, div, text)
import Html.Events exposing (onClick)

type alias Model = { count : Int }

initialModel : Model
initialModel = { count = 0 }

type Msg = Increment
         | Decrement

update : Msg -> Model -> Model
update msg model =
    case msg of
        Increment ->
            { model | count = model.count + 1 }

        Decrement ->
            { model | count = model.count - 1 }

view : Model -> Html Msg
view model =
    div []
        [ button [ onClick Increment ] [ text "+1" ]
        , div [] [ text <| String.fromInt model.count ]
        , button [ onClick Decrement ] [ text "-1" ]
        ]

main : Program () Model Msg
main =
    Browser.sandbox
        { init = initialModel
        , view = view
        , update = update
        }
```

Elm Lang

Why a *functi*

- No runtime errors
- Friendly error me
- Well-architected
- Automatically enf

https://ellie

BEST FUNCTIONAL LANGUAGES TO LEARN FOR WEB-FRONTEND DEVELOPMENT		PRICE	SITE	PRICE
90	Elm	-	http://elm-lang.org/	Open Sou
--	Reason ML	-	-	Free, Open s
71	ClojureScript	OPEN SOURCE	https://clojurescript.org/	Open Sou
--	OCaml	-	http://ocaml.org	Open source
--	F#	-	http://www.fsharp.org	-
--	Purescript	-	-	-
58	Scala	-	scala-lang.org	Open Source
--	JavaScript	-	-	-
--	Haste	-	-	-
--	Haskell	-	https://www.haskell.org/	-
--	Elixir	-	https://elixir-lang.org	Open source
--	Kotlin	-	http://kotlinlang.org	Free, Open S

```
xt)
1 }
1 }
xt "+1" ]
odel.count ]
xt "-1" ]
```

Loom Language

- By Nuno Castro Martins, MSc FCT UNL 2017

https://loom-lang.gitlab.io/loom-playground/ https://gitlab.com/loom-lang/loom

```
// Tasks counter
var id = 0

// Mutable list of tasks
var tasks = mut [createTask("Sleep"), createTask("Eat"), createTask("Play")]

// Creates a new task object (with a mutable "done" value)
function createTask(text)
  ({id: id++, text: text, done: mut false})

// Adds a task to the list of tasks when "Enter" is pressed
function addTaskOnEnter(e)
  if (e.key == "Enter") {
    *tasks = [...*tasks, createTask(e.target.value)]
    e.target.value = ""
  }

// CSS of a task
function taskCSS(task)
  css {
    |.text:if(task.done)| {
      textDecoration: "line-through"
    }
  }
```

Loom Language

- By Nuno Castro Martins, MSc FCT UNL 2017

https://loom-lang.gitlab.io/loom-playground/ https://gitlab.com/loom-lang/loom

```
// HTML representation of a task
function taskHtml(task)
  <li.task {key: task.id, css: taskCSS(task)}> [
    <input.done {type: "checkbox", checked: task.done} />
    <span.text> task.text
  ]

// Number of tasks left to do
var nLeft = sig *tasks.reduce((n, task) => n + if (*(task.done)) 0 else 1, 0)

// Page to show
export default <div> [
  <input {onKeyDown: addTaskOnEnter} />
  <span> [nLeft, " left"]
  <ul>
    sig for (var task in *tasks) taskHtml(task)
]
```

Internet Applications Design and Implementation

2020 - 2021

(Lecture 7 - Part 4: Front-end languages)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

JavaScript in the browser

```
<h1>JavaScript Can Validate Input</h1>

<p>Please input a number between 1 and 10:</p>

<input id="numb" type="number">

<button type="button" onclick="myFunction()">Submit</button>

<p id="demo"></p>

<script>
function myFunction() {
    var x, text;

    // Get the value of the input field with id="numb"
    x = document.getElementById("numb").value;

    // If x is Not a Number or less than one or greater than 10
    if (isNaN(x) || x < 1 || x > 10) {
        text = "Input not valid";
    } else {
        text = "Input OK";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>
```

JavaScript - events

```
<some-HTML-element some-event='some JavaScript'>
```

```
<button onclick='document.getElementById("demo").innerHTML=Date()'>
The time is?
</button>
```

```
<button onclick="displayDate()">The time is?</button>
```

JavaScript + AJAX

- Asynchronous requests

```
<div id="demo"><h2>Let AJAX change this text</h2></div>

<button type="button" onclick="loadDoc()">Change Content</button>

<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (xhttp.readyState == 4 && xhttp.status == 200) {
      document.getElementById("demo").innerHTML = xhttp.responseText;
    }
  }
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>
```

JavaScript Frameworks

JavaScript Frameworks

- Abstraction of browser related details
  (No, they are not the same as the standard!).
- Higher abstraction level in browser related operations.
  - Searching the DOM and iterating elements (jQuery)
  - Updating values of elements in the DOM (React)
- Inversion of control
  - Basic event model already provide it
  - Frameworks extend it further

jQuery

What is jQuery?

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

jQuery

$(*selector*).*action*()

- A $ sign to define/access jQuery
- A (*selector*) to "query (or find)" HTML elements
- A jQuery *action*() to be performed on the element(s)

jQuery - examples

$(this).hide() - hides the current element.

$("p").hide() - hides all <p> elements.

$(".test").hide() - hides all elements with class="test".

$("#test").hide() - hides the element with id="test".

```
$(document).ready(function(){
    // jQuery methods go here...
});
```

jQuery - selectors

Syntax	Description	Ex
$("*")	Selects all elements	Tr
$(this)	Selects the current HTML element	Tr
$("p.intro")	Selects all <p> elements with class="intro"	Tr
$("p:first")	Selects the first <p> element	Tr
$("ul li:first")	Selects the first <li> element of the first <ul>	Tr
$("ul li:first-child")	Selects the first <li> element of every <ul>	Tr
$("[href]")	Selects all elements with an href attribute	Tr
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"	Tr
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"	Tr
$(":button")	Selects all <button> elements and <input> elements of type="button"	Tr
$("tr:even")	Selects all even <tr> elements	Tr
$("tr:odd")	Selects all odd <tr> elements	Tr

jQuery - events

```
$("p").click(function(){
    $(this).hide();
});
```

```
$("#p1").hover(function(){
    alert("You entered p1!");
},
function(){
    alert("Bye! You now leave p1!");
});
```

jQuery - AJAX

```
$.ajax({
  method: "POST",
  url: "some.php",
  data: { name: "John", location: "Boston" }
})
  .done(function( msg ) {
    alert( "Data Saved: " + msg );
  });
```

```
$.ajax({
  url: "test.html",
  cache: false
})
  .done(function( html ) {
    $( "#results" ).append( html );
  });
```

Languages - TypeScript

- type annotations
- interfaces
- classes (now in ES6),
- Mixins
- any type, Generics
- Type inference,
- Namespaces and modules
- JSX

```
class Student {
    fullName: string;
    constructor(public firstName: string,
           public middleInitial: string,
           public lastName: string) {
        this.fullName = firstName + " " +
                        middleInitial + " " +
                        lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person : Person) {
    return "Hello, " +
           person.firstName + " " +
           person.lastName;
}

var user = new Student("Jane", "M.", "User");

document.body.innerHTML = greeter(user);
```

Roadmap (again)

Tools for Internet Applications

Programming Languages for Internet Applications

https://hackernoon.com/the-2019-web-developer-roadmap-ab89ac8c380e

Client-side frameworks (a case for React and Typescript)

- Client applications include many standardised code for many common situations
- Frameworks and libraries provide support for those patterns
  - separation of concerns, reuse of code and inversion of control
- Strongly typed languages provide a safe usage of libraries and frameworks

Internet Applications Design and Implementation

2019 - 2020

(Lab class 7 - barebones web client)

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 7

HTML, Javascript & css

HTML, Javascript & css

- In this lab you will exercise the base browser technologies by building a dynamic web page that lists a set of resources loaded from a service using Ajax (XMLHttpRequest) and dynamically builds the HTML elements in the web page.

Internet Applications Design and Implementation

2020 - 2021

(Lecture 8 - Frontend Libraries/Frameworks/Languages)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outline

- Front-end libraries and tools
- React

Internet Applications Design and Implementation

2019 - 2020

(Lecture 8 - Part 1 - Front-end Libraries)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Client Development (HTML5 - HTML, JS, CSS)

- Unobtrusive Javascript (and CSS)
  - Good and valid markup, easier to write without inline javascript
  - Loosely coupled structure, behaviour and presentation layers.
  - Name conventions and string based hooks hinder the development.
  - CSS lacks abstraction mechanisms, results in expanded and repetitive stylesheets
  - In practice, low code reuse and poor organisation.
  - Javascript is not friendly to manipulate DOM elements
  - Layouts are hard to get right and flexible in all devices and sizes

Development Tools

- **JQuery**

  Solves dynamic manipulation of DOM structures. Does not add much to the native event based model of javascript.

- **Bootstrap** (and many other CSS frameworks)

  Solves presentation and layout problems, still needs many string based conventions, tight coupling, which are hard to maintain.

- **Coffeescript**

  Simplifies the syntax of JavaScript. compiles down to Javascript.

- **Sass**

  Adds nested rules (less repetition), variables (abstraction, less literal repetition), mixins (composition/extension), file organization (imports). compiles to basic CSS.

Libraries - example: JQuery

- Provides shortcuts for the dynamic DOM manipulation
- Follows the *well-known* style of css selectors

```
$( selector ).action( arguments )
```

- Abstraction layer for DOM events

```
$(function() { do this or that })
```

- Functional style in event handlers and operations
- The "*de facto*" Javascript library

Libraries - example: JQuery

- Abstraction layer for AJAX requests

```
$.get("/serverurl/path/?querystring",
  function(data,status) {…})

$.post("/serverurl/path/?querystring",
  { arg1: val1, arg2: val2, … },
  function(data,status) {…})

$.ajax({type:"POST", …
        success: function() {…},
        error: function() {…}})
```

Libraries - example: JQuery

```
$( selector ).action( arguments )
```

- Returns a list of (wrapped) elements matching `selector`
- Wraps all resulting elements with JQuery DOM objects.
- Chains operations on results (`fmap` functional style)

```
$("div p.cell").click(handleClick)
      .filter(".hidden")
      .addClass("done")
      .hide()

messages
   .map((msg)->$("ul.todo")
                .append($("li").text(msg))
```

Libraries - example: JQuery

- Still based on Javascript code patterns
- Chains of operations are ok, but wrappers are annoying…
- It's still "stringly typed" and error prone

Frameworks - NodeJS

- Cross-platform environment (server and client)
- Node provides a modular environment for Javascript (and a package manager, NPM)
- `require('fs')`,
- `exports.a = 1`
- `import` is part of ES2015, but… not really in browsers and node 6, but only using transpilers (babel) or bundlers (webpack).

Library - Bootstrap (CSS, HTML, and JS)

- Based on JQuery, developed using Sass
- Distributed in compiled and minified files
- Abstracts browser differences
- Responsive design from scratch, mobile centred
- Uniform look (the modern default)
- Grid system that makes layout construction easy and adaptable
- Javascript plugins (e.g. modals) that include behaviour and custom events.
- Configuration using Sass & NPM build scripts

Frameworks - Babel

- Babel is a javascript compiler
- Provides tools to build new languages and DSLs over javascript (and generate javascript target code)
- Solves the issues of ES6 and Node.JS on module loading time, by translating them to `require` operations.

Languages - Coffeescript

- compiles down to Javascript.
- based on npm and babel.
- Simplifies the syntax of JavaScript.

```
square = (x) -> x * x
cube   = (x) -> square(x) * x
```

```
var cube, square;

square = function(x) {
  return x * x;
};

cube = function(x) {
  return square(x) * x;
};
```

Languages - Coffeescript

- compiles down to Javascript.
- based on npm and babel.
- Simplifies the syntax of JavaScript.

```
kids =
  brother:
    name: "Max"
    age:  11
  sister:
    name: "Ida"
    age:  9
```

```
kids = {
  brother: {
    name: "Max",
    age: 11
  },
  sister: {
    name: "Ida",
    age: 9
  }
};
```

Languages - Preprocessors: Sass, Less

- modularity, abstraction and extension mechanisms

https://www.keycdn.com/blog/sass-vs-less/

Client Development (HTML5 - HTML, JS, CSS)

- Unobtrusive Javascript (and CSS)
  - Good and valid markup, easier to write without inline javascript
  - Loosely coupled structure, behaviour and presentation layers.
  - Name conventions and string based hooks hinder the development.
  - ~~CSS lacks abstraction mechanisms, hence expanded and repetitive stylesheets are frequent.~~
  - ~~In practice, low code reuse and poor organisation.~~
  - ~~Javascript is not friendly to manipulate DOM elements~~
  - ~~Layouts are hard to get right and flexible in all devices and sizes~~

Frameworks - AngularJS

- Two way binding between view and model
- Reactive, valid HTML with custom attributes

```
1.  <!doctype html>
2.  <html ng-app="todoApp">
3.    <head>
4.      <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.6/angular.min.js"></script>
5.      <script src="todo.js"></script>
6.      <link rel="stylesheet" href="todo.css">
7.    </head>
8.    <body>
9.      <h2>Todo</h2>
10.     <div ng-controller="TodoListController as todoList">
11.       <span>{{todoList.remaining()}} of {{todoList.todos.length}} remaining</span>
12.       [ <a href="" ng-click="todoList.archive()">archive</a> ]
13.       <ul class="unstyled">
14.         <li ng-repeat="todo in todoList.todos">
15.           <label class="checkbox">
16.             <input type="checkbox" ng-model="todo.done">
17.             <span class="done-{{todo.done}}">{{todo.text}}</span>
18.           </label>
19.         </li>
20.       </ul>
21.       <form ng-submit="todoList.addTodo()">
22.         <input type="text" ng-model="todoList.todoText"  size="30"
23.                placeholder="add new todo here">
24.         <input class="btn-primary" type="submit" value="add">
25.       </form>
26.     </div>
27.   </body>
28. </html>
```

Frameworks - React

- Structure building of UI
  - Components
  - Parameters
  - State
- Reactive refreshing
- JSX syntax
- Routing
- State management with Redux/MobX

```
class TodoApp extends React.Component {
  constructor(props) {
    super(props);
    this.state = { items: [], text: '' };
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  render() {
    return (
      <div>
        <h3>TODO</h3>
        <TodoList items={this.state.items} />
        <form onSubmit={this.handleSubmit}>
          <input
            onChange={this.handleChange}
            value={this.state.text}
          />
          <button>
            Add #{this.state.items.length + 1}
          </button>
        </form>
      </div>
    );
  }
  handleChange(e) {
    this.setState({ text: e.target.value });
  }
  handleSubmit(e) {
    e.preventDefault();
    const newItem = {
      text: this.state.text,
      id: Date.now()
    };
    this.setState((prevState) => ({
      items: prevState.items.concat(newItem),
      text: ''
    }));
  }
}
class TodoList extends React.Component {
  render() {
    return (
      <ul>
        {this.props.items.map(item => (
          <li key={item.id}>{item.text}</li>
        ))}
      </ul>
    );
  }
}
ReactDOM.render(<TodoApp />, mountNode);
```

Languages - TypeScript

- type annotations
- interfaces
- classes (now in ES6),
- Mixins
- any type, Generics
- Type inference,
- Namespaces and modules
- JSX

```
class Student {
    fullName: string;
    constructor(public firstName: string,
          public middleInitial: string,
          public lastName: string) {
        this.fullName = firstName + " " +
                        middleInitial + " " +
                        lastName;
    }
}

interface Person {
    firstName: string;
    lastName: string;
}

function greeter(person : Person) {
    return "Hello, " +
           person.firstName + " " +
           person.lastName;
}

var user = new Student("Jane", "M.", "User");

document.body.innerHTML = greeter(user);
```

Internet Applications Design and Implementation

2019 - 2020

(Lecture 8 - Part 2 - React)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

React

https://facebook.github.io/react/

React is component-based

- The user interface is a composition of predefined building blocks
- It allows to build encapsulated components that manage their own state
- Components can be assembled/composed to make more elaborate UIs
- Hierarchical structures are easily instantiated in many different contexts

React is declarative, reactive and responsive

- Component logic is written in JavaScript instead of HTML templates
- Design simple views for each state in the app
- Combine and configure the active view using simple dynamic decisions
- React efficiently updates and renders components when data changes

The Key Ingredients

React components

- React components are classes that implement a `render()` method that **takes input data** and **returns what to display**
- The next example uses an XML-like syntax called JSX to represent html elements
- Input data that is passed hierarchically to the component can be accessed by `render()` via `this.props`
- JSX is optional and is not required to be used in React.

```
class HelloMessage extends React.Component<{name:String}> {
    render() {
        return <div>Hello {this.props.name}</div>;
    }
}

ReactDOM.render(<HelloMessage name="Jane" />,
                document.getElementById("root"));
```

A react component with state

- A component can maintain internal state
- Accessed via `this.state` and changed by `setState()`
- When a component's state data changes, the rendered markup will be updated by re-invoking `render()`

```
interface TimerInterface { secondsElapsed: number }

class Timer extends React.Component<{},TimerInterface> {
 …
    render() {
        return (
            <div>
                Seconds elapsed since you arrived: {this.state.secondsElapsed}
            </div>
        )
    }
}

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

```
interface TimerInterface { secondsElapsed: number }

class Timer extends React.Component<{},TimerInterface> {

    constructor(props:any) {
        super(props)
        this.state = { secondsElapsed: 0 };
    }
    …
    render() {
        return (<div>
                    Seconds elapsed since you arrived: {this.state.secondsElapsed}
                </div>)
    }
}

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

initial state

getter

```
class Timer extends React.Component<{},TimerInterface> {
    interval:any

    constructor(props:any) {
        super(props)
        this.state = { secondsElapsed: 0 };
        this.interval = null;
    }

    tick() { this.setState(({secondsElapsed}) => {secondsElapsed: secondsElapsed + 1}) }

    componentDidMount() { this.interval = setInterval(() => this.tick(), 1000); }

    componentWillUnmount() { clearInterval(this.interval); }

    render() {…}
}
```

setter

```
interface TimerInterface { secondsElapsed: number }

class Timer extends React.Component<{},TimerInterface> {
    interval:any

    constructor(props:any) {
        super(props)
        this.state = { secondsElapsed: 0 };
        this.interval = null;
    }

    tick() { this.setState(({secondsElapsed}) => {secondsElapsed: secondsElapsed + 1}) }

    componentDidMount() { this.interval = setInterval(() => this.tick(), 1000); }

    componentWillUnmount() { clearInterval(this.interval); }

    render() {
        return (<div>
                    Seconds elapsed since you arrived: {this.state.secondsElapsed}
                </div>)
    }
}

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

React components

- React components can be defined as **stateless components**, functions that implement the `render()` function directly and **take input data** and **return what to display**

- Input data that is passed into the component can be accessed by `render()` via parameters for `props`

- Light-weight components are faster to render and take less memory

```
const HelloMessage =
  (props:{name:string}) => <div>Hello {props.name}</div>;

ReactDOM.render(<HelloMessage name="Jane" />,
                document.getElementById("root"));
```

A react component with state

- A component can maintain internal state data
- Accessed via “hooks” (`useState`)
- When a component's state data changes, the rendered markup will be updated by re-evaluating the stateless component function.
- Other effects are achieved with other (specialised) hooks (`useEffect`)

https://react-hooks-cheatsheet.com/useeffect

```
const Timer = () => {
    const [ seconds, setSeconds ] = useState(0)

    let tick = () => { setSeconds((seconds) => seconds+1) }

    let interval = null;

    useEffect(() => { interval = setInterval(() => tick(), 1000); });

    return (
        <div>
            Seconds elapsed since you arrived: {seconds}
        </div>
    )
};

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

getter setter Initial state

```
const Timer = () => {
    const [ seconds, setSeconds ] = useState(0)

    let tick = () => { setSeconds((seconds)=>seconds+1) }

    let interval = null;

    useEffect(() => { interval = setInterval(() => tick(), 1000); });

    return (
        <div>
            Seconds elapsed since you arrived: {seconds}
        </div>
    )
};

ReactDOM.render(<Timer />, document.getElementById("react_content"));
```

setter

getter

A complete example

Example: A complete reactive TODO app

TO DO

- CIAI midterm test
- CIAI final test

CIAI Project | Add #3: CIAI Project

Example: A complete reactive TODO app

- How to render a list of items?
- Use a stateless component that receives the list (as props) and renders it:

```
interface Item { id:number, text:string }

const TodoList = (props:{items:Item[]}) => {
    return (
        <ul>
            {props.items.map(item => (
                <li key={item.id}>{item.text}</li>
            ))}
        </ul>
    );
};
```

Example: A complete reactive TODO app

- Where to store the list of items? and the text of new tasks?

```
interface ToDoState { items:Item[], text:string }

class ToDo extends React.Component<{},ToDoState> {

    constructor(props:{}) {
        super(props);
        this.state = { items: [], text:"" }
    }
…
```

Example: A complete reactive TODO app

- How to render the list of tasks, and accept new ones? (with a list and a form)

```
class ToDo extends React.Component<{},ToDoState> {
  …
  render() {
        return (
            <div>
                <h3>TO DO</h3>
                <TodoList items={this.state.items}/>
                <form onSubmit={this.handleSubmit}>
                    <input onChange={this.handleChange} value={this.state.text}/>
                    <button> Add </button>
                </form>
            </div>
        );
    }
```

In this case, every time the item text changes, the button label also changes

Example: A complete reactive TODO app

- How to react to user interaction events? using event handlers that change the state…

```
class ToDo extends React.Component<{},ToDoState> {
  …
  handleSubmit = (e:any) => {
    e.preventDefault();
    var newItem = { text: this.state.text, id: Date.now() };
    this.setState((prevState) => ({
        items: [...prevState.items, newItem],
        text: “”
    }));
  };

  handleChange = (e:any) => { this.setState({text: e.target.value}); };
  …
```

In this case, every time the item text changes, the button label also changes

with hooks

```
const ToDo = () => {

    const [ state, setState ] = useState({ items: [] as Item[], text:"" });

    let handleSubmit = (e:any) => {
        e.preventDefault();
        let newItem = { text: state.text, id: Date.now() }
        setState({ items: [...state.items, newItem], text: '' });
    };

    let handleChange = (e:any) => { setState({...state, text: e.target.value}); };

    return (<div> <h3>TO DO</h3>
                <TodoList items={state.items}/>
               <form onSubmit={handleSubmit}>
                   <input onChange={handleChange} value={state.text}/>
                   <button>
                       {'Add #' + (state.items.length + 1) + ': ' + state.text}
                   </button>
               </form>
            </div>);
};
```

Ownership of components

- In React, an owner is the component that sets the `props` of other components
- More formally, if a component X is created in component Y's render() method, it is said that X is owned by Y
- Note that a component cannot mutate its `props`
- `props` are always consistent with what its owner sets them to
- This fundamental invariant leads to UIs that are guaranteed to be consistent
- In the todo example, TodoApp is the owner of TodoList

Ownership vs. Parenting

- There is a difference between the **owner-ownee** relationship and the **parent-child** relationship
- The owner-ownee relationship is specific to React
- The parent-child relationship is simply the one you know from the DOM
- In the todo example TodoApp is the **owner** of h3, and TodoList and form is the **parent** of input

Inverse data flow or Two-way binding

- Components can only update their own state
- Ownees can only call owners update functions
- So how to do this?

```
const TodoList = (props:{items:Item[], completeItem:(value:boolean)=>void }) => {

    let check = (e:any) => {
        if(e.target.checked)
            props.completeItem(true);
        else
            props.completeItem(false);
    };

    return (
        <ul>
            {props.items.map(item => (
                <li key={item.id}>
                    <input type="checkbox" key={"done"+item.id} onClick={check} />
                    {item.text}
                </li>
            ))}
        </ul>
    );
};
```

```
const ToDo = () => {
    const [ state, setState ] = useState({ items: [] as Item[], text:"", done:0 });

    let handleSubmit = …

    let handleChange = …

    let completeItem = (value:boolean) => {
        if(value) setState({...state, done:state.done+1 });
        else      setState({...state, done:state.done-1 });
    };

    return (<div>
                <h3>TO DO</h3>
                <TodoList items={state.items} completeItem={completeItem}/>
                <form onSubmit={handleSubmit}>
                    <input onChange={handleChange} value={state.text}/>
                    <button>
                        {'Add #' + (state.items.length + 1) + ': ' + state.text}
                    </button>
                    <p>{ "Tasks done "+ state.done }</p>
                </form>
            </div>);
};
```

Routing - setup

- In the React application:

```
var Link = ReactRouter.Link;
var Router = ReactRouter.Router;
var Route = ReactRouter.Route;
```

https://css-tricks.com/learning-react-router/

Routing

```
var HelloMessageRouted = React.createClass ({
   render: function() {
      return (<div>Hello {this.props.params.name}</div>)
   }});

var SomeComponent = React.createClass ({
   render: function() {
      return (…
     <Link to={`/timer`}>Click to go to timer app</Link>
   …)
   }});

ReactDOM.render((
      <Router>
         <Route path="/" component={TodoApp}/>
         <Route path="/timer" component={Timer}/>
         <Route path="/hello/:name" component={HelloMessage}/>
        </Router>
      ), document.getElementById('react_content'))
```

React Thinking

https://facebook.github.io/react/docs/thinking-in-react.html

Break the UI into a component hierarchy

Identify the UI minimal complete state

1. Is it passed in from a parent via props? If so, it probably isn't state.
2. Does it remain unchanged over time? If so, it probably isn't state.
3. Can you compute it based on any other state or props in your component? If so, it isn't state.

State example

- Pieces of data:
  - The original list of products
  - The search text the user has entered
  - The value of the checkbox
  - The filtered list of products

- Actual state:
  - The search text the user has entered
  - The value of the checkbox

Where should the state live?

- Probably not in the place you think…
- Form component? Think again…
- React is all about **one-way** data flow **down the component hierarchy**
- It may not be immediately clear which component should own what state

Find the state component by Q&A

For each piece of state in your application:

- Identify every component that renders something based on that state.
- Find a common owner component (a single component above all the components that need the state in the hierarchy).
- Either the common owner or another component higher up in the hierarchy should own the state.
- If you can't find a component where it makes sense to own the state, create a new component simply for holding the state and add it somewhere in the hierarchy above the common owner component.

So, where should the state be?

- Pieces of data:
  - The original list of products
  - The search text the user has entered
  - The value of the checkbox
  - The filtered list of products

- Actual state:
  - checkbox value: yellow component, complete app
  - search text: yellow component, complete app

Summary

- Create a component for each “part” of your app
- A component receives input (optional)
- And defines how it should be rendered (mandatory) in the `render` method
- Use `this.props` to access the inputs

Summary

- Optionally define a state to your component as a record (in `getInitialState`)
- Every time the component state changes the React runs the `render` method
- **Never** update the state directly!!
- Use `setState` to update it so Rect can properly propagate changes
- You can access the previous state just before updating it
- Just define as input for `setState` a function that receives the previous state and use it inside the function in any way you need

Summary

- Recall that React is all about one-way data flow
- Two make it two-way you need to explicitly state that in your code
- Owner components send a call-back function to ownees (through `props`) so they can update the owner state by calling the function

React and Bootstrap

Bootstrap and React

- Bootstrap provides css and javascript that can be linked to React output

```
import React from 'react';

const Example = () => {
    return (
        <div class="alert alert-danger alert-dismissible fade show" role="alert">
            <strong>Oh snap! You got an error!</strong>
            <p>
                Change this and that and try again.
            </p>
            <button type="button" class="close" data-dismiss="alert" aria-label="Close">
                <span aria-hidden="true">&times;</span>
            </button>
        </div>
    )
}
```

Bootstrap and React

- Bootstrap-react is a complete re-implementation of bootstrap to provide react components. Connection at the abstract level and not at the browser level.

```
import React, { Component } from 'react';
import Alert from 'react-bootstrap/Alert';

const Example = () => {
    return (
        <Alert dismissible variant="danger">
            <Alert.Heading>Oh snap! You got an error!</Alert.Heading>
            <p>
                Change this and that and try again.
            </p>
        </Alert>
    )
}
```

Bootstrap and React - also with state

```
const AlertDismissible = () => {
    const [show, setShow] = useState(true);

    return (
        <> <Alert show={show} variant="success">
                <Alert.Heading>How's it going?!</Alert.Heading>
                <p>
                    Duis mollis, est non commodo luctus, nisi erat porttitor ligula, eget
                    lacinia odio sem nec elit. Cras mattis consectetur purus sit amet
                    fermentum.
                </p>
                <hr />
                <div className="d-flex justify-content-end">
                    <Button onClick={() => setShow(false)} variant="outline-success">
                        Close me ya'll!
                    </Button>
                </div>
            </Alert>
            {!show && <Button onClick={() => setShow(true)}>Show Alert</Button>}
        </>);}
```

Bootstrap and React

- Bootstrap-react provides the bootstrap grid system to be easily reused.

```
<Container>
    <Row>
        <Col>1 of 2</Col>
        <Col>2 of 2</Col>
    </Row>
    <Row>
        <Col>1 of 3</Col>
        <Col>2 of 3</Col>
        <Col>3 of 3</Col>
    </Row>
</Container>
```

- And all the components integrated in React

Home / Library / Data

Primary Secondary Success Warning Danger Info Light Dark

```
<Container>
    <Row>
        <Col xs={12} md={8}>
            xs=12 md=8
        </Col>
        <Col xs={6} md={4}>
            xs=6 md=4
        </Col>
    </Row>
    <Row>
        <Col xs={6} md={4}>
            xs=6 md=4
        </Col>
        <Col xs={6} md={4}>
            xs=6 md=4
        </Col>
        <Col xs={6} md={4}>
            xs=6 md=4
        </Col>
    </Row>
    <Row>
        <Col xs={6}>xs=6</Col>
        <Col xs={6}>xs=6</Col>
    </Row>
</Container>
```

Internet Applications Design and Implementation

2020 - 2021

(Lab class 8)

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 9
React

Lab class 9 - React

- Lab Objective: Create a filtered list of students
- Lab Protocol:
  1. Create a React App with typescript
  2. Add a proxy configuration to your server at port :8080
  3. Add some seed data to the server database (students)
  4. Implement a list of students that loads from your server (use openAPI generator to create the interface code)
  5. Create a filter on the list of students on the client side
  6. Adapt the list to filter on the server side.
  7. Use Bootstrap react implementation to change the layout of the list

Lab class 9 - React

- Create a react app with typescript, be sure to have installed create-react-app.
  - help here: https://create-react-app.dev/docs/adding-typescript/
  - command: `create-react-app pets –typescript`
  - open the folder on webstorm or another IDE.

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import * as serviceWorker from './serviceWorker';

ReactDOM.render( element: <App />, document.getElementById( elementId: 'root'));
```

Lab class 9 - React

- Add a proxy configuration to your server at port :8080 so to avoid CORS error in development time.

```
{
  "name": "pets",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@types/jest": "24.0.22",
    "@types/node": "12.12.7",
    "@types/react": "16.9.11",
    "@types/react-dom": "16.9.4",
    "react": "^16.11.0",
    "react-dom": "^16.11.0",
    "react-scripts": "3.2.0",
    "typescript": "3.7.2"
  },
  "scripts": {"start": "react-scripts start"...},
  "eslintConfig": {
    "extends": "react-app"
  },
  "browserslist": {...},
  "proxy":"http://localhost:8080"
}
```

Lab class 9 - React

- Add some seed data to the server database

```
pets — -zsh — 80×24
jrcs@JoaoCosSecosMac pets % http :8080/pets
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Date: Tue, 12 Nov 2019 10:46:58 GMT
Transfer-Encoding: chunked

[
    {
        "id": 1,
        "name": "pantufas",
        "species": "dog"
    },
    {
        "id": 2,
        "name": "bigodes",
        "species": "cat"
    }
]

jrcs@JoaoCosSecosMac pets %
```

Lab class 9 - React

- Implement a list of resources that loads from your server, start with…

```
import React from 'react';
import './App.css';

interface Pet { name:String, species: String }

const PetList = (props:{ pets: Pet[] }) => {

  let petlist = props.pets.map( callbackfn: pet => <li>pet.name</li>);

  return <ul> {petlist} </ul>
};

const App: React.FC = () => <PetList pets={[{name:"pantufas", species:"dog"}]}/>;

export default App;
```

Lab class 9 - React

- Continue on your own:
  - Create a filter on the list of pets on the client side
  - Adapt the list to filter on the server side.
  - Use Bootstrap react implementation to change the layout of the list

Internet Applications Design and Implementation

2020 - 2021

(Lecture 9 - Modelling Interface & Interaction - React meets IFML)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

How do you specify interaction in a user interface?

How do you specify its views?

How do you specify navigation actions?

How do you specify the data needed?

From navigation? from server?

You don’t!

Most people use mocks only! or worse…

They just code ahead!

<a href=“gotohere”>go to here</a>

<form action=“jumptohere”…>…</form>

Outline

- Interaction Flow Modelling Language (IFML)
- IFML by example
- From IFML to React
- From User Stories to IFML to React

Internet Applications Design and Implementation

2020 - 2021

(Lecture 9 - Part 1 - IFML)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

The UI Design Problem

Costly and
Inefficient process

Complexity of
user interfaces (UIs)

Ineffective
tools

Manual
development

No MDE
technology

Solution: Abstraction of content and flow

Solution: Abstraction of content and flow

The Interaction Flow Modeling Language

The new OMG Standard for integrating the front-end design in your system and enterprise models

- OMG (Object Management Group) standard (since 03/2013)

- To express content, user interaction, and control behavior of front-end apps

Practical Results of Having a Standard

- An official metamodel/grammar of the language which describes the semantics of and relations between the modelling constructs
- A graphical concrete syntax for the interaction flow notation which provides an intuitive representation of the user interface composition, interaction and control logic for the front-end designer
- A UML Profile consistent to the metamodel
- An interchange format between tools using XMI
- All this, specified through standard notations themselves

The UI Design solution: IFML

Covered aspects

- Multiple views for the same application
- Visualization and input of data, and production of events
- Components independent of concrete widgets and presentation
- Interaction flow, initiated by the user or by external events
- Modularization of the model (design-time containers for reuse purpose)
- User input validation and constraints, according to OCL or other existing constraint languages

IFML Essentials

https://www.omg.org/spec/IFML/1.0/PDF

Internet Applications Design and Implementation

2020 - 2021

(Lecture 9 - Part 2 - IFML by Example)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

The design of UI usually starts with mockups (ifml.org)

- IFML defines the behaviour and helps defining the components of UI parts.
- Start with a scenario of a book store and its shopping cart. The UI starts with a list of categories, products, and the corresponding details.
- The shopping process continues by asking user detailed information and connecting to the bank to process the payment.

https://www.ifml.org/wp-content/uploads/IFML-Bookstore-Example.pdf

IFML by Example

Consider the main screens of a sample application.
Only visual structure is defined.

https://www.ifml.org/wp-content/uploads/IFML-Bookstore-Example.pdf

IFML navigation between pages

Flow between screens is defined using IFML. Data transmitted is also defined here.

Events, interactions, actions

When a user “buys” a book

The event in the “Add to cart” button

Events - Interactive interfaces

When landing on the shopping cart page

Events - Interactive interfaces

Events - Interactive interfaces

Composition and subcomponents

Summary

- Mock up screens and components are the main design tool for user interfaces
- IFML complements mock ups with behaviour: data passing, event handling, action triggering
- IFML is hierarchical and compositional

Internet Applications Design and Implementation

2020 - 2021

(Lecture 9 - Part 3 - From IFML to React)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

IFML Syntax – View Components: Details

```
const ArtistDetails = (props:{artist:Artist}) =>
    <div>
        <p>First Name: {props.artist.firstName}</p>
        <p>Last Name: {props.artist.lastName}</p>
        <img src={props.artist.photo} />
    </div>
```

- User defines:
  - Type/Entity/Interface (Artist in this case)
  - Shown attributes (e.g. id, name, photo, etc.)
  - Value to be shown ("`artist = ?`")
- Also available *Multiple Details* and *Scroller*

IFML Syntax – View Components: Details

```
const ArtistDetails = (props:{artistId:number}) => {
    const [ artist, setArtist] = useState<Artist | null>(null);
    getData<Artist>(`/artists/${props.artistId}`)
        .then( data => data && setArtist(data));

    return <>
        {artist &&
            <div>
                <p>First Name: {artist.firstName}</p>
                <p>Last Name: {artist.lastName}</p>
                <img src={artist.photo}/>
            </div>}
        { !artist && <div>No artist</div>}
        </>;
};
```

- Value to be shown can be indirect
  if “oid = ?” is the parameter, then search the entry with the given oid

IFML Syntax – View Components: List

```
const ArtistListItem = (props:{artist:Artist}) =>
    <li key={props.artist.id}> {props.artist.firstName} {props.artist.lastName}</li>;

const ArtistIndex = (props:{artists:Artist[]}) =>
    <ul>
        { props.artists.map((artist:Artist) => <ArtistListItem artist={artist}/>)}
    </ul>;
```

- User defines:
  - Type/Entity/Interface (Artist in this case)
  - Shown attributes (e.g. id, name, etc.)
  - Selection criteria (… where X = Y)
- Also available *Checkable List*, *Hierarchical List*, etc.

IFML Syntax – View Components: Form

```
const AlbumSearch = (props:{loadAlbums:(title:string, year:string) => void}) => {
    const [ searchTitle, setSearchTitle ] = useState("");
    const [ searchYear, setSearchYear ] = useState("");

    useEffect(() => props.loadAlbums(searchTitle, searchYear), [searchTitle, searchYear]);

    let handleTitle = (e:ChangeEvent<HTMLInputElement>) => setSearchTitle(e.target.value);
    let handleYear = (e:ChangeEvent<HTMLInputElement>) => setSearchTitle(e.target.value);

    return <form>
            <label> Title: <input type="text" onChange={handleTitle}/></label>
            <label> Year: <input type="text" onChange={handleYear}/></label>
        </form>;
};
```

- User defines:
  - Entity, optionally (Album in this case)
  - Fields (e.g. title, year, etc.)
- Also available *Multiple Form*

IFML Syntax – Containers: Page

- Containers to hold other constructors (including other pages - nesting)
- Not necessarily mapped into web pages

IFML Syntax – Containers: Alternative Page

- Container to hold other pages in XOR mode
- It either contains Album or Artist, not both at the same time (depends on navigation)

IFML Syntax – (User) Navigation Flow

```
<Link to={`${this.props.id}`}>{"More details about "+this.props.firstName+" "+this.props.lastName}</Link>
```

- User defines:
  - Source (from where to navigate)
  - Target (to where to navigate)
  - Binding parameters/values
- Also available *OK Flow*, and *KO Flow*

IFML Syntax – Data Flow

- - - - - ▶

```
<ArtistDetails artist={artist} />
```

- User defines:
  - Source (from where to navigate)
  - Target (to where to navigate)
  - Binding parameters/values

IFML Syntax – Parameters Binding

- User defines:
  - Source parameter/value (SelectedArtist in this case)
  - Target parameter/value (AnArtist in this case)
  - Data flows from source to target
  - Usually associated with flows (user navigation flow or data flow)

IFML Syntax – Parameters Binding

IFML Syntax – Parameters Binding

```
const ArtistIndex = (props:{artists:Artist[], selectArtist:(idx:number)=>void}) =>
    <ul>
        { props.artists.map((artist:Artist) => <ArtistListItem artist={artist}/>)}
    </ul>;

const ArtistIndexWithDetails = (props:{artists:Artist[]}) => {
    const [selectedArtist, setSelectedArtist] = useState<number | null>(null);
    return <>
            <ArtistIndex artists={props.artists} selectArtist={setSelectedArtist}/>
            { selectedArtist && <ArtistDetails artistId={artists[selectedArtist]}/> }
           </>;
};
```

IFML Syntax – Events

- Created when user clicks on something
  - To select an element from a list
  - To navigate
  - Etc.

♪ All Together Now, All Together Now ♪

IFML by example

IFML by example

IFML – Adding Details to View Components

View Component Parts:

- Data binding
- Parameters

More IFML Syntax – Actions

- Generic actions can be defined by the user
- Tool support adds a few very useful (next slide)

More IFML Syntax – Actions: Create

- Create
  - Creates an Entity entry (user selects entity)
  - Receives necessary data through flow and parameter binding

More IFML Syntax – Actions: Create

IFML by example

IFML – Subtyping Events

OnSelect event

OnSubmit event

Catching events

Throwing events

IFML – Subtyping Components

And as many others as you want!

IFML by example – mailbox

ActivationExpression, SubmitEvent, Event generation

IFML example – online payment

IFML – Modules

IFML – (Re)using Modules

IFML by example – email

intra-component events and flows

Tools & docs

- IFML page: ifml.org
- Eclipse Modelling Tools : http://www.eclipse.org/downloads/packages/
- IFML Eclipse editor: http://ifml.github.io/

Internet Applications Design and Implementation

2020 - 2021

(Lecture 9 - Part 4 - An App end-to-end)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Example for today: Shared Task List

- A project management app where users have a backlog, can create tasks, divide them into sprints, assign them to users, mark them as completed, and see a variety of statistics about tasks and sprints.

Roadmap

1. User stories define a user centred application
2. IFML specifications to specify the (user stories) interactions and components.
3. React components implement each one of the IFML components and views.
4. Interface Mockups define the style and final composition of the UI.

User stories (I)

1. A user creates a new **task** given a name, a description, and due date, and sees the result in the **backlog**.
2. A user creates a new **sprint** given a name, a description, and a due date, and sees the result in the list of sprints.
3. A user selects a task (from the backlog or a sprint), assigns it to a (another) sprint and sees its name in the sprint task list.
4. A user selects a task (from the backlog or a sprint), and sees its name, description, and dates.
5. A user selects a task (from the backlog or a sprint), assigns it to a (another) user and sees the result in the details of the task and on the original list.
6. A user selects a task (from the backlog or a sprint), completes it (in a given date) and sees the end date in original list of tasks.

User Stories (II)

7. A user opens its home page, selects a sprint, and sees the sprint details, statistics and task list.

8. A user opens its home page, selects a user, and sees the user details, statistics and tasks list.

9. A user opens its home page, selects the backlog and sees the task list.

10. A user opens its home page, searches the backlog using a text and task properties, and sees the task list filtered by the given criteria.

IFML Specification

- A user creates a new **task** given a name, a description, and due date, and sees the result in the **backlog**.

IFML Specification

- A user creates a new **sprint** given a name, a description, and a due date, and sees the result in the list of sprints.

IFML Specification

- A user selects a task from the backlog, assigns it to a (another) sprint and sees its name in the sprint task list.

IFML Specification

- A user selects a task from a sprint, assigns it to a (another) sprint and sees its name in the sprint task list.

IFML Specification

- A user selects a task from a sprint, and sees its name, description, and dates.

IFML Specification

- A user selects a task from a sprint, assigns it to a (another) user and sees the result in the details of the task and on the original list.
-

IFML Specification

- A user selects a task from a sprint, completes it (in a given date) and sees the end date in original list of tasks.

IFML Specification

- A user opens its home page, selects a sprint, and sees the sprint details, statistics and task list.

IFML Specification

- A user opens its home page, selects a user, and sees the user details, statistics and tasks list.

IFML Specification

- A user opens its home page, selects the backlog and sees the task list.

IFML Specification

- A user opens its home page, searches the backlog using a text and task properties, and sees the task list filtered by the given criteria.

IFML Specification: The full picture

React structure

```
import * as React from "react";
import { BrowserRouter as Router, Route, Link } from "react-router-dom";

export interface HelloProps { compiler: string; framework: string; }

export const Dashboard = () => <div>Dashboard</div>

export const BackLog = () => <div>BackLog</div>

export const Sprints = () => <div>Sprints</div>

export const Users = () => <div>Users</div>

export const App = () =>
    <Router>
        <div className="container">
            <h1>Task List</h1>
            <div>
                <ul>
                    <li><Link to="/"> Home </Link></li>
                    <li><Link to="/backlog"> BackLog </Link></li>
                    <li><Link to="/sprints"> Sprints </Link></li>
                    <li><Link to="/users"> Users </Link></li>
                </ul>
            </div>

            <div>
                <div>
                    <Route exact path="/" component={Dashboard}/>
                    <Route path="/backlog" component={BackLog}/>
                    <Route path="/sprints" component={Sprints}/>
                    <Route path="/users" component={Users}/>
                </div>
            </div>
        </div>
    </Router>
```

React structure

```
const SearchCriteria = () =>
    <form>
        <input type="text" name="search"/>
        <input type="submit" value="Search"/>
    </form>

const SeacheableUserTaskList = () =>
    <div>
        <TaskList/>
        <SearchCriteria/>
    </div>

const Dashboard = () =>
    <div>
        <h1>Dashboard</h1>
        <SprintList/>
        <UserList/>
        <SeacheableUserTaskList/>
    </div>
```

```
const UserList = () =>
    <div>
        <h2>Users</h2>
        <ul>
            <li>User 1</li>
            <li>User 2</li>
            <li>User 3</li>
            <li>User 4</li>
        </ul>
    </div>

const SprintList = () =>
    <div>
        <ul>
            <li>Sprint 1</li>
            <li>Sprint 2</li>
            <li>Sprint 3</li>
            <li>Sprint 4</li>
        </ul>
    </div>

const TaskList = () =>
    <div>
        <h2>Tasks</h2>
        <ul>
            <li>Task 1</li>
            <li>Task 2</li>
            <li>Task 3</li>
            <li>Task 4</li>
        </ul>
    </div>
```

React structure

```
export class App extends Component<any,any> {

    constructor(props: any) {
        super(props);
        this.state = {
            sprints: [],
            tasks: []
        }
    }

    render() {
        return (
            <Router>
                <div className="container">
                    <h1>Task List</h1>
                    <div>
                        <ul>
                            <li><Link to="/"> Home </Link></li>
                            <li><Link to="/sprints"> Sprints </Link></li>
                            <li><Link to="/backlog"> BackLog </Link></li>
                            <li><Link to="/users"> Users </Link></li>
                        </ul>
                    </div>

                    <div>
                        <div>
                            <Route exact path="/"
                                   component={() => <DashboardPage sprints={this.state.sprints}/>}/>
                            <Route path="/sprints" component={Sprints}/>
                            <Route path="/backlog" component={BackLogPage}/>
                            <Route path="/users" component={UsersPage}/>
                        </div>
                    </div>
                </div>
            </Router>);
    }
}
```

Summary

Pros and Cons of using specs and frameworks

- Formal Design and Method
- Communication between stakeholders
- Documentation for future reference
  - It's important to use tools that sync specs and code
- Maintenance and evolution
  - Allows the analysis of the impact of code evolutions

Internet Applications Design and Implementation

2020 - 2021

(Lecture 10 - State Management. Flux&Redux: Design Patterns to the rescue)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)
Jácome Cunha (jacome@fct.unl.pt)
João Leitão (jc.leitao@fct.unl.pt)

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

React

- React implements a presentation layer
- React Pros:
  - Highly structured and stable applications
  - Highly Composable and reusable UI components
  - Loose coupling of components (through binding and render)
  - Great match with specification languages like IFML.
- React Cons:
  - There is not a great way to connect to a back-end.
  - Poor scalability with relation to the size of state and number of actions needed (coupling rises).

(Recall) Identify the UI minimal complete state

- Is it passed in from a parent via props? If so, it probably isn't state.
- Does it remain unchanged over time? If so, it probably isn't state.
- Can you compute it based on any other state or props in your component? If so, it isn't state.

(Recall) Find the state component by Q&A

For each piece of state in your application:

- Identify every component that renders something based on that state.
- Find a common owner component (a single component above all the components that need the state in the hierarchy).
- Either the common owner or another component higher up in the hierarchy should own the state.
- If you can't find a component where it makes sense to own the state, create a new component simply for holding the state and add it somewhere in the hierarchy above the common owner component.

Problem?

- The main model entities for a React application will be placed on the state of the application.
  - Application class will be cluttered with Layout, State and State changing and sync code (controllers?)
  - The whole application will be (re)rendered on every state change (efficiency problem?)
  - Solution 1: functions passed through props.
  - Solution 2: controller object(s) with groups of functions will be passed through props.
  - These are “indirect pointers” in the ownership hierarchy!
  - Although React creates new objects and clones state well, the code coupling does not scale well, needs a lot of lifecycle management code.

Sketch

```
componentDidMount() {
    $.getJSON(this.TASKS_URL)
        .then( doneFilter: (data:Task[]) =>
                this.setState(
            f: (state:StateSig) =>
                this.setState( state: {...state, tasks:data})))
    $.getJSON(this.SPRINTS_URL)
        .then( doneFilter: (data:Sprint[]) =>
                this.setState(
            f: (state:StateSig) =>
                this.setState( state: {...state, sprints:data})))
    $.getJSON(this.USERS_URL)
        .then( doneFilter: (data:User[]) =>
                this.setState(
            f: (state:StateSig) =>
                this.setState( state: {...state, users:data})))
}
```

Sketch

```
addSprint = (sprint:Sprint) =>
    $.post(this.SPRINTS_URL, JSON.stringify(sprint))
        .then( doneFilter: ()=> this.setState( f: (state:any)=>({...state, sprints:[sprint,...state.sprints]})))

addTask = (task:Task) =>
    $.post(this.SPRINTS_URL, JSON.stringify(task))
        .then( doneFilter: ()=>(this.setState( f: (state:any)=>({...state, tasks:[task,...state.tasks]}))))
```

Solution: complete a (kind of) client-side MVC

- Flux pattern extends the React with a store and dispatcher components.
- Implements a uni-directional flow of data
  (controller does not intermediate the views)
  It's not really MVC or MVP.
- React components (views) bind to stored elements.
- Actions (store functions) are triggered by the view
- Actions get dispatched to have effect on a store

Flux

Flux pattern

- diagram

Flux

A Store is an EventEmitter (“Subject”)

```
import { EventEmitter } from "events"
import {Task} from "../Tasks";


class TasksStore extends EventEmitter {
    tasks:Task[]

    constructor() {
        super()
        this.tasks = [
            {id: 1, name:"Go shopping", dueDate:"2017-10-23"},
            {id: 2, name:"Do the dishes", dueDate:"2017-10-22"},
            {id: 3, name:"Do homework", dueDate:"2017-10-21"},
        ] // Later will have to be managed from the server
    }

    getAllTasks() {
        return this.tasks;
    }
}

const tasksStore = new TasksStore()

export default tasksStore
```

A Store is an EventEmitter (“Subject”)

```
class TasksStore extends EventEmitter {
    tasks:Task[]
    count:number

    constructor() {...}

    createTask(name:string, dueDate:string) {
        this.tasks.push({id: this.count++, name, dueDate})
        this.emit( event: "change")
    }

    getAllTasks() {...}
}
```

App:

```
componentWillMount() {
    tasksStore.on( event: "change",
        listener: () => this.setState(
            f: (state:StateSig) => ({...state, tasks:tasksStore.getAllTasks()})
        ))
}
```

Flux - connect with server

```
class TasksStore extends EventEmitter {
    private TASKS_URL: "/tasks"

    tasks:Task[]
    count:number

    constructor() {...}

    createTask(task:Task) {
        this.tasks.push({...task, id: this.count++})
        $.post(this.TASKS_URL,JSON.stringify(task))
            .then( doneFilter: ()=>this.emit( event: "change"))
            .fail( failCallback: () => this.emit( event: "TASK_ERROR"))
    }

    getAllTasks() {...}
}
```

Redux

React

State

React Component

Side-effects

Props

λ Callbacks

Redux

- Implements a variant of a flux pattern.
- Based on functional state transformations. The state is read-only, can only be changed by action dispatching.
- A store that implements the observer pattern.
- Ingredients are Actions, Store, Reducers.

https://redux.js.org/introduction/getting-started

```
import {Action, createStore} from 'redux'

function counter(state = 0, action:Action) {
    switch (action.type) {
        case 'INCREMENT':
            return state + 1;
        case 'DECREMENT':
            return state - 1;
        default:
            return state
    }
}

let store = createStore(counter);

store.subscribe(() => console.log(store.getState()));

store.dispatch({ type: 'INCREMENT'});
store.dispatch({ type: 'INCREMENT'});
store.dispatch({ type: 'DECREMENT'});
```

Redux - reducers

```
counter(undefined, {type:'INCREMENT'}) = 1
counter(1, {type:'INCREMENT'}) = 2
counter(2, {type:'DECREMENT'}) = 1
```

Redux - reducers

- The state can be compositionally built from smaller pieces.
- Reducers can be combined to form a more elaborate state.

```
class ToDo { text:string; completed:boolean;
    constructor(text:string, completed: boolean) {
        this.text = text; this.completed = completed;
    }
}
interface AddTodoAction extends Action { text:string }
interface CompleteTodoAction extends Action { index:number }

function todos(state = [] as ToDo[], action:Action): ToDo[] {
    switch (action.type) {
        case 'ADD_TODO':
            let a1 = action as AddTodoAction;
            return [...state, new ToDo(a1.text, false)]
        case 'COMPLETE_TODO':
            let a2 = action as CompleteTodoAction;
            return state.map((todo:ToDo, index) => {
                return index == a2.index ? {...todo, completed:true}: todo
            });
        default:
            return state
    }
}
```

Redux - reducers

- The state can be compositionally built from smaller pieces.
- Reducers can be combined to form a more elaborate state.

```
interface SetFilterAction extends Action { filter: string }

function filter(state = 'SHOW_ALL', action:Action): string {
    switch (action.type) {
        case 'SET_FILTER':
            return (action as SetFilterAction).filter;
        default:
            return state;
    }
}

const reducer = combineReducers({ todos, filter });
const todoStore = createStore(reducer);

todoStore.subscribe(() => console.log(todoStore.getState()));

todoStore.dispatch({type:'ADD_TODO', text:'Complete the project'});
todoStore.dispatch({type:'ADD_TODO', text:'Start the project'});
todoStore.dispatch({type:'ADD_TODO', text:'Submit the project'});
todoStore.dispatch({type:'COMPLETE_TODO', index:0});
todoStore.dispatch({type:'COMPLETE_TODO', index:1});
todoStore.dispatch({type:'COMPLETE_TODO', index:2});
```

Redux - reducers

- The state can be compositionally built from smaller pieces.
- Reducers can be combined to form a more elaborate state.

```
[ ToDo { text: 'Complete the project', completed: false } ]

[ ToDo { text: 'Complete the project', completed: false },
  ToDo { text: 'Start the project', completed: false } ]

[ ToDo { text: 'Complete the project', completed: false },
  ToDo { text: 'Start the project', completed: false },
  ToDo { text: 'Submit the project', completed: false } ]

[ { text: 'Complete the project', completed: true },
  ToDo { text: 'Start the project', completed: false },
  ToDo { text: 'Submit the project', completed: false } ]

[ { text: 'Complete the project', completed: true },
  { text: 'Start the project', completed: true },
  ToDo { text: 'Submit the project', completed: false } ]

[ { text: 'Complete the project', completed: true },
  { text: 'Start the project', completed: true },
  { text: 'Submit the project', completed: true } ]
```

Redux-react

Presentational Components

- React and redux can be combined by having a presentational component, extended with connections to the state generated automatically.

```
interface ToDoItemInterface {text:string, completed:boolean, onClick:() => void}
const ToDoItem = ({text, completed, onClick}:ToDoItemInterface) =>
    <li
        onClick={onClick}
        style={{textDecoration: completed ? 'line-through': 'none'}}
    >
        {text}
    </li>;

interface ToDoListInterface { todos: ToDo[], onToDoClick: (index:number) => void}
const ToDoList = ({todos, onToDoClick}:ToDoListInterface) =>
    <ul>
        { todos.map( (todo, index) => {
            console.log(index);
            return <ToDoItem key={index} {...todo} onClick={() => onToDoClick(index) }/>
        })}
    </ul>;
```

Presentational Components

- React and redux can be combined by having a presentational component, extended with connections to the state generated automatically.

```
const mapStateToProps =
  (state:ToDoAppStateInterface) => ({todos: filterTodos(state.todos, state.filter)}};

const completeToDo = (index:number) => ({ type:'COMPLETE_TODO', index});
const mapDispatchToProps = (dispatch:(action:Action)=>void) =>
    ({ onToDoClick: (id:number) => { dispatch(completeToDo(id)) }});

const VisibleToDoList = connect(mapStateToProps, mapDispatchToProps)(ToDoList);

const ToDoListWithStore = () =>
    <Provider store={todoStore}>
       <VisibleToDoList/>
    </Provider>;
```

Async actions

- All changes to the s
- Integration with asy

```
class Pet { name:string; constructor(name:string) { this.name = name; }}
interface PetState { pets: Pet[] }

const ADD_PET = 'ADD_PET';
const REQUEST_PETS = 'REQUEST_PETS';
const RECEIVE_PETS = 'RECEIVE_PETS';

const addPet = (name:string) => ({type: REQUEST_PETS, name:string});

interface AddPetAction extends Action { name:string }

function petsReducer(pets = [] as Pet[], action:Action):Pet[] {
    switch (action.type) {
        case ADD_PET:
            let addPet = action as AddPetAction;
            return [...pets, new Pet(addPet.name)];
        default:
            return pets
    }
}
```

Async actions

- All changes
- Integration v

```
const requestPets = () => ({type: REQUEST_PETS});
const receivePets = (data:Pet[]) => ({type: RECEIVE_PETS, data:data});

function fetchPets() {
    return (dispatch) => {
        dispatch(requestPets());
        return fetch("/pets")
            .then( response => response.json())
            .then( data => dispatch(receivePets(data)))
    }
}

const loggerMiddleware = createLogger();

const store =
    createStore( combineReducers({pets: petsReducer}),
                 applyMiddleware( thunkMiddleware, loggerMiddleware ) );
store.dispatch(fetchPets()).then(() => console.log(store.getState()));
```

Further readings

- https://redux.js.org/introduction/getting-started
- https://react-redux.js.org/
- https://egghead.io/courses/getting-started-with-redux
- https://react-redux.js.org/next/api/hooks
- https://egghead.io/lessons/react-redux-the-single-immutable-state-tree
- https://redux.js.org/advanced/async-actions