

# Spring Boot Rest

- REST
- Spring Framework
  - Spring Boot
  - Web
  - JPA → JDBC → Oracle DB
- Progetto di riferimento
  - <https://github.com/egalli64/sbr>

# REST

- **Representational State Transfer** – Roy Fielding (2000)
- Stile architetturale per la creazione di Web Service
  - Un web service che segue lo stile REST è detto RESTful
- Uso di un insieme uniforme e predefinito di operazioni stateless
  - Client e server si scambiano messaggi usando il protocollo HTTP
  - Nella URL di base, la risorsa specifica la collezione di entità su cui si lavora:
    - <https://example.com/coders>
  - Può essere completata dall'id di una risorsa specifica
    - <https://example.com/coders/42>

# Request / response REST

- Request
  - Metodi HTTP: GET, POST, ...
  - URL: indica la risorsa di riferimento
  - Header e body: informazioni aggiuntive (JSON)
- Response
  - Status: 200, 404, ...
  - Header e body: informazioni aggiuntive (JSON)
    - Ad es., per caching: max-age

# Metodi HTTP REST → CRUD

- Comportamento differente se URL base o con id della risorsa
  - Operazione sulla collezione o sulla specifica risorsa (eccezione: POST)
- **POST** → CREATE: creazione una nuova risorsa, ne ritorna (almeno) l'id
  - Se è stato passato un id, è usato per la creazione
- **GET** → READ: lettura di collezione / risorsa
- **PUT** → UPDATE: aggiornamento di collezione / risorsa
  - Se non esiste, simile a POST
- PATCH → UPDATE: aggiornamento parziale di collezione / risorsa
  - Se non esiste (collezione / risorsa) potrebbe essere simile a POST
- **DELETE** → DELETE: eliminazione di collezione / risorsa
- L'implementazione REST di GET, PUT, DELETE deve essere idempotente
  - GET deve anche essere *safe*, non può modificare lo stato del server

# Spring Framework

- Nato nel 2003 come alternativa allo sviluppo Java Enterprise via J2EE
  - <https://spring.io/>
- Leggero e modulare, complementare a JEE
  - implementa molte tra le sue specifiche (Servlet, JPA, JMS, ...)
- Basato sul pattern Inversion of Control (IoC) aka Hollywood Principle
  - Dependency Injection (DI): proprietà inizializzate e gestite da Spring
- Spring Boot
  - Usa il paradigma convention over configuration
    - POM standard arricchito, uso intensivo di annotazioni
  - Applicazioni Java standalone con web server (Tomcat o Jetty) embedded

# Sviluppo con Spring

- Eclipse
  - <https://www.eclipse.org/downloads/packages/> → Enterprise Java
  - Plugin per Spring
    - Help > Eclipse Marketplace → Spring Tools 4
- Spring Tools 4 (STS)
  - <https://spring.io/tools>
- Altre soluzioni sono disponibili
  - IntelliJ IDEA – versione Ultimate
  - VS Code supporto diretto di Spring
  - ...

# Nuovo progetto Spring

- File > New > Project → Spring Boot > Spring Starter Project
  - Service URL: <https://start.spring.io>
  - Type: Maven, Java Version: 11
  - Packaging: jar, Language: Java
  - Dependencies
    - Web - Spring Web
    - JPA - Spring Data JPA
  - Nel POM, aggiungere la dependency JDBC

# application.properties

- In source/main/resources
- Configurazione dell'applicazione

```
spring.jpa.open-in-view=false
```

```
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver
```

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521/xepdb1
```

```
spring.datasource.username=me
```

```
spring.datasource.password=password
```

```
logging.level.com.example=TRACE
```



# Spring Boot Application

- La classe main di una applicazione Spring Boot può essere annotata `@SpringBootApplication`
  - Equivalente a `@Configuration`, `@EnableAutoConfiguration`, `@ComponentScan`
- Il suo main invoca `SpringApplication.run()`

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

# Boot Dashboard

- Configurazione del progetto (in local)
  - Right click sul nome del progetto, Open config
  - Spring Boot tab
    - Main type: la nostra SpringBootApplication
- Esecuzione
  - (Re)start o (Re)debug

# Entity

- **@Entity**
  - Java Bean che fa da DTO per il database, JPA assume una relazione con tabella omonima
- **@Table**
  - Nome della tabella da associare all'entity
- **@Id**
  - Specifica l'identificatore univoco dell'oggetto
- **@GeneratedValue**
  - Indica che la tabella può generare la PK automaticamente
    - MySQL richiede `strategy=GenerationType.IDENTITY`
    - Per usare una sequence Oracle, `GenerationType.Sequence` + `@SequenceGenerator`
- **@Column**
  - Nome proprietà → colonna della tabella, Spring traduce da camelCase Java a underscore SQL



# Entity con Sequence Oracle

```
@Entity
@Table(name = "CODERS")
public class Coder {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "CodGen")
    @SequenceGenerator(sequenceName = "CODER_SEQ", allocationSize = 1, name = "CodGen")
    @Column(name = "CODER_ID")
    private long id;

    // @Column(name = "FIRST_NAME")
    private String firstName;

    // etc.
}
```

valore generato da JPA  
strategy: sequenza fornita dal database  
generator: nome JPA

informazioni sul generatore  
sequenceName: nome ORACLE  
allocationSize: INCREMENT della sequenza  
name: nome JPA

# Repository

- @Repository
  - Interfaccia che definisce funzionalità di accesso al fornitore di dati  
Il codice del repository viene generato dal provider JPA
  - CrudRepository dichiara funzionalità CRUD di base, richiede:
    - Il tipo dell'entity che intendiamo utilizzare
    - Il tipo dell'identificatore univoco dell'entity
  - JpaRepository estende ulteriormente Repository

```
@Repository  
public interface CoderRepo extends JpaRepository<Coder, Long> {  
}
```

# Metodi in CrudRepository

- count()
- delete(T)
- deleteAll()
- deleteAll(Iterable<>)
- deleteById(ID)
- existsById(ID)
- findAll()
- findAllById(Iterable<>)
- findById(ID)
- save(S)
- saveAll(Iterable<S>)

# Controller e RestController

- Spring include nel controller dell'applicazione
  - le classi con annotazione `@Controller` (o derivate)
  - nel package della `@SpringBootApplication` e nei suoi subpackage
- `@RestController` specializza il comportamento per un Web Service RESTful
- `@CrossOrigin` permette il Cross-Origin Resource Sharing (CORS)

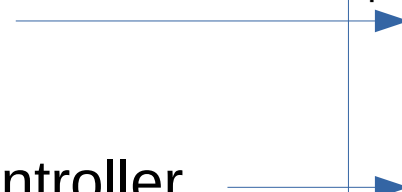
```
// ...  
@CrossOrigin(origins = "http://localhost:4200")  
@RestController  
public class CoderCtrl {  
    // ...  
}
```

# Repository in Controller

- Dependency Injection
  - Variabile di istanza Repository
  - Annotata @Autowired
  - Gestione delegata a Spring
- Metodo @GetMapping del controller
  - Usa il repository per accedere il database
  - Spring prende il valore ritornato
    - lo converte in una stringa JSON
    - lo usa come body della response

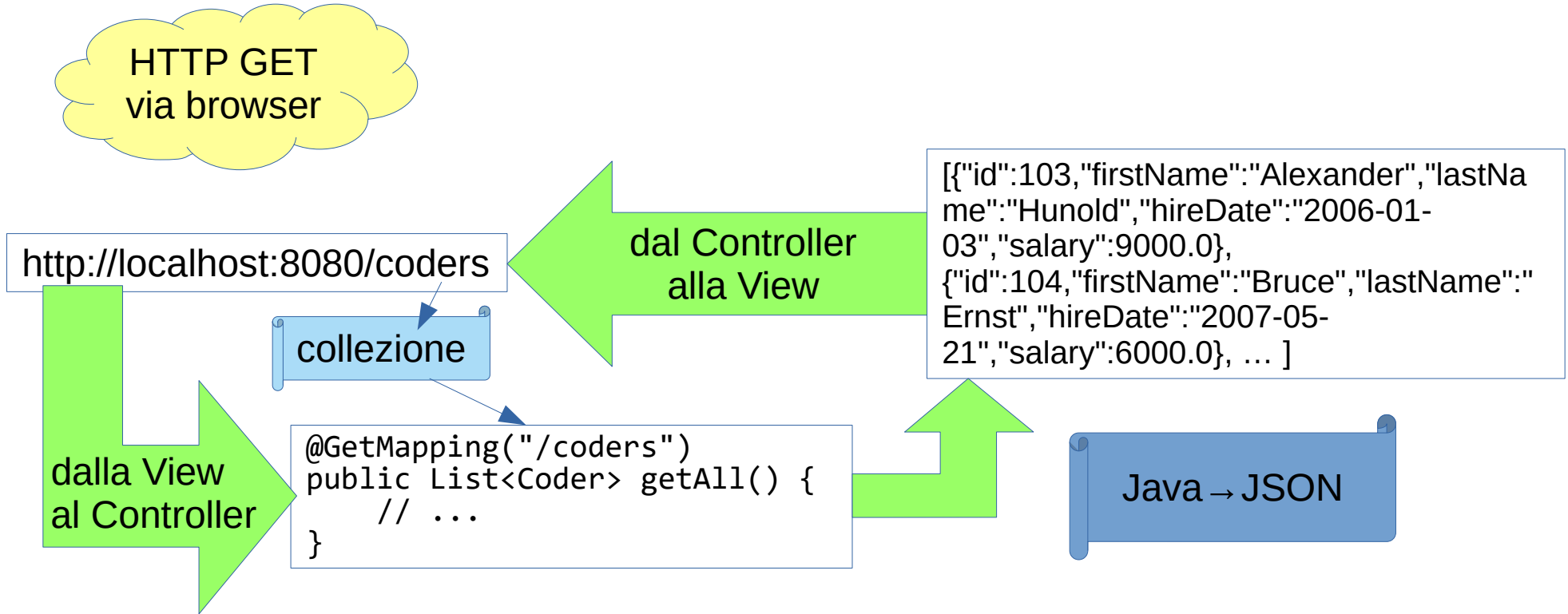
```
@RestController
public class CoderCtrl {
    @Autowired
    private CoderRepo repo;

    @GetMapping("/coders")
    List<Coder> getAll() {
        return repo.findAll();
    }
}
```





# Request / Response



# RequestBody e PathVariable

- Annotazioni sui parametri
  - Metodo XxxMapping (Get, ...)
  - Classe RestController
- RequestBody
  - Conversione da JSON
- PathVariable
  - Conversione da String

```
@PostMapping("/coders")  
public Coders create(@RequestBody Coders coders) {  
    // ...  
}
```

```
@GetMapping("/coders/{id}")  
public Coders get(@PathVariable Long id) {  
    // ...  
}
```

# Curl per testing (Windows)

- `curl -v localhost:8080/coders`
- `curl -v localhost:8080/coders/103`
- `curl -X POST -H "Content-Type: application/json" -d {"firstName\":\"X\",\"lastName\":\"Y\",\"hireDate\":\"2020-01-01\",\"salary\":\"9000.0\"} localhost:8080/coders`
- `curl -X PUT -H "Content-Type: application/json" -d {"firstName\":\"Z\",\"lastName\":\"Y\",\"hireDate\":\"2020-03-01\",\"salary\":\"9500.0\"} localhost:8080/coders/223`
- `curl -X PATCH -H "Content-Type: application/json" -d {"hireDate\":\"2020-09-01\"} localhost:8080/coders/223`
- `curl -X DELETE localhost:8080/coders/223`